



Intel® 64 and IA-32 Architectures Software Developer's Manual

Documentation Changes

October 2025

Notice: The Intel® 64 and IA-32 architectures may contain design defects or errors known as errata that may cause the product to deviate from published specifications. Current characterized errata are documented in the specification updates.

Document Number: 252046-081



Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

All product plans and roadmaps are subject to change without notice.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), <https://opensource.org/licenses/0BSD>. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Contents

Revision History	4
Preface	9
Summary Tables of Changes	10
Documentation Changes	11



Revision History

Revision	Description	Date
-001	Initial release	November 2002
-002	- Added 1-10 Documentation Changes. - Removed old Documentation Changes items that already have been incorporated in the published Software Developer's manual	December 2002
-003	- Added 9 -17 Documentation Changes. - Removed Documentation Change #6 - References to bits Gen and Len Deleted. - Removed Documentation Change #4 - VIF Information Added to CLI Discussion	February 2003
-004	- Removed Documentation changes 1-17. - Added Documentation changes 1-24.	June 2003
-005	- Removed Documentation Changes 1-24. - Added Documentation Changes 1-15.	September 2003
-006	Added Documentation Changes 16- 34.	November 2003
-007	- Updated Documentation changes 14, 16, 17, and 28. - Added Documentation Changes 35-45.	January 2004
-008	- Removed Documentation Changes 1-45. - Added Documentation Changes 1-5.	March 2004
-009	Added Documentation Changes 7-27.	May 2004
-010	- Removed Documentation Changes 1-27. - Added Documentation Changes 1.	August 2004
-011	Added Documentation Changes 2-28.	November 2004
-012	- Removed Documentation Changes 1-28. - Added Documentation Changes 1-16.	March 2005
-013	- Updated title. - There are no Documentation Changes for this revision of the document.	July 2005
-014	Added Documentation Changes 1-21.	September 2005
-015	- Removed Documentation Changes 1-21. - Added Documentation Changes 1-20.	March 9, 2006
-016	Added Documentation changes 21-23.	March 27, 2006
-017	- Removed Documentation Changes 1-23. - Added Documentation Changes 1-36.	September 2006
-018	Added Documentation Changes 37-42.	October 2006
-019	- Removed Documentation Changes 1-42. - Added Documentation Changes 1-19.	March 2007
-020	Added Documentation Changes 20-27.	May 2007
-021	- Removed Documentation Changes 1-27. - Added Documentation Changes 1-6	November 2007
-022	- Removed Documentation Changes 1-6 - Added Documentation Changes 1-6	August 2008
-023	- Removed Documentation Changes 1-6 - Added Documentation Changes 1-21	March 2009

Revision	Description	Date
-024	- Removed Documentation Changes 1-21 - Added Documentation Changes 1-16	June 2009
-025	- Removed Documentation Changes 1-16 - Added Documentation Changes 1-18	September 2009
-026	- Removed Documentation Changes 1-18 - Added Documentation Changes 1-15	December 2009
-027	- Removed Documentation Changes 1-15 - Added Documentation Changes 1-24	March 2010
-028	- Removed Documentation Changes 1-24 - Added Documentation Changes 1-29	June 2010
-029	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-29	September 2010
-030	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-29	January 2011
-031	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-29	April 2011
-032	- Removed Documentation Changes 1-29 - Added Documentation Changes 1-14	May 2011
-033	- Removed Documentation Changes 1-14 - Added Documentation Changes 1-38	October 2011
-034	- Removed Documentation Changes 1-38 - Added Documentation Changes 1-16	December 2011
-035	- Removed Documentation Changes 1-16 - Added Documentation Changes 1-18	March 2012
-036	- Removed Documentation Changes 1-18 - Added Documentation Changes 1-17	May 2012
-037	- Removed Documentation Changes 1-17 - Added Documentation Changes 1-28	August 2012
-038	- Removed Documentation Changes 1-28 - Add Documentation Changes 1-22	January 2013
-039	- Removed Documentation Changes 1-22 - Add Documentation Changes 1-17	June 2013
-040	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-24	September 2013
-041	- Removed Documentation Changes 1-24 - Add Documentation Changes 1-20	February 2014
-042	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-8	February 2014
-043	- Removed Documentation Changes 1-8 - Add Documentation Changes 1-43	June 2014
-044	- Removed Documentation Changes 1-43 - Add Documentation Changes 1-12	September 2014
-045	- Removed Documentation Changes 1-12 - Add Documentation Changes 1-22	January 2015
-046	- Removed Documentation Changes 1-22 - Add Documentation Changes 1-25	April 2015
-047	- Removed Documentation Changes 1-25 - Add Documentation Changes 1-19	June 2015

Revision	Description	Date
-048	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-33	September 2015
-049	- Removed Documentation Changes 1-33 - Add Documentation Changes 1-33	December 2015
-050	- Removed Documentation Changes 1-33 - Add Documentation Changes 1-9	April 2016
-051	- Removed Documentation Changes 1-9 - Add Documentation Changes 1-20	June 2016
-052	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-22	September 2016
-053	- Removed Documentation Changes 1-22 - Add Documentation Changes 1-26	December 2016
-054	- Removed Documentation Changes 1-26 - Add Documentation Changes 1-20	March 2017
-055	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-28	July 2017
-056	- Removed Documentation Changes 1-28 - Add Documentation Changes 1-18	October 2017
-057	- Removed Documentation Changes 1-18 - Add Documentation Changes 1-29	December 2017
-058	- Removed Documentation Changes 1-29 - Add Documentation Changes 1-17	March 2018
-059	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-24	May 2018
-060	- Removed Documentation Changes 1-24 - Add Documentation Changes 1-23	November 2018
-061	- Removed Documentation Changes 1-23 - Add Documentation Changes 1-21	January 2019
-062	- Removed Documentation Changes 1-21 - Add Documentation Changes 1-28	May 2019
-063	- Removed Documentation Changes 1-28 - Add Documentation Changes 1-34	October 2019
-064	- Removed Documentation Changes 1-34 - Add Documentation Changes 1-36	May 2020
-065	- Removed Documentation Changes 1-36 - Add Documentation Changes 1-31	November 2020
-066	- Removed Documentation Changes 1-31 - Add Documentation Changes 1-24	April 2021
-067	- Removed Documentation Changes 1-24 - Add Documentation Changes 1-30	June 2021
-068	- Removed Documentation Changes 1-30 - Add Documentation Changes 1-29	December 2021
-069	- Removed Documentation Changes 1-29 - Add Documentation Changes 1-18	April 2022
-070	- Removed Documentation Changes 1-18 - Add Documentation Changes 1-41	December 2022
-071	- Removed Documentation Changes 1-41 - Add Documentation Changes 1-23	March 2023

Revision	Description	Date
-072	- Removed Documentation Changes 1-23 - Add Documentation Changes 1-19	June 2023
-073	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-19	September 2023
-074	- Removed Documentation Changes 1-19 - Add Documentation Changes 1-20	December 2023
-075	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-20	March 2024
-076	- Removed Documentation Changes 1-20 - Add Documentation Changes 1-8	June 2024
-077	- Removed Documentation Changes 1-8 - Add Documentation Changes 1-27	October 2024
-078	- Removed Documentation Changes 1-27 - Add Documentation Changes 1-15	December 2024
-079	- Removed Documentation Changes 1-15 - Add Documentation Changes 1-17	March 2025
-080	- Removed Documentation Changes 1-17 - Add Documentation Changes 1-25	June 2025
-081	- Removed Documentation Changes 1-25 - Add Documentation Changes 1-33	October 2025

§



Preface

This document is an update to the specifications contained in the [Affected Documents](#) table below. This document is a compilation of device and documentation errata, specification clarifications and changes. It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools.

Affected Documents

Document Title	Document Number/ Location
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture</i>	253665
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L</i>	253666
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U</i>	253667
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference, V</i>	326018
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z</i>	334569
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1</i>	253668
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2</i>	253669
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3</i>	326019
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4</i>	332831
<i>Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model Specific Registers</i>	335592

Nomenclature

Documentation Changes include typos, errors, or omissions from the current published specifications. These will be incorporated in any new release of the specification.

Summary Tables of Changes

The following table indicates documentation changes which apply to the Intel® 64 and IA-32 architectures. This table uses the following notations:

Codes Used in Summary Tables

A violet change bar to left of table row indicates this erratum is either new or modified from the previous version of the document.

Documentation Changes(Sheet 1 of 2)

No.	DOCUMENTATION CHANGES
1	Updates to Chapter 1, Volume 1
2	Updates to Chapter 17, Volume 1
3	Updates to Chapter 21, Volume 1
4	Updates to Chapter 3, Volume 2A
5	Updates to Chapter 4, Volume 2B
6	Updates to Chapter 5, Volume 2C
7	Updates to Appendix A, Volume 2D
8	Updates to Chapter 2, Volume 3A
9	Updates to Chapter 3, Volume 3A
10	Updates to Chapter 5, Volume 3A
11	Updates to Chapter 6, Volume 3A
12	Updates to Chapter 7, Volume 3A
13	Updates to Chapter 12, Volume 3A
14	Updates to Chapter 13, Volume 3A
15	Updates to Chapter 16, Volume 3B
16	Updates to Chapter 17, Volume 3B
17	Updates to Chapter 19, Volume 3B
18	Updates to Chapter 24, Volume 3B
19	Updates to Chapter 26, Volume 3C
20	Updates to Chapter 27, Volume 3C
21	Updates to Chapter 28, Volume 3B
22	Updates to Chapter 29, Volume 3C
23	Updates to Chapter 30, Volume 3C
24	Updates to Chapter 31, Volume 3C
25	Updates to Chapter 32, Volume 3C
26	Updates to Chapter 33, Volume 3C

Documentation Changes(Sheet 2 of 2)

No.	DOCUMENTATION CHANGES
27	Updates to Chapter 34, Volume 3C
28	Updates to Chapter 35, Volume 3C
29	Updates to Chapter 37, Volume 3D
30	Updates Appendix A, Volume 3D
31	Updates Appendix B, Volume 3D
32	Updates Appendix C, Volume 3D
33	Updates to Chapter 2, Volume 4

Documentation Changes

Changes to the Intel® 64 and IA-32 Architectures Software Developer's Manual volumes follow, and are listed by chapter. Only chapters with changes are included in this document.

1. Updates to Chapter 1, Volume 1

Change bars and violet text show changes to Chapter 1 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- In Section 1.3.5, "A Syntax for CPUID, CR, and MSR Values", updated Figure 1-2.

The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture (order number 253665) is part of a set that describes the architecture and programming environment of Intel® 64 and IA-32 architecture processors. Other volumes in this set are:

- The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C & 2D: Instruction Set Reference (order numbers 253666, 253667, 326018, and 334569).
- The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C & 3D: System Programming Guide (order numbers 253668, 253669, 326019, and 332831).
- The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers (order number 335592).

The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, describes the basic architecture and programming environment of Intel 64 and IA-32 processors. The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D, describe the instruction set of the processor and the opcode structure. These volumes apply to application programmers and to programmers who write operating systems or executives. The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 3A, 3B, 3C, & 3D, describe the operating-system support environment of Intel 64 and IA-32 processors. These volumes target operating-system and BIOS designers. In addition, the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, addresses the programming environment for classes of software that host operating systems. The Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, describes the model-specific registers of Intel 64 and IA-32 processors.

1.1 INTEL® 64 AND IA-32 PROCESSORS COVERED IN THIS MANUAL

This manual set includes information pertaining primarily to the most recent Intel 64 and IA-32 processors, which include:

- Pentium® processors
- P6 family processors
- Pentium® 4 processors
- Pentium® M processors
- Intel® Xeon® processors
- Pentium® D processors
- Pentium® processor Extreme Editions
- 64-bit Intel® Xeon® processors
- Intel® Core™ Duo processor
- Intel® Core™ Solo processor
- Dual-Core Intel® Xeon® processor LV
- Intel® Core™ 2 Duo processor
- Intel® Core™ 2 Quad processor Q6000 series
- Intel® Xeon® processor 3000, 3200 series
- Intel® Xeon® processor 5000 series
- Intel® Xeon® processor 5100, 5300 series
- Intel® Core™ 2 Extreme processor X7000 and X6800 series
- Intel® Core™ 2 Extreme processor QX6000 series
- Intel® Xeon® processor 7100 series

- Intel® Pentium® Dual-Core processor
- Intel® Xeon® processor 7200, 7300 series
- Intel® Xeon® processor 5200, 5400, 7400 series
- Intel® Core™ 2 Extreme processor QX9000 and X9000 series
- Intel® Core™ 2 Quad processor Q9000 series
- Intel® Core™ 2 Duo processor E8000, T9000 series
- Intel Atom® processor family
- Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are built from 45 nm and 32 nm processes
- Intel® Core™ i7 processor
- Intel® Core™ i5 processor
- Intel® Xeon® processor E7-8800/4800/2800 product families
- Intel® Core™ i7-3930K processor
- 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series
- Intel® Xeon® processor E3-1200 product family
- Intel® Xeon® processor E5-2400/1400 product family
- Intel® Xeon® processor E5-4600/2600/1600 product family
- 3rd generation Intel® Core™ processors
- Intel® Xeon® processor E3-1200 v2 product family
- Intel® Xeon® processor E5-2400/1400 v2 product families
- Intel® Xeon® processor E5-4600/2600/1600 v2 product families
- Intel® Xeon® processor E7-8800/4800/2800 v2 product families
- 4th generation Intel® Core™ processors
- The Intel® Core™ M processor family
- Intel® Core™ i7-59xx Processor Extreme Edition
- Intel® Core™ i7-49xx Processor Extreme Edition
- Intel® Xeon® processor E3-1200 v3 product family
- Intel® Xeon® processor E5-2600/1600 v3 product families
- 5th generation Intel® Core™ processors
- Intel® Xeon® processor D-1500 product family
- Intel® Xeon® processor E5 v4 family
- Intel Atom® processor X7-Z8000 and X5-Z8000 series
- Intel Atom® processor Z3400 series
- Intel Atom® processor Z3500 series
- 6th generation Intel® Core™ processors
- Intel® Xeon® processor E3-1500m v5 product family
- 7th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series
- Intel® Xeon® Scalable Processor Family
- 8th generation Intel® Core™ processors
- Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series
- Intel® Xeon® E processors
- 9th generation Intel® Core™ processors
- 2nd generation Intel® Xeon® Scalable Processor Family

- 10th generation Intel® Core™ processors
- 11th generation Intel® Core™ processors
- 3rd generation Intel® Xeon® Scalable Processor Family
- 12th generation Intel® Core™ processors
- 13th generation Intel® Core™ processors
- 4th generation Intel® Xeon® Scalable Processor Family
- 5th generation Intel® Xeon® Scalable Processor Family
- Intel® Core™ Ultra 7 processors
- Intel® Xeon® 6 E-Core processors
- Intel® Xeon® 6 P-Core processors
- Intel® Series 2 Core™ Ultra processors

P6 family processors are IA-32 processors based on the P6 family microarchitecture. This includes the Pentium® Pro, Pentium® II, Pentium® III, and Pentium® III Xeon® processors.

The Pentium® 4, Pentium® D, and Pentium® processor Extreme Editions are based on the Intel NetBurst® microarchitecture. Most early Intel® Xeon® processors are based on the Intel NetBurst® microarchitecture. Intel Xeon processor 5000, 7100 series are based on the Intel NetBurst® microarchitecture.

The Intel® Core™ Duo, Intel® Core™ Solo and dual-core Intel® Xeon® processor LV are based on an improved Pentium® M processor microarchitecture.

The Intel® Xeon® processor 3000, 3200, 5100, 5300, 7200, and 7300 series, Intel® Pentium® dual-core, Intel® Core™ 2 Duo, Intel® Core™ 2 Quad, and Intel® Core™ 2 Extreme processors are based on Intel® Core™ microarchitecture.

The Intel® Xeon® processor 5200, 5400, 7400 series, Intel® Core™ 2 Quad processor Q9000 series, and Intel® Core™ 2 Extreme processors QX9000, X9000 series, Intel® Core™ 2 processor E8000 series are based on Enhanced Intel® Core™ microarchitecture.

The Intel Atom® processors 200, 300, D400, D500, D2000, N200, N400, N2000, E2000, Z500, Z600, Z2000, C1000 series are based on the Intel Atom® microarchitecture and supports Intel 64 architecture.

P6 family, Pentium® M, Intel® Core™ Solo, Intel® Core™ Duo processors, dual-core Intel® Xeon® processor LV, and early generations of Pentium 4 and Intel Xeon processors support IA-32 architecture. The Intel® Atom™ processor Z5xx series support IA-32 architecture.

The Intel® Xeon® processor 3000, 3200, 5000, 5100, 5200, 5300, 5400, 7100, 7200, 7300, 7400 series, Intel® Core™ 2 Duo, Intel® Core™ 2 Extreme, Intel® Core™ 2 Quad processors, Pentium® D processors, Pentium® Dual-Core processor, newer generations of Pentium 4 and Intel Xeon processor family support Intel® 64 architecture.

The Intel® Core™ i7 processor and Intel® Xeon® processor 3400, 5500, 7500 series are based on 45 nm Nehalem microarchitecture. Westmere microarchitecture is a 32 nm version of the Nehalem microarchitecture. Intel® Xeon® processor 5600 series, Intel Xeon processor E7 and various Intel Core i7, i5, i3 processors are based on the Westmere microarchitecture. These processors support Intel 64 architecture.

The Intel® Xeon® processor E5 family, Intel® Xeon® processor E3-1200 family, Intel® Xeon® processor E7-8800/4800/2800 product families, Intel® Core™ i7-3930K processor, and 2nd generation Intel® Core™ i7-2xxx, Intel® Core™ i5-2xxx, Intel® Core™ i3-2xxx processor series are based on the Sandy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E7-8800/4800/2800 v2 product families, Intel® Xeon® processor E3-1200 v2 product family and 3rd generation Intel® Core™ processors are based on the Ivy Bridge microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-4600/2600/1600 v2 product families, Intel® Xeon® processor E5-2400/1400 v2 product families and Intel® Core™ i7-49xx Processor Extreme Edition are based on the Ivy Bridge-E microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E3-1200 v3 product family and 4th Generation Intel® Core™ processors are based on the Haswell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® processor E5-2600/1600 v3 product families and the Intel® Core™ i7-59xx Processor Extreme Edition are based on the Haswell-E microarchitecture and support Intel 64 architecture.

The Intel Atom® processor Z8000 series is based on the Airmont microarchitecture.

The Intel Atom® processor Z3400 series and the Intel Atom® processor Z3500 series are based on the Silvermont microarchitecture.

The Intel® Core™ M processor family, 5th generation Intel® Core™ processors, Intel® Xeon® processor D-1500 product family and the Intel® Xeon® processor E5 v4 family are based on the Broadwell microarchitecture and support Intel 64 architecture.

The Intel® Xeon® Scalable Processor Family, Intel® Xeon® processor E3-1500m v5 product family and 6th generation Intel® Core™ processors are based on the Skylake microarchitecture and support Intel 64 architecture.

The 7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture and support Intel 64 architecture.

The Intel Atom® processor C series, the Intel Atom® processor X series, the Intel® Pentium® processor J series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont microarchitecture.

The Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series is based on the Knights Landing microarchitecture and supports Intel 64 architecture.

The Intel® Pentium® Silver processor series, the Intel® Celeron® processor J series, and the Intel® Celeron® processor N series are based on the Goldmont Plus microarchitecture.

The 8th generation Intel® Core™ processors, 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on the Coffee Lake microarchitecture and support Intel 64 architecture.

The Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series is based on the Knights Mill microarchitecture and supports Intel 64 architecture.

The 2nd generation Intel® Xeon® Scalable Processor Family is based on the Cascade Lake product and supports Intel 64 architecture.

Some 10th generation Intel® Core™ processors are based on the Ice Lake microarchitecture, and some are based on the Comet Lake microarchitecture; both support Intel 64 architecture.

Some 11th generation Intel® Core™ processors are based on the Tiger Lake microarchitecture, and some are based on the Rocket Lake microarchitecture; both support Intel 64 architecture.

Some 3rd generation Intel® Xeon® Scalable Processor Family processors are based on the Cooper Lake product, and some are based on the Ice Lake microarchitecture; both support Intel 64 architecture.

The 12th generation Intel® Core™ processors are based on the Alder Lake performance hybrid architecture and support Intel 64 architecture.

The 13th generation Intel® Core™ processors are based on the Raptor Lake performance hybrid architecture and support Intel 64 architecture.

The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture and supports Intel 64 architecture.

The 5th generation Intel® Xeon® Scalable Processor Family is based on Emerald Rapids microarchitecture and supports Intel 64 architecture.

The Intel® Core™ Ultra 7 processor is based on Meteor Lake performance hybrid architecture and supports Intel 64 architecture.

The Intel® Xeon® 6 E-core processor is based on Sierra Forest microarchitecture and supports Intel 64 architecture.

The Intel® Xeon® 6 P-core processor is based on Granite Rapids microarchitecture and supports Intel 64 architecture.

The Intel® Series 2 Core™ Ultra processor is based on Lunar Lake performance hybrid architecture and supports Intel 64 architecture.

IA-32 architecture is the instruction set architecture and programming environment for Intel's 32-bit microprocessors. Intel® 64 architecture is the instruction set architecture and programming environment which is the superset of Intel's 32-bit and 64-bit architectures. It is compatible with the IA-32 architecture.

1.2 OVERVIEW OF VOLUME 1: BASIC ARCHITECTURE

A description of this manual's content follows:

Chapter 1 — About This Manual. Gives an overview of all volumes of the Intel® 64 and IA-32 Architectures Software Developer's Manual. It also describes the notational conventions in these manuals and lists related Intel manuals and documentation of interest to programmers and hardware designers.

Chapter 2 — Intel® 64 and IA-32 Architectures. Introduces the Intel 64 and IA-32 architectures along with the families of Intel processors that are based on these architectures. It also gives an overview of the common features found in these processors and brief history of the Intel 64 and IA-32 architectures.

Chapter 3 — Basic Execution Environment. Introduces the models of memory organization and describes the register set used by applications.

Chapter 4 — Data Types. Describes the data types and addressing modes recognized by the processor; provides an overview of real numbers and floating-point formats and of floating-point exceptions.

Chapter 5 — Instruction Set Summary. Lists all Intel 64 and IA-32 instructions, divided into technology groups.

Chapter 6 — Procedure Calls, Interrupts, and Exceptions. Describes the procedure stack and mechanisms provided for making procedure calls and for servicing interrupts and exceptions.

Chapter 7 — Programming with General-Purpose Instructions. Describes basic load and store, program control, arithmetic, and string instructions that operate on basic data types, general-purpose and segment registers; also describes system instructions that are executed in protected mode.

Chapter 8 — Programming with the x87 FPU. Describes the x87 floating-point unit (FPU), including floating-point registers and data types; gives an overview of the floating-point instruction set and describes the processor's floating-point exception conditions.

Chapter 9 — Programming with Intel® MMX™ Technology. Describes Intel MMX technology, including MMX registers and data types; also provides an overview of the MMX instruction set.

Chapter 10 — Programming with Intel® Streaming SIMD Extensions (Intel® SSE). Describes SSE extensions, including XMM registers, the MXCSR register, and packed single precision floating-point data types; provides an overview of the SSE instruction set and gives guidelines for writing code that accesses the SSE extensions.

Chapter 11 — Programming with Intel® Streaming SIMD Extensions 2 (Intel® SSE2). Describes SSE2 extensions, including XMM registers and packed double precision floating-point data types; provides an overview of the SSE2 instruction set and gives guidelines for writing code that accesses SSE2 extensions. This chapter also describes SIMD floating-point exceptions that can be generated with SSE and SSE2 instructions. It also provides general guidelines for incorporating support for SSE and SSE2 extensions into operating system and applications code.

Chapter 12 — Programming with Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Supplemental Streaming SIMD Extensions 3 (SSSE3), Intel® Streaming SIMD Extensions 4 (Intel® SSE4) and Intel® AES New Instructions (Intel® AES-NI). Provides an overview of the SSE3 instruction set, Supplemental SSE3, SSE4, AESNI instructions, and guidelines for writing code that access these extensions.

Chapter 13 — Managing State Using the XSAVE Feature Set. Describes the XSAVE feature set instructions and explains how software can enable the XSAVE feature set and XSAVE-enabled features.

Chapter 14 — Programming with Intel® AVX, FMA, and Intel® AVX2. Provides an overview of the Intel® AVX instruction set, FMA, and Intel® AVX2 extensions and gives guidelines for writing code that access these extensions.

Chapter 15 — Programming with Intel® AVX-512. Provides an overview of the Intel® AVX-512 instruction set extensions and gives guidelines for writing code that access these extensions.

Chapter 16 — Programming with Intel® AVX10. Provides an overview of the Intel® AVX10 instruction set extensions and gives guidelines for writing code that access these extensions.

- Chapter 17 — Programming with Intel® Transactional Synchronization Extensions.** Describes the instruction extensions that support lock elision techniques to improve the performance of multi-threaded software with contended locks.
- Chapter 18 — Control-flow Enforcement Technology.** Provides an overview of the Control-flow Enforcement Technology (CET) and gives guidelines for writing code that access these extensions.
- Chapter 19 — Programming with Intel® Advanced Matrix Extensions.** Provides an overview of the Intel® Advanced Matrix Extensions and gives guidelines for writing code that access these extensions.
- Chapter 20 — Input/Output.** Describes the processor’s I/O mechanism, including I/O port addressing, I/O instructions, and I/O protection mechanisms.
- Chapter 21 — Processor Identification and Feature Determination.** Describes how to determine the CPU type and features available in the processor.
- Appendix A — EFLAGS Cross-Reference.** Summarizes how the IA-32 instructions affect the flags in the EFLAGS register.
- Appendix B — EFLAGS Condition Codes.** Summarizes how conditional jump, move, and ‘byte set on condition code’ instructions use condition code flags (OF, CF, ZF, SF, and PF) in the EFLAGS register.
- Appendix C — Floating-Point Exceptions Summary.** Summarizes exceptions raised by the x87 FPU floating-point and SSE/SSE2/SSE3 floating-point instructions.
- Appendix D — Guidelines for Writing SIMD Floating-Point Exception Handlers.** Gives guidelines for writing exception handlers for exceptions generated by SSE/SSE2/SSE3 floating-point instructions.
- Appendix E — Intel® Memory Protection Extensions.** Provides an overview of the Intel® Memory Protection Extensions, a feature that has been deprecated and will not be available on future processors.

1.3 NOTATIONAL CONVENTIONS

This manual uses specific notation for data-structure formats, for symbolic representation of instructions, and for hexadecimal and binary numbers. This notation is described below.

1.3.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to two raised to the power of the bit position. Intel 64 and IA-32 processors are “little endian” machines; this means the bytes of a word are numbered starting from the least significant byte. See Figure 1-1.

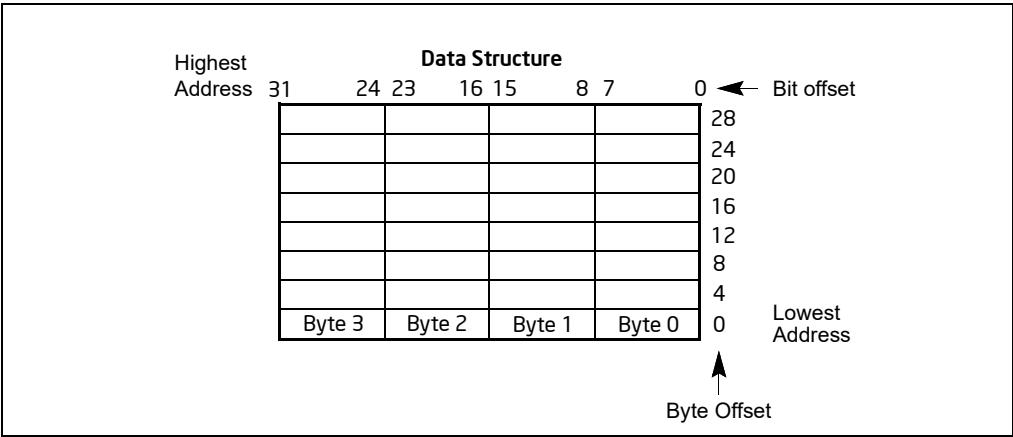


Figure 1-1. Bit and Byte Order

1.3.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable.

Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers that contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

NOTE

Avoid any software dependence upon the state of reserved bits in Intel 64 and IA-32 registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Programs that depend upon reserved values risk incompatibility with future processors.

1.3.2.1 Instruction Operands

When instructions are represented symbolically, a subset of the IA-32 assembly language is used. In this subset, an instruction has the following format:

label: mnemonic argument1, argument2, argument3

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.
- The operands **argument1**, **argument2**, and **argument3** are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

LOADREG: MOV EAX, SUBTOTAL

In this example, LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.3.3 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, 0F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The "B" designation is only used in situations where confusion as to the type of number might arise.

1.3.4 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

Segment-register:Byte-address

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

DS:FF79H

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

CS:EIP

1.3.5 A Syntax for CPUID, CR, and MSR Values

Obtain feature flags, status, and system information by using the CPUID instruction, by checking control register bits, and by reading model-specific registers. See Figure 1-2 for details on the syntax that represents this information.

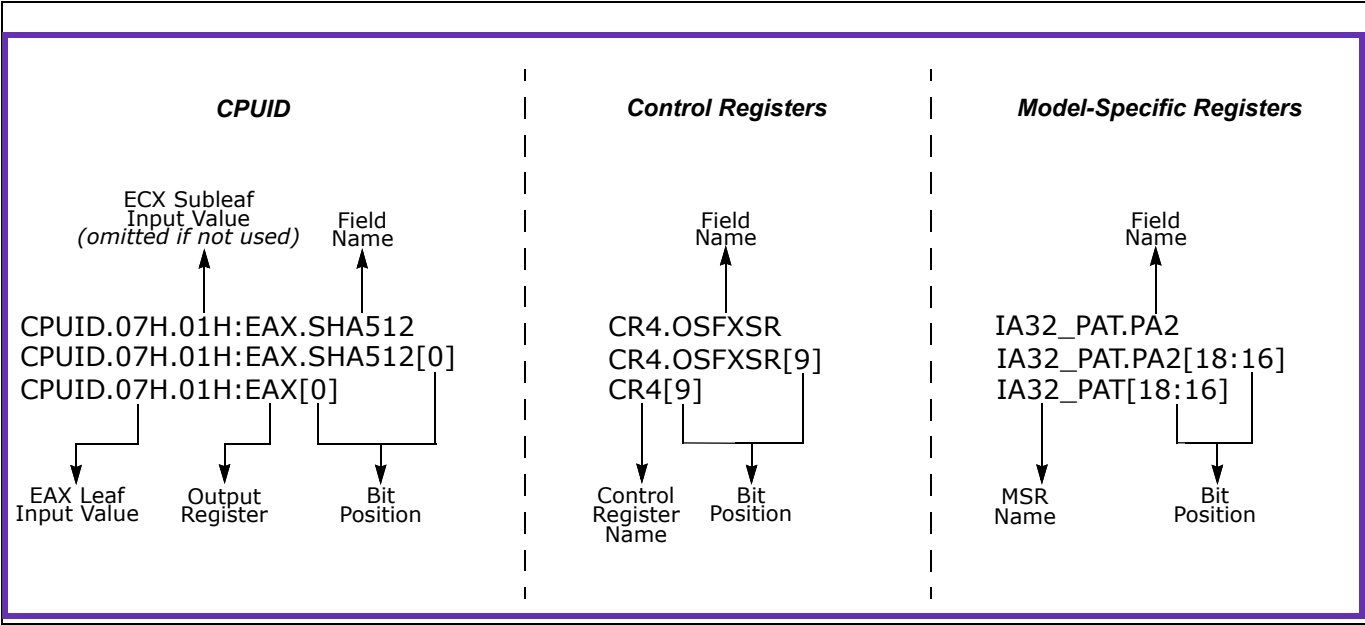


Figure 1-2. Syntax for CPUID, CR, and MSR Data Presentation

1.3.6 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below:

#PF(fault code)

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions that produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception:

#GP(0)

1.4 RELATED LITERATURE

Literature related to Intel 64 and IA-32 processors is listed and viewable on-line at:

<https://software.intel.com/en-us/articles/intel-sdm>

See also:

- The latest security information on Intel® products:
<https://www.intel.com/content/www/us/en/security-center/default.html>
- Software developer resources, guidance, and insights for security advisories:
<https://software.intel.com/security-software-guidance/>
- The data sheet for a particular Intel 64 or IA-32 processor
- The specification update for a particular Intel 64 or IA-32 processor
- Intel® C++ Compiler documentation and online help:
<http://software.intel.com/en-us/articles/intel-compilers/>
- Intel® Fortran Compiler documentation and online help:
<http://software.intel.com/en-us/articles/intel-compilers/>
- Intel® Software Development Tools:
<https://software.intel.com/en-us/intel-sdp-home>
- Intel® 64 and IA-32 Architectures Software Developer's Manual (in one, four or ten volumes):
<https://software.intel.com/en-us/articles/intel-sdm>
- Intel® 64 and IA-32 Architectures Optimization Reference Manual:
<https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm/intel64-and-ia32-architectures-optimization.html>
- Intel® Trusted Execution Technology Measured Launched Environment Developer's Guide:
<http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html>
- Intel® Software Guard Extensions (Intel® SGX) Information:
<https://software.intel.com/en-us/isa-extensions/intel-sgx>
- Developing Multi-threaded Applications: A Platform Consistent Approach:
<https://software.intel.com/sites/default/files/article/147714/51534-developing-multithreaded-applications.pdf>
- Using Spin-Loops on Intel® Pentium® 4 Processor and Intel® Xeon® Processor:
<https://software.intel.com/sites/default/files/22/30/25602>
- Performance Monitoring Unit Sharing Guide:
<http://software.intel.com/file/30388>

Literature related to select features in future Intel processors are available at:

- Intel® Architecture Instruction Set Extensions Programming Reference:
<https://software.intel.com/en-us/isa-extensions>

More relevant links are:

- Intel® Developer Zone:
<https://software.intel.com/en-us>
- Developer centers:
<http://www.intel.com/content/www/us/en/hardware-developers/developer-centers.html>

ABOUT THIS MANUAL

- Processor support general link:
<http://www.intel.com/support/processors/>
- Intel® Hyper-Threading Technology (Intel® HT Technology):
<http://www.intel.com/technology/platform-technology/hyper-threading/index.htm>

2. Updates to Chapter 17, Volume 1

Change bars and violet text show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- In Section 17.3.9.1, “Instruction Based Considerations”, added UDB to the list of instructions that may cause transactional aborts.

17.1 OVERVIEW

This chapter describes the software programming interface to the Intel® Transactional Synchronization Extensions of the Intel 64 architecture.

Multi-threaded applications take advantage of increasing number of cores to achieve high performance. However, writing multi-threaded applications requires programmers to reason about data sharing among multiple threads. Access to shared data typically requires synchronization mechanisms. These mechanisms ensure multiple threads update shared data by serializing operations on the shared data, often through the use of a critical section protected by a lock. Since serialization limits concurrency, programmers try to limit synchronization overheads. They do this either through minimizing the use of synchronization or through the use of fine-grain locks; where multiple locks each protect different shared data. Unfortunately, this process is difficult and error prone; a missed or incorrect synchronization can cause an application to fail. Conservatively adding synchronization and using coarser granularity locks, where a few locks each protect many items of shared data, helps avoid correctness problems but limits performance due to excessive serialization. While programmers must use static information to determine when to serialize, the determination as to whether actually to serialize is best done dynamically.

Intel® Transactional Synchronization Extensions aim to improve the performance of lock-protected critical sections while maintaining the lock-based programming model.

17.2 INTEL® TRANSACTIONAL SYNCHRONIZATION EXTENSIONS

Intel® Transactional Synchronization Extensions (Intel® TSX) allow the processor to determine dynamically whether threads need to serialize through lock-protected critical sections, and to perform serialization only when required. This lets the hardware expose and exploit concurrency hidden in an application due to dynamically unnecessary synchronization through a technique known as lock elision.

With lock elision, the hardware executes the programmer-specified critical sections (also referred to as transactional regions) transactionally. In such an execution, the lock variable is only read within the transactional region; it is not written to (and therefore not acquired) with the expectation that the lock variable remains unchanged after the transactional region, thus exposing concurrency.

If the transactional execution completes successfully, then the hardware ensures that all memory operations performed within the transactional region will appear to have occurred instantaneously when viewed from other logical processors, a process referred to as an **atomic commit**. Any updates performed within the transactional region are made visible to other processors only on an atomic commit.

Since a successful transactional execution ensures an atomic commit, the processor can execute the programmer-specified code section optimistically without synchronization. If synchronization was unnecessary for that specific execution, execution can commit without any cross-thread serialization.

If the transactional execution is unsuccessful, the processor cannot commit the updates atomically. When this happens, the processor will roll back the execution, a process referred to as a **transactional abort**. On a transactional abort, the processor will discard all updates performed in the region, restore architectural state to appear as if the optimistic execution never occurred, and resume execution non-transactionally. Depending on the policy in place, lock elision may be retried or the lock may be explicitly acquired to ensure forward progress.

Intel TSX provides two software interfaces for programmers.

- Hardware Lock Elision (HLE) is a legacy compatible instruction set extension comprising the XACQUIRE and XRELEASE prefixes.
- Restricted Transactional Memory (RTM) is an instruction set interface comprising the XBEGIN and XEND instructions.

Programmers who would like to run Intel TSX-enabled software on legacy hardware would use the HLE interface to implement lock elision. On the other hand, programmers who do not have legacy hardware requirements and who deal with more complex locking primitives would use the RTM software interface of Intel TSX to implement lock elision. In the latter case when using new instructions, the programmer must always provide a non-transactional path (which would have code to eventually acquire the lock being elided) to execute following a transactional abort and must not rely on the transactional execution alone.

Intel TSX provides the XTEST instruction to test whether a logical processor is executing transactionally, and the XABORT instruction to abort a transactional region.

A processor can perform a transactional abort for numerous reasons. A primary cause is due to conflicting accesses between the transactionally executing logical processor and another logical processor. Such conflicting accesses may prevent a successful transactional execution. Memory addresses read from within a transactional region constitute the **read-set** of the transactional region and addresses written to within the transactional region constitute the **write-set** of the transactional region. Intel TSX maintains the read- and write-sets at the granularity of a cache line.

A conflicting data access occurs if another logical processor either reads a location that is part of the transactional region's write-set or writes a location that is a part of either the read- or write-set of the transactional region. We refer to this as a **data conflict**. Since Intel TSX detects data conflicts at the granularity of a cache line, unrelated data locations placed in the same cache line will be detected as conflicts. Transactional aborts may also occur due to limited transactional resources. For example, the amount of data accessed in the region may exceed an implementation-specific capacity. Additionally, some instructions and system events may cause transactional aborts.

Additionally, Intel TSX provides the XSUSLDRK and XRESLDRK instructions to suspend and resume load address tracking.

17.2.1 HLE Software Interface

HLE provides two instruction prefix hints: XACQUIRE and XRELEASE.

The programmer uses the XACQUIRE prefix in front of the instruction that is used to acquire the lock that is protecting the critical section. The processor treats the indication as a hint to elide the write associated with the lock acquire operation. Even though the lock acquire has an associated write operation to the lock, the processor does not add the address of the lock to the transactional region's write-set nor does it issue any write requests to the lock. Instead, the address of the lock is added to the read-set. The logical processor enters transactional execution. If the lock was available before the XACQUIRE prefixed instruction, all other processors will continue to see it as available afterwards. Since the transactionally executing logical processor neither added the address of the lock to its write-set nor performed externally visible write operations to it, other logical processors can read the lock without causing a data conflict. This allows other logical processors to also enter and concurrently execute the critical section protected by the lock. The processor automatically detects any data conflicts that occur during the transactional execution and will perform a transactional abort if necessary.

Even though the eliding processor did not perform any external write operations to the lock, the hardware ensures program order of operations on the lock. If the eliding processor itself reads the value of the lock in the critical section, it will appear as if the processor had acquired the lock, i.e., the read will return the non-elided value. This behavior makes an HLE execution functionally equivalent to an execution without the HLE prefixes.

The programmer uses the XRELEASE prefix in front of the instruction that is used to release the lock protecting the critical section. This involves a write to the lock. If the instruction is restoring the value of the lock to the value it had prior to the XACQUIRE prefixed lock acquire operation on the same lock, then the processor elides the external write request associated with the release of the lock and does not add the address of the lock to the write-set. The processor then attempts to commit the transactional execution.

With HLE, if multiple threads execute critical sections protected by the same lock but they do not perform any conflicting operations on each other's data, then the threads can execute concurrently and without serialization. Even though the software uses lock acquisition operations on a common lock, the hardware recognizes this, elides the lock, and executes the critical sections on the two threads without requiring any communication through the lock — if such communication was dynamically unnecessary.

If the processor is unable to execute the region transactionally, it will execute the region non-transactionally and without elision. HLE enabled software has the same forward progress guarantees as the underlying non-HLE lock-based execution. For successful HLE execution, the lock and the critical section code must follow certain guidelines

(discussed in Section 17.3.3 and Section 17.3.9). These guidelines only affect performance; not following these guidelines will not cause a functional failure.

Hardware without HLE support will ignore the XACQUIRE and XRELEASE prefix hints and will not perform any elision since these prefixes correspond to the REPNE/REPE IA-32 prefixes which are ignored on the instructions where XACQUIRE and XRELEASE are valid. Importantly, HLE is compatible with the existing lock-based programming model. Improper use of hints will not cause functional bugs though it may expose latent bugs already in the code.

17.2.2 RTM Software Interface

RTM provides three instructions: XBEGIN, XEND, and XABORT.

Software uses the XBEGIN instruction to specify the start of the transactional region and the XEND instruction to specify the end of the transactional region. The XBEGIN instruction takes an operand that provides a relative offset to the **fallback instruction address** if the transactional region could not be successfully executed transactionally. Software using these instructions to implement lock elision must test the lock within the transactional region, and only if free should try to commit. Further, the software may also define a policy to retry if the lock is not free.

A processor may abort transactional execution for many reasons. The hardware automatically detects transactional abort conditions and restarts execution from the fallback instruction address with the architectural state corresponding to that at the start of the XBEGIN instruction and the EAX register updated to describe the abort status.

The XABORT instruction allows programmers to abort the execution of a transactional region explicitly. The XABORT instruction takes an 8 bit immediate argument that is loaded into the EAX register and will thus be available to software following a transactional abort.

Hardware provides no guarantees as to whether a transactional execution will ever successfully commit. Programmers must always provide an alternative code sequence in the fallback path to guarantee forward progress. When using the instructions for lock elision, this may be as simple as acquiring a lock and executing the specified code region non-transactionally. Further, a transactional region that always aborts on a given implementation may complete transactionally on a future implementation. Therefore, programmers must ensure the code paths for the transactional region and the alternative code sequence are functionally tested.

If the RTM software interface is used for anything other than lock elision, the programmer must similarly ensure that the fallback path is inter-operable with the transactionally executing path.

17.3 INTEL® TSX APPLICATION PROGRAMMING MODEL

17.3.1 Detection of Transactional Synchronization Support

17.3.1.1 Detection of HLE Support

A processor supports HLE execution if CPUID.07H.00H:EBX.HLE[4] = 1. However, an application can use the HLE prefixes (XACQUIRE and XRELEASE) without checking whether the processor supports HLE. Processors without HLE support ignore these prefixes and will execute the code without entering transactional execution.

17.3.1.2 Detection of RTM Support

A processor supports RTM execution if CPUID.07H.00H:EBX.RTM[11] = 1. An application must check if the processor supports RTM before it uses the RTM instructions (XBEGIN, XEND, and XABORT). These instructions will generate a #UD exception when used on a processor that does not support RTM.

17.3.1.3 Detection of XTEST Instruction

A processor supports the XTEST instruction if it supports either HLE or RTM. An application must check either of these feature flags before using the XTEST instruction. This instruction will generate a #UD exception when used on a processor that does not support either HLE or RTM.

17.3.1.4 Detection of Intel® TSX Suspend Load Address Tracking

A processor supports Intel TSX suspend/resume of load address tracking if CPUID.07H.00H:EDX.TSXLDRK[16] = 1. An application must check if the processor supports Intel TSX suspend/resume of load address tracking before it uses the Intel TSX suspend/resume load address tracking instructions (XSUSLDRK and XRESLDRK). These instructions will generate a #UD exception when used on a processor that does not support Intel TSX suspend/resume load address tracking.

17.3.2 Querying Transactional Execution Status

The XTEST instruction can be used to determine the transactional status of a transactional region specified by HLE or RTM. Note, while the HLE prefixes are ignored on processors that do not support HLE, the XTEST instruction will generate a #UD exception when used on processors that do not support either HLE or RTM.

17.3.3 Requirements for HLE Locks

For HLE execution to successfully commit transactionally, the lock must satisfy certain properties and access to the lock must follow certain guidelines.

- An XRELEASE prefixed instruction must restore the value of the elided lock to the value it had before the lock acquisition. This allows hardware to safely elide locks by not adding them to the write-set. The data size and data address of the lock release (XRELEASE prefixed) instruction must match that of the lock acquire (XACQUIRE prefixed) and the lock must not cross a cache line boundary.
- Software should not write to the elided lock inside a transactional HLE region with any instruction other than an XRELEASE prefixed instruction, otherwise it may cause a transactional abort. In addition, recursive locks (where a thread acquires the same lock multiple times without first releasing the lock) may also cause a transactional abort. Note that software can observe the result of the elided lock acquire inside the critical section. Such a read operation will return the value of the write to the lock.

The processor automatically detects violations to these guidelines, and safely transitions to a non-transactional execution without elision. Since Intel TSX detects conflicts at the granularity of a cache line, writes to data collocated on the same cache line as the elided lock may be detected as data conflicts by other logical processors eliding the same lock.

17.3.4 Transactional Nesting

Both HLE- and RTM-based transactional executions support nested transactional regions. However, a transactional abort restores state to the operation that started transactional execution: either the outermost XACQUIRE prefixed HLE eligible instruction or the outermost XBEGIN instruction. The processor treats all nested transactional regions as one monolithic transactional region.

17.3.4.1 HLE Nesting and Elision

Programmers can nest HLE regions up to an implementation specific depth of MAX_HLE_NEST_COUNT. Each logical processor tracks the nesting count internally but this count is not available to software. An XACQUIRE prefixed HLE-eligible instruction increments the nesting count, and an XRELEASE prefixed HLE-eligible instruction decrements it. The logical processor enters transactional execution when the nesting count goes from zero to one. The logical processor attempts to commit only when the nesting count becomes zero. A transactional abort may occur if the nesting count exceeds MAX_HLE_NEST_COUNT.

In addition to supporting nested HLE regions, the processor can also elide multiple nested locks. The processor tracks a lock for elision beginning with the XACQUIRE prefixed HLE eligible instruction for that lock and ending with

the XRELEASE prefixed HLE eligible instruction for that same lock. The processor can, at any one time, track up to a MAX_HLE_ELIDED_LOCKS number of locks. For example, if the implementation supports a MAX_HLE_ELIDED_LOCKS value of two and if the programmer nests three HLE identified critical sections (by performing XACQUIRE prefixed HLE eligible instructions on three distinct locks without performing an intervening XRELEASE prefixed HLE eligible instruction on any one of the locks), then the first two locks will be elided, but the third won't be elided (but will be added to the transaction's write-set). However, the execution will still continue transactionally. Once an XRELEASE for one of the two elided locks is encountered, a subsequent lock acquired through the XACQUIRE prefixed HLE eligible instruction will be elided.

The processor attempts to commit the HLE execution when all elided XACQUIRE and XRELEASE pairs have been matched, the nesting count goes to zero, and the locks have satisfied the requirements described earlier. If execution cannot commit atomically, then execution transitions to a non-transactional execution without elision as if the first instruction did not have an XACQUIRE prefix.

17.3.4.2 RTM Nesting

Programmers can nest RTM-based transactional regions up to an implementation specific MAX_RT-M_NEST_COUNT. The logical processor tracks the nesting count internally but this count is not available to software. An XBEGIN instruction increments the nesting count, and an XEND instruction decrements it. The logical processor attempts to commit only if the nesting count becomes zero. A transactional abort occurs if the nesting count exceeds MAX_RTM_NEST_COUNT.

17.3.4.3 Nesting HLE and RTM

HLE and RTM provide two alternative software interfaces to a common transactional execution capability. The behavior when HLE and RTM are nested together—HLE inside RTM or RTM inside HLE—is implementation specific. However, in all cases, the implementation will maintain HLE and RTM semantics. An implementation may choose to ignore HLE hints when used inside RTM regions, and may cause a transactional abort when RTM instructions are used inside HLE regions. In the latter case, the transition from transactional to non-transactional execution occurs seamlessly since the processor will re-execute the HLE region without actually doing elision, and then execute the RTM instructions.

17.3.5 RTM Abort Status Definition

RTM uses the EAX register to communicate abort status to software. Following an RTM abort the EAX register has the following definition.

Table 17-1. RTM Abort Status Definition

EAX Register Bit Position	Meaning
0	Set if abort caused by XABORT instruction.
1	If set, the transactional execution may succeed on a retry. This bit is always clear if bit 0 is set.
2	Set if another logical processor conflicted with a memory address that was part of the transactional execution that aborted.
3	Set if an internal buffer to track transactional state overflowed.
4	Set if a debug exception (#DB) or breakpoint exception (#BP) was hit.
5	Set if an abort occurred during execution of a nested transactional execution.
23:6	Reserved.
31:24	XABORT argument (only valid if bit 0 set, otherwise reserved).

The EAX abort status for RTM only provides causes for aborts. It does not by itself encode whether an abort or commit occurred for the RTM region. The value of EAX can be 0 following an RTM abort. For example, a CPUID

instruction when used inside an RTM region causes a transactional abort and may not satisfy the requirements for setting any of the EAX bits. This may result in an EAX value of 0.

17.3.6 RTM Memory Ordering

A successful RTM commit causes all memory operations in the RTM region to appear to execute atomically. A successfully committed RTM region consisting of an XBEGIN followed by an XEND, even with no memory operations in the RTM region, has the same ordering semantics as a LOCK prefixed instruction.

The XBEGIN instruction does not have fencing semantics. However, if an RTM execution aborts, all memory updates from within the RTM region are discarded and never made visible to any other logical processor.

17.3.7 RTM-Enabled Debugger Support

Any debug exception (#DB) or breakpoint exception (#BP) inside an RTM region causes a transactional abort and, by default, redirects control flow to the fallback instruction address with architectural state recovered and bit 4 in EAX set. However, to allow software debuggers to intercept execution on debug or breakpoint exceptions, the RTM architecture provides additional capability called **advanced debugging of RTM transactional regions**.

Advanced debugging of RTM transactional regions is enabled if bit 11 of DR7 and bit 15 of the IA32_DEBUGCTL MSR are both 1. In this case, any RTM transactional abort due to a #DB or #BP causes execution to roll back to just before the XBEGIN instruction (EAX is restored to the value it had before XBEGIN) and then delivers a #DB. (A #DB is delivered even if the transactional abort was caused by a #BP.) DR6[16] is cleared to indicate that the exception resulted from a debug or breakpoint exception inside an RTM region. See also Section 19.3.3, “Debug Exceptions, Breakpoint Exceptions, and Restricted Transactional Memory (RTM),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

17.3.8 Intel® TSX Suspend/Resume Load Address Tracking Support

Programmers can use Intel TSX suspend/resume of load address tracking to choose which memory accesses do not need to be tracked in the Intel TSX read set. A programmer who uses the suspend/resume load address tracking feature must ensure that there are no atomicity requirements related to the addresses they choose to exclude from the read set as hardware will not detect read-write conflicts for those addresses.

To prevent load addresses from being entered into the read set, the programmer should use the XSUSLDTRK and XRESLDTRK instructions. The XSUSLDTRK instruction suspends loads tracking and thus specifies the start of a **suspend region**; addresses of subsequent loads will not be added to the transaction read set. The XRESLDTRK instruction resumes load tracking and thus specifies the end of a suspend region; addresses of subsequent loads will be added to the transaction read set.

The execution of a suspend region is similar to transaction execution with the following exceptions:

- The addresses of loads in a suspend region are not tracked for read-write conflicts if the addresses are accessed inside the suspend region only (they are not added to the transaction read set). The addresses are still tracked if they are accessed outside of the suspend region inside the transaction.
- Transaction start/end inside a suspend region is not supported; any execution of XBEGIN or XEND inside a suspend region will cause the transaction to abort.
- There is no support for nesting of suspend regions; execution of XSUSLDTRK in a suspend region will cause a transaction to abort.

17.3.9 Programming Considerations

Typical programmer-identified regions are expected to execute transactionally and to commit successfully. However, Intel TSX does not provide any such guarantee. A transactional execution may abort for many reasons. To take full advantage of the transactional capabilities, programmers should follow certain guidelines to increase the probability of their transactional execution committing successfully.

This section discusses various events that may cause transactional aborts. The architecture ensures that updates performed within a transactional region that subsequently aborts execution will never become visible. Only a

committed transactional execution updates architectural state. Transactional aborts never cause functional failures and only affect performance.

17.3.9.1 Instruction Based Considerations

Programmers can use any instruction safely inside a transactional region. Further, programmers can use the Intel TSX instructions and prefixes at any privilege level. However, some instructions will always abort the transactional execution and cause execution to seamlessly and safely transition to a non-transactional path.

Intel TSX allows for most common instructions to be used inside transactional regions without causing aborts. The following operations inside a transactional region do not typically cause an abort.

- Operations on the instruction pointer register, general purpose registers (GPRs) and the status flags (CF, OF, SF, PF, AF, and ZF).
- Operations on XMM and YMM registers and the MXCSR register

However, programmers must be careful when intermixing SSE and AVX operations inside a transactional region. Intermixing SSE instructions accessing XMM registers and AVX instructions accessing YMM registers may cause transactional regions to abort.

CLD and STD instructions when used inside transactional regions may cause aborts if they change the value of the DF flag. However, if DF is 1, the STD instruction will not cause an abort. Similarly, if DF is 0, the CLD instruction will not cause an abort.

Instructions not enumerated here as causing abort when used inside a transactional region will typically not cause the execution to abort (examples include but are not limited to MFENCE, LFENCE, SFENCE, RDTSC, RDTSCP, etc.).

The following instructions will abort transactional execution on any implementation:

- XABORT
- CPUID
- PAUSE
- ENCLS
- ENCLU

In addition, in some implementations, the following instructions may always cause transactional aborts. These instructions are not expected to be commonly used inside typical transactional regions. However, programmers must not rely on these instructions to force a transactional abort, since whether they cause transactional aborts is implementation dependent.

- Operations on X87 and MMX architecture state. This includes all MMX and X87 instructions, including the FXRSTOR and FXSAVE instructions.
- Update to non-status portion of EFLAGS or to UIF: CLI, CLUI, STI, STUI, POPFD, POPFQ, CLAC, and STAC.
- Instructions that update segment registers, debug registers and/or control registers: MOV to DS/ES/FS/GS/SS, POP DS/ES/FS/GS/SS, LDS, LES, LFS, LGS, LSS, SWAPGS, WRFSBASE, WRGSBASE, LGDT, SGDT, LIDT, SIDT, LLDT, SLDT, LTR, STR, Far CALL, Far JMP, Far RET, IRET, MOV to DRx, MOV to CR0/CR2/CR3/CR4/CR8, CLTS, and LMSW.
- Ring transitions: SYSENTER, SYSCALL, SYSEXIT, and SYSRET.
- TLB and Cacheability control: CLFLUSH, CLFLUSHOPT, CLWB, INVD, WBINVD, INVLPG, INVPCID, and memory instructions with a non-temporal hint (V/MOVNTDQA, V/MOVNTDQ, V/MOVNTI, V/MOVNTPD, V/MOVNTPS, V/MOVNTQ, V/MASKMOVQ, and V/MASKMOVDQU).
- Extended state management: XRSTOR, XRSTORS, XSAVE, XSAVEC, XSAVEOPT, XSAVES, and XSETBV.
- Interrupts: INT *n*, INTO, INT3, and INT1.
- I/O: IN, INS, REP INS, OUT, OUTS, REP OUTS and their variants.
- VMX: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON, INVEPT, INVVPID, and VMFUNC.
- SMX: GETSEC.
- UD0, UD1, UD2, **UDB**, RSM, RDMSR, WRMSR, WRPKRU, HLT, MONITOR, MWAIT, and VZEROUPPER.

17.3.9.2 Runtime Considerations

In addition to the instruction-based considerations, runtime events may cause transactional execution to abort. These may be due to data access patterns or micro-architectural implementation causes. Keep in mind that the following list is not a comprehensive discussion of all abort causes.

Any fault or trap in a transactional region that must be exposed to software will be suppressed. Transactional execution will abort and execution will transition to a non-transactional execution, as if the fault or trap had never occurred. If any exception is not masked, that will result in a transactional abort and it will be as if the exception had never occurred.

When executed in VMX non-root operation, certain instructions may result in a VM exit. When such instructions are executed inside a transactional region, then instead of causing a VM exit, they will cause a transactional abort and the execution will appear as if instruction that would have caused a VM exit never executed.

Synchronous exception events (#DE, #OF, #NP, #SS, #GP, #BR, #UD, #AC, #XM, #PF, #NM, #TS, #MF, #DB, #BP/INT3) that occur during transactional execution may cause an execution not to commit transactionally, and require a non-transactional execution. These events are suppressed as if they had never occurred. With HLE, since the non-transactional code path is identical to the transactional code path, these events will typically re-appear when the instruction that caused the exception is re-executed non-transactionally, causing the associated synchronous events to be delivered appropriately in the non-transactional execution. The same behavior also applies to synchronous events (EPT violations, EPT misconfigurations, and accesses to the APIC-access page) that occur in VMX non-root operation.

Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution. The asynchronous events will be pending and handled after the transactional abort is processed. The same behavior also applies to asynchronous events (VMX-preemption timer expiry, virtual-interrupt delivery, and interrupt-window exiting) that occur in VMX non-root operation.

Transactional execution only supports write-back cacheable memory type operations. A transactional region may always abort if it includes operations on any other memory type. This includes instruction fetches to UC memory type.

Memory accesses within a transactional region may require the processor to set the Accessed and Dirty flags of the referenced page table entry. The behavior of how the processor handles this is implementation specific. Some implementations may allow the updates to these flags to become externally visible even if the transactional region subsequently aborts. Some Intel TSX implementations may choose to abort the transactional execution if these flags need to be updated. Further, a processor's page-table walk may generate accesses to its own transactionally written but uncommitted state. Some Intel TSX implementations may choose to abort the execution of a transactional region in such situations. Regardless, the architecture ensures that, if the transactional region aborts, then the transactionally written state will not be made architecturally visible through the behavior of structures such as TLBs.

Executing self-modifying code transactionally may also cause transactional aborts. Programmers must continue to follow the Intel recommended guidelines for writing self-modifying and cross-modifying code even when employing Intel TSX.

While an Intel TSX implementation will typically provide sufficient resources for executing common transactional regions, implementation constraints and excessive sizes for transactional regions may cause a transactional execution to abort and transition to a non-transactional execution. The architecture provides no guarantee of the amount of resources available to do transactional execution and does not guarantee that a transactional execution will ever succeed.

Conflicting requests to a cache line accessed within a transactional region may prevent the transactional region from executing successfully. For example, if logical processor P0 reads line A in a transactional region and another logical processor P1 writes A (either inside or outside a transactional region) then logical processor P0 may abort if logical processor P1's write interferes with processor P0's ability to execute transactionally. Similarly, if P0 writes line A in a transactional region and P1 reads or writes A (either inside or outside a transactional region), then P0 may abort if P1's access to A interferes with P0's ability to execute transactionally. In addition, other coherence traffic may at times appear as conflicting requests and may cause aborts. While these false conflicts may happen, they are expected to be uncommon. The conflict resolution policy to determine whether P0 or P1 aborts in the above scenarios is implementation specific.

3. Updates to Chapter 21, Volume 1

Change bars and violet text show changes to Chapter 21 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.

Changes to this chapter:

- Added CPUID enumeration information, which was previously contained under the CPUID instruction in Chapter 3 of Volume 2A in the *Intel® 64 and IA-32 Architectures Software Developer's Manual*.

CHAPTER 21

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

When writing software intended to run on Intel processors, it is necessary to identify the type of processor present in a system and the processor features that are available to an application.

The CPUID instruction, known as CPU Identification, was introduced with the Intel® Pentium processor to query the processor's information name space for its identity and supported features. Logically, the CPUID name space comprises a series of nodes indexed by leaf (using the input value of EAX) and in some cases further indexed by sub-leaf (using the input value of ECX). The value of a queried node is returned in EAX, EBX, ECX, and EDX. Note that not all leaves have sub-leaf indexing and the input ECX value will be ignored in those cases. The full description of CPUID can be found in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

■ All references to "MAX_LEAF" throughout this chapter are used as an abbreviation of "CPUID.00H:EAX.MAX_LEAF".

21.1 IMPORTANT CONSIDERATIONS WHEN USING THE CPUID INSTRUCTION

This section outlines additional factors to consider when using the CPUID instruction.

21.1.1 Guidelines for Using the CPUID Instruction

Use the CPUID instruction for processor identification in the Pentium M processor family, Pentium 4 processor family, Intel Xeon processor family, P6 family, Pentium processor, and later Intel486 processors. This instruction returns the family, model, and (for some processors) a brand string for the processor that executes the instruction. It also indicates the features that are present in the processor and gives information about the processor's caches and TLB.

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. The CPUID instruction will cause the invalid opcode exception (#UD) if executed on a processor that does not support it.

To obtain processor identification information, a source operand value is placed in the EAX register to select the type of information to be returned. When the CPUID instruction is executed, selected information is returned in the EAX, EBX, ECX, and EDX registers.

The following guidelines are among the most important and should always be followed when using the CPUID instruction to determine available features:

- Always begin by testing for the "GenuineIntel," message in the EBX, EDX, and ECX registers when the CPUID instruction is executed with EAX equal to 0. If the processor is not genuine Intel, the feature identification flags may have different meanings than are described in Intel documentation.
- Test feature identification flags individually and do not make assumptions about undefined bits.

21.1.2 Identification of Earlier Processors

The CPUID instruction is not available in earlier Intel processors up through the earlier Intel 486 processors. For these processors, several other architectural features can be exploited to identify the processor.

The settings of bits 12 and 13 (IOPL), 14 (NT), and 15 (reserved) in the EFLAGS register are different for Intel's 32-bit processors than for the Intel 8086 and Intel 286 processors. By examining the settings of these bits (with the PUSHF/PUSHFD and POPF/POPFD instructions), an application program can determine whether the processor is an 8086, Intel 286, or one of the Intel 32-bit processors:

- 8086 processor — Bits 12 through 15 of the EFLAGS register are always set.
- Intel 286 processor — Bits 12 through 15 are always clear in real-address mode.

- 32-bit processors — In real-address mode, bit 15 is always clear and bits 12 through 14 have the last value loaded into them. In protected mode, bit 15 is always clear, bit 14 has the last value loaded into it, and the IOPL bits depend on the current privilege level (CPL). The IOPL field can be changed only if the CPL is 0.

Other EFLAGS register bits that can be used to differentiate between the 32-bit processors:

- Bit 18 (AC) — Implemented only on the Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors. The inability to set or clear this bit distinguishes an Intel386 processor from the later IA-32 processors.
- Bit 21 (ID) — Determines if the processor is able to execute the CPUID instruction. The ability to set and clear this bit indicates that it is a Pentium 4, Intel Xeon, P6 family, Pentium, or later-version Intel486 processor.

To determine whether an x87 FPU or Numeric Processor Extension (NPX) is present in a system, applications can write to the x87 FPU status and control registers using the FNINIT instruction and then verify that the correct values are read back using the FNSTENV instruction.

After determining that an x87 FPU or NPX is present, its type can then be determined. In most cases, the processor type will determine the type of FPU or NPX; however, an Intel386 processor is compatible with either an Intel 287 or Intel 387 math coprocessor.

The method the coprocessor uses to represent ∞ (after the execution of the FINIT, FNINIT, or RESET instruction) indicates which coprocessor is present. The Intel 287 math coprocessor uses the same bit representation for $+\infty$ and $-\infty$; whereas, the Intel 387 math coprocessor uses different representations for $+\infty$ and $-\infty$.

21.1.3 CPUID Basic and Extended Range

The CPUID basic range starts at CPUID.00H and ends at the maximum leaf enumerated in CPUID.00H:EAX.MAX_LEAF[31:0].

The legacy set of CPUID leaves are defined as leaves 00H, 01H, and 02H, which represent the architecture up to and including Pentium II. Processors provided legacy compatibility by limiting the exposed number of leaves to just these legacy leaves by setting IA32_MISC_ENABLE[22] (Limit CPUID Maxval). This is no longer supported on processors that report CPUID.07H.01H:EBX.CPUIDMAXVAL_LIM_RMV[3] as 1; for such processors, IA32_MISC_ENABLE[22] cannot be set to 1 to limit the value returned by CPUID.00H:EAX.MAX_LEAF.

The extended CPUID range starts at leaf 80000000H and ends at the maximum leaf enumerated in CPUID.80000000H:EAX.MAX_EXTENDED_LEAF[31:0].

Older processors before the Pentium 4 do not support the extended CPUID range and treat bit 31 of CPUID's input EAX value as zero.

If a value entered for CPUID.EAX is higher than the maximum input value for basic or extended function for that processor then the data for the highest basic information leaf is returned. Software should not rely on the values returned by the processor outside of the above ranges.

The range CPUID.40000000H to CPUID.4FFFFFFFH do not return feature information for the processor. These are allocated for emulation by software.

21.1.4 CPUID Domains

The fields of each CPUID node are classified into one of several CPUID domains. The fields may be classified separately within a specific node or in aggregate for all the nodes in a leaf or sub-leaf. On a properly configured platform, all logical processors within a CPUID domain return a consistent output value for fields belonging to that domain. As an example, the initial X2APIC ID value returned in CPUID.1FH.00H:EDX[31:0] is classified as being in the Logical Processor Domain because the value is unique for each logical processor in the platform. Whereas, the CLFLUSH Line Size returned in CPUID.00H:EBX[15:8] is classified as Platform Domain because it must be consistent for all logical processor within the entire platform.

- **Platform Domain**—A properly configured platform would provide consistent values for these CPUID fields for each logical processor in the platform.
- **Package Domain**—A properly configured platform provides consistent values for these CPUID fields for each logical processor within the same processor package. These values however can be different when comparing the values of logical processors on different packages.

- **Logical Processor Domain**—A properly configured platform can provide different values for these CPUID fields for each logical processor in the platform. The values contained within these may have their own scope as per a specific shared resource (i.e., cache, hybrid, etc.); in that case, each logical processor may need to be queried to obtain the full platform view of given features.

21.1.5 CPUID Runtime Mutable Fields

A CPUID field is said to be mutable if it can change during runtime. Such fields are affected by supervisor-mode operations that can affect processor mode, status bits, or privileged registers. Mutable fields that are expected to change dynamically as part of normal operation are shown in the table Table 21-1. Mutable fields that should remain consistent are shown in the table Table 21-2. Note all of the listed controls may not be available on all Intel processors.

Table 21-1. Runtime Mutable CPUID Fields Expected to Change During Normal Operation

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
01H	Ignored	ECX[27]	OSXSAVE	If 1, the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCR0 and to support processor extended state management using XSAVE/XRSTOR.
07H	00H	ECX[4]	OSPKE	If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions).
0DH	00H	EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURE	The size of the XSAVE/XRSTOR area required for the state bits enabled in XCR0.
0DH	01H	EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURE	The size of the XSAVES/XRSTORS area required for the state bits enabled in XCR0 and IA32_XSS.
19H	00H	EBX[0]	AESKLE	If 1, if the AES Key Locker instructions have been activated by system firmware and the OS has set CR4.KL[bit 19] = 1.
80000001H	Ignored	EDX[20]	SYSCALL_SYSRET_64	Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.

Table 21-2. Runtime Mutable CPUID Fields That Should Remain Consistent

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
00H	Ignored	EAX[31:0]	MAX_LEAF	Support for legacy software by limiting CPUID number of leaves reporting to a maximum of 2. This is set by using IA32_MISC_ENABLE[22] Limit CPUID Maxval.
01H	Ignored	ECX[3]	MONITOR	This feature flag reflects the setting in IA32_MISC_ENABLE[18] Enable Monitor FSM.
01H	Ignored	ECX[7]	EIST	This feature flag reflects the setting in IA32_MISC_ENABLE[16] Enhanced Intel SpeedStep Technology Enable.
01H	Ignored	EDX[9]	APIC	This feature flag reflects IA32_APIC_BASE[11], APIC Global Enable.
05H	Ignored	ECX[0]	MONITOR_MWAIT_EXTENSIONS	This field not available when CPUID.01H:ECX.MONITOR[3] = 0.

Table 21-2. Runtime Mutable CPUID Fields That Should Remain Consistent (Contd.)

Leaf	Sub-Leaf	Register	Field Name	Description and Mutability Control
05H	Ignored	ECX[1]	INTERRUPT_AS_BREAK_EVENT	This field not available when CPUID.01H:ECX.MONITOR[3] = 0.
05H	Ignored	EDX[15:0]	LARGEST_MONITOR_LINE_SIZE	This field not available when CPUID.01H:ECX.MONITOR[3] = 0.
05H	Ignored	EAX[15:0]	SMALLEST_MONITOR_LINE_SIZE	This field not available when CPUID.01H:ECX.MONITOR[3] = 0.

21.1.6 CPUID Reserved Fields

Software must ignore and not rely upon the values returned by reserved fields of a CPUID leaf or sub-leaf because they may have meaning on future processors. Once a previously-reserved field becomes defined, this specification will be updated to reflect that.

21.1.7 CPUID Instruction for Serialization

Although the CPUID instruction provides serialization, it is not the preferred method on newer processors that support the SERIALIZE instruction, which is enumerated via CPUID.07H.00H:EDX[14]=1. If backward compatibility is required with older processors, use leaf 00H [CPUID.00H] for serialization because it has the lowest latency when executed. See “Serializing Instructions” in Chapter 10 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A for more details.

21.1.8 IA32_BIOS_SIGN_ID Returns Microcode Update Signature

For processors that support the microcode update facility, the IA32_BIOS_SIGN_ID MSR is loaded with the update signature whenever CPUID executes. The signature is returned in the upper DWORD. For details, see Chapter 10 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

21.2 METHODS FOR RETURNING BRANDING INFORMATION USING CPUID

Use the following techniques to access branding information:

- 1. Processor brand string method.
- 2. Processor brand index; this method uses a software supplied brand string table.

These two methods are discussed in the following sections. For methods that are available in early processors, see Section 21.1.2, “Identification of Earlier Processors,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

21.2.1 The Processor Brand String Method

Figure 21-1 describes the algorithm used for detection of the brand string. Processor brand identification software should execute this algorithm on all Intel 64 and IA-32 processors.

This method (introduced with Pentium 4 processors) returns an ASCII brand identification string and the Processor Base frequency of the processor to the EAX, EBX, ECX, and EDX registers.

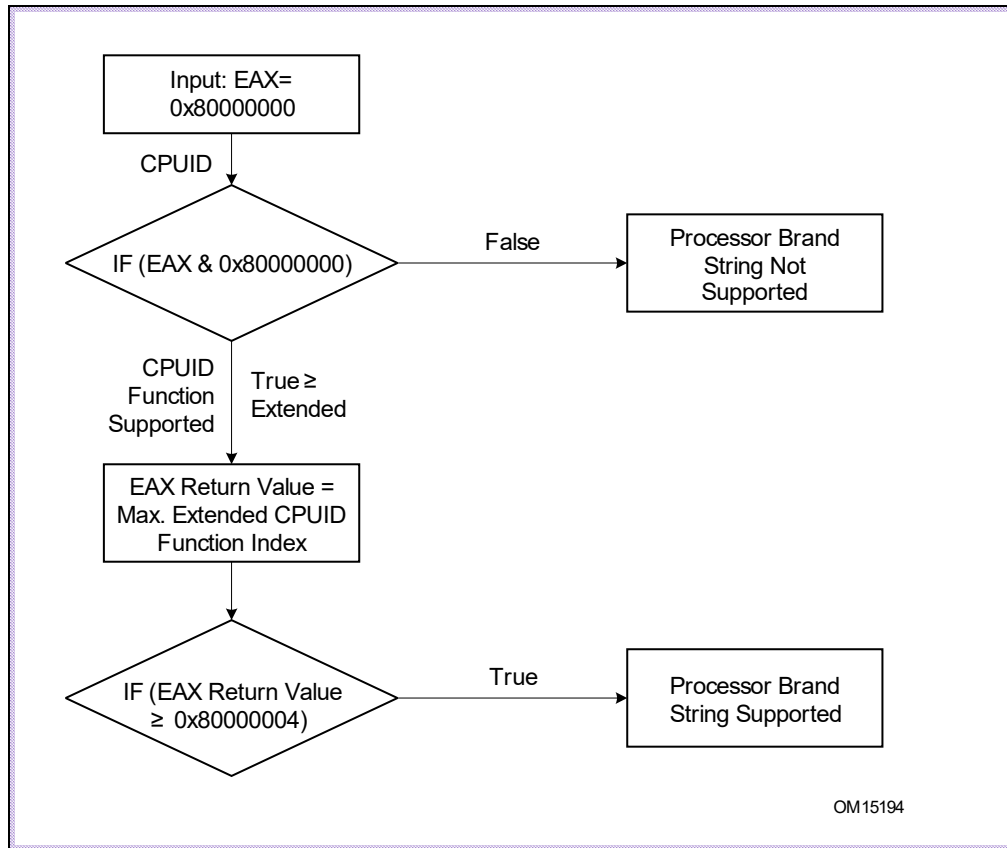


Figure 21-1. Determination of Support for the Processor Brand String

21.2.2 The Processor Brand Index Method

The brand index method (introduced with Pentium® III Xeon® processors) provides an entry point into a brand identification table that is maintained in memory by software. In this table, each brand index is associated with an ASCII brand identification string that identifies the official Intel family and model number of a processor.

When CUID executes with EAX set to 1, the processor returns a brand index to the low byte in EBX. Software can then use this index to locate the brand identification string for the processor in the brand identification table. The first entry (brand index 0) in this table is reserved, allowing for backward compatibility with processors that do not support the brand identification feature. Starting with processor signature family ID = 0FH, model = 03H, brand index method is no longer supported. Use brand string method instead.

Table 21-3 shows brand indices that have identification strings associated with them.

Table 21-3. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

Brand Index	Brand String
00H	This processor does not support the brand identification feature
01H	Intel® Celeron® processor ¹
02H	Intel® Pentium® III processor ¹
03H	Intel® Pentium® III Xeon® processor; If processor signature = 000006B1h, then Intel® Celeron® processor
04H	Intel® Pentium® III processor
06H	Mobile Intel® Pentium® III processor-M
07H	Mobile Intel® Celeron® processor ¹

Table 21-3. Mapping of Brand Indices; and Intel 64 and IA-32 Processor Brand Strings

08H	Intel® Pentium® 4 processor
09H	Intel® Pentium® 4 processor
0AH	Intel® Celeron® processor ¹
0BH	Intel® Xeon® processor; If processor signature = 00000F13h, then Intel® Xeon® processor MP
0CH	Intel® Xeon® processor MP
0EH	Mobile Intel® Pentium® 4 processor-M; If processor signature = 00000F13h, then Intel® Xeon® processor
0FH	Mobile Intel® Celeron® processor ¹
11H	Mobile Genuine Intel® processor
12H	Intel® Celeron® M processor
13H	Mobile Intel® Celeron® processor ¹
14H	Intel® Celeron® processor
15H	Mobile Genuine Intel® processor
16H	Intel® Pentium® M processor
17H	Mobile Intel® Celeron® processor ¹
18H – 0FFH	RESERVED

NOTES:

1. Indicates versions of these processors that were introduced after the Pentium III.

21.3 CPUID LEAVES

The remainder of this chapter provides CPUID enumeration information for Intel® 64 and IA-32 architectures.

CPUID.00H -- Maximum Input for Basic CPUID and Vendor ID

CPUID.00H returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register and is processor specific.

- This leaf is always valid.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-4. Leaf 00H Maximum Input for Basic CPUID and Vendor ID

Register	Field Name	Description	Domain
EAX[31:0]	MAX_LEAF	Maximum input value for basic CPUID Information.	Platform
EBX[31:0]	VENDOR_ID_1	"Genu"	Platform
ECX[31:0]	VENDOR_ID_2	"ntel"	Platform
EDX[31:0]	VENDOR_ID_3	"inel"	Platform

A vendor identification string is also returned in EBX, EDX, and ECX. For Intel processors, the string is "GenuineIntel" and is expressed:

EBX := 756e6547h (* "Genu", with G in the low eight bits of BL *)

EDX := 49656e69h (* "ineI", with i in the low eight bits of DL *)

ECX := 6c65746eh (* "ntel", with n in the low eight bits of CL *)

CPUID.01H -- Version and Features

CPUID.01H returns type, family, model, stepping, and feature information.

- This leaf is valid if MAX_LEAF $\geq$ 01H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-5. Leaf 01H Output Registers

CPUID Output Registers	Description
EAX[31:0]	Version information: Type, Family, Model, and Stepping ID (see “CPUID.01H:EAX—Version Information: Type, Family, Model and Stepping ID”).
EBX[31:0]	Feature information (see “CPUID.01H:EBX—Feature Information”).
ECX[31:0]	Feature information (see “CPUID.01H:ECX—Feature Information”).
EDX[31:0]	Feature information (see “CPUID.01H:EDX—Feature Information”).

CPUID.01H:EAX Version Information: Type, Family, Model and Stepping ID

The EAX register of CPUID.01H returns the information shown below.

Table 21-6. Leaf 01H Version and Features Returned in EAX

Register	Field Name	Description	Domain
EAX[3:0]	STEPPING_ID	Identifies a revision of the specific processor family and model. The stepping information is specified as a per-package basis for legacy processors. More recent processors do not allow mixing steppings.	Package
EAX[7:4]	MODEL_ID	Identifies a set of processors within a family. Certain models of Pentium® 4 processors allowed mixed Model IDs and would have this identified as a Package Domain.	Platform
EAX[11:8]	FAMILY_ID	Identifies a set of processors that have a general architectural similarity.	Platform
EAX[13:12]	PROCESSOR_TYPE	Identifies specific type of processor.	Platform
EAX[15:14]	Reserved	Reserved.	
EAX[19:16]	EXTENDED_MODEL_ID	When the Family ID is 06H or 0FH, this field is prepended to the Model ID to provide an 8-bit model identification.	Platform
EAX[27:20]	EXTENDED_FAMILY_ID	When the Family ID is 0FH, this field is added to the Family ID to provide an 8-bit family identification.	Platform
EAX[31:28]	Reserved	Reserved.	

CPUID.01H returns version information in EAX. For example: model, family, and processor type for the Intel Xeon processor 5100 series is as follows:

- Model — 1111B
- Family — 0101B
- Processor Type — 00B

See table below for available processor type values. Stepping IDs are provided as needed.

Table 21-7. Processor Type Field

Type	Encoding
Original OEM Processor	00B
Intel OverDrive® Processor	01B
Dual processor (not applicable to Intel486 processors)	10B
Intel reserved	11B

NOTE

See Section 21.1.2, "Identification of Earlier Processors," for information on identifying earlier IA-32 processors. The Extended Family ID needs to be examined only when the Family ID is 0FH. Integrate the fields into a display using the following rule:

```

IF Family_ID ≠ 0FH THEN
    DisplayFamily = Family_ID;
ELSE
    DisplayFamily = Extended_Family_ID + Family_ID;
FI;

```

The Extended Model ID needs to be examined only when the Family ID is 06H or 0FH. Integrate the field into a display using the following rule:

```

IF (Family_ID = 06H or Family_ID = 0FH) THEN
    DisplayModel = (Extended_Model_ID « 4) + Model_ID;
ELSE
    DisplayModel = Model_ID;
FI;

```

CPUID.01H:EBX Feature information.

The EBX register of CPUID.01H returns the information shown below.

Table 21-8. Leaf 01H Version and Features Returned in EBX

Register	Field Name	Description	Domain
EBX[7:0]	BRAND_INDEX	This number provides an entry into a brand string table that contains brand strings for IA-32 processors. More information about this field is provided in Section 21.2.2, "The Processor Brand Index Method."	Platform
EBX[15:8]	CLFLUSH_LINE_SIZE	Value * 8 = cache line size in bytes. This number indicates the size of the cache line flushed by the CLFLUSH and CLFLUSHOPT instructions in 8-byte increments. This field was introduced in the Pentium 4 processor.	Platform
EBX[23:16]	APIC_ID_SPACE	Maximum number of addressable IDs for logical processors in this physical package. The nearest power-of-2 integer that is not smaller than EBX[23:16] is the number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. This field is only valid if CPUID.01H.EDX.HTT[28] = 1.	Platform

EBX[31:24]	INITIAL_APIC_ID	This number is the 8-bit ID that is assigned to the local APIC on the processor during power up. This field was introduced in the Pentium 4 processor. The 8-bit initial APIC ID in EBX[31:24] is replaced by the 32-bit x2APIC ID, available in Leaf 0BH and Leaf 1FH.	Logical Processor

The Maximum Addressable IDs for logical processors in this package should not be used on platforms that support CPUID leaf 0BH or CPUID leaf 1FH as it can be saturated and incorrect. Modern platforms can have many more processors than can be enumerated or have topology domains with discontinuous APIC ID reservations that can only be correctly identified through the usage of CPUID.0BH or CPUID.1FH.

CPUID.01H:ECX Feature Information

The ECX register of CPUID.01H returns the information shown below. For all feature flags, a 1 indicates that the feature is supported. Software should identify Intel as the vendor to properly interpret feature flags. Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

Table 21-9. Leaf 01H Version and Features Returned in ECX

Register	Field Name	Description	Domain
ECX[0]	SSE3	If 1, supports Streaming SIMD Extensions 3.	Platform
ECX[1]	PCLMULQDQ	If 1, supports the PCLMULQDQ instruction.	Platform
ECX[2]	DTES64	64-bit DS Area. If 1, supports DS area using 64-bit layout.	Platform
ECX[3]	MONITOR	If 1, supports the MONITOR/MWAIT and CPUID.05H.	Platform
ECX[4]	DS_CPL	If 1, supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.	Platform
ECX[5]	VMX	If 1, supports the Virtual Machine Extensions.	Platform
ECX[6]	SMX	If 1, supports Safer Mode Extensions. See Chapter 7, "Safer Mode Extensions Reference."	Platform
ECX[7]	EIST	If 1, supports Enhanced Intel SpeedStep® technology.	Platform
ECX[8]	TM2	If 1, supports Thermal Monitor 2.	Platform
ECX[9]	SSSE3	If 1, supports Supplemental Streaming SIMD Extensions 3.	Platform
ECX[10]	L1_CONTEXT_ID	If 1, the L1 data cache mode can be set to either adaptive mode or shared mode. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.	Platform
ECX[11]	DEBUG_INTERFACE	If 1, supports IA32_DEBUG_INTERFACE MSR for silicon debug.	Platform
ECX[12]	FMA	If 1, supports FMA extensions using YMM state.	Platform

ECX[13]	CMPXCHG16B	If 1, supports this instruction. See the “CMPXCHG8B/CMPXCHG16B—Compare and Exchange Bytes” section in this chapter for a description.	Platform
ECX[14]	XTPR_UPDATE_CONTROL	If 1, supports changing IA32_MISC_ENABLE[bit 23].	Platform
ECX[15]	PERF_CAPABILITIES	If 1, supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.	Platform
ECX[16]	Reserved	Reserved.	
ECX[17]	PCID	If 1, supports Process-context identifiers and software setting CR4.PCIDE to 1.Process-context identifiers.	Platform
ECX[18]	DCA	If 1, supports the ability to prefetch data from a memory mapped device. See CPUID.09H.	Platform
ECX[19]	SSE4_1	If 1, supports SSE4.1.	Platform
ECX[20]	SSE4_2	If 1, supports SSE4.2.	Platform
ECX[21]	X2APIC	If 1, supports x2APIC feature.	Platform
ECX[22]	MOVBE	If 1, supports MOVBE instruction.	Platform
ECX[23]	POPCNT	If 1, supports the POPCNT instruction.	Platform
ECX[24]	TSC_DEADLINE	If 1, the processor’s local APIC timer supports one-shot operation using a TSC deadline value.	Platform
ECX[25]	AESNI	If 1, supports the AESNI instruction extensions.	Platform
ECX[26]	XSAVE	If 1, supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCRO.	Platform
ECX[27]	OSXSAVE	If 1, the OS has set CR4.OSXSAVE[bit 18] to enable XSETBV/XGETBV instructions to access XCRO and to support processor extended state management using XSAVE/XRSTO.	Logical Processor
ECX[28]	AVX	If 1, supports the AVX instruction extensions.	Platform
ECX[29]	F16C	If 1, supports 16-bit floating-point conversion instructions.	Platform
ECX[30]	RDRAND	If 1, supports RDRAND instruction.	Platform
ECX[31]	Not Used	Always returns 0.	Platform

CPUID.01H:EDX Feature Information

The EDX register of CPUID.01H returns the information shown below. For all feature flags, a 1 indicates that the feature is supported. Software should identify Intel as the vendor to properly interpret feature flags. Software must confirm that a processor feature is present using feature flags returned by CPUID prior to using the feature. Software should not depend on future offerings retaining all features.

Table 21-10. Leaf 01H Version and Features Returned in EDX

Register	Field Name	Description	Domain
EDX[0]	FPU	Floating Point Unit On-Chip. The processor contains an x87 FPU.	Platform

EDX[1]	VME	If 1, supports Virtual 8086 mode enhancements, including CR4.VME for controlling the feature; CR4.PVI for protected mode virtual interrupts; software interrupt indirection; expansion of the TSS with the software indirection bitmap; and EFLAGS.VIF and EFLAGS.VIP flags.	Platform
EDX[2]	DE	If 1, supports I/O breakpoints debugging extensions, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.	Platform
EDX[3]	PSE	P If 1, supports page size extensions for large pages of size 4 MByte, including: CR4.PSE for controlling the feature; the defined dirty bit in PDE (Page Directory Entries); optional reserved bit trapping in CR3; PDEs; and PTEs.	Platform
EDX[4]	TSC	If 1, supports the Time Stamp Counter, RDTSC instruction, including CR4.TSD for controlling privilege.	Platform
EDX[5]	MSR	If 1, supports the Model Specific Registers RDMSR and WRMSR Instructions. Some of the MSRs are implementation dependent.	Platform
EDX[6]	PAE	If 1, supports the Physical Address Extension which is for physical addresses greater than 32 bits, including: extended page table entry formats; an extra level in the page translation tables; and 2-MByte pages rather than 4 Mbyte pages.	Platform
EDX[7]	MCE	If 1, supports exception 18 for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the modelspecific implementations of machine-check error logging, reporting, and processor shutdowns. Machine Check exception handlers may have to depend on processor version to do model specific processing of the exception, or test for the presence of the Machine Check feature.	Platform
EDX[8]	CMPXCHG8B	If 1, supports the CMPXCHG8B (64 bits) Instruction, implicitly locked and atomic.	Platform
EDX[9]	APIC	If 1, the processor contains an Advanced Programmable Interrupt Controller (APIC), responding to memory mapped commands in the physical address range FFFE0000H to FFFE0FFFH (by default - some processors permit the APIC to be relocated).	Platform
EDX[10]	Reserved	Reserved.	
EDX[11]	SEP	If 1 supports the SYSENTER and SYSEXIT Instructions and associated MSRs.	Platform

EDX[12]	MTRR	If 1, supports the Memory Type Range Registers. (The MTRRcap MSR contains feature bits that describe what memory types are supported, how many variable MTRRs are supported, and whether fixed MTRRs are supported.)	Platform
EDX[13]	PGE	If 1, supports the global bit in paging-structure entries that map a page, indicating TLB entries that are common to different processes and need not be flushed. The CR4.PGE bit controls this feature.	Platform
EDX[14]	MCA	If 1, supports the Machine Check Architecture feature. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.	Platform
EDX[15]	CMOV	If 1, supports the Conditional Move Instructions. If CPUID.01H:EDX.FPU[0] (x87 FPU present) is 1 also, supports the FCOMI and FCMOV instructions.	Platform
EDX[16]	PAT	If 1, supports the Page Attribute Table feature. (This feature augments the Memory Type Range Registers (MTRRs), allowing an operating system to specify attributes of memory accessed through a linear address on a 4KB granularity.)	Platform
EDX[17]	PSE_36	If 1, supports the 36-Bit Page Size Extension which enables 4-MByte pages addressing physical memory beyond 4 GBytes with 32-bit paging. This feature indicates that upper bits of the physical address of a 4-MByte page are encoded in bits 20:13 of the page directory entry. Such physical addresses are limited by MAXPHYADDR and may be up to 40 bits in size.	Platform
EDX[18]	PSN	If 1, supports the 96-bit Processor Serial Number identification number feature, and the feature is enabled. Available only in Pentium III, see CPUID.03H.	Platform
EDX[19]	CLFLUSH	If 1, supports the CLFLUSH instruction.	Platform
EDX[20]	Reserved	Reserved.	
EDX[21]	DEBUG_STORE	If 1, supports the Debug Store feature which provides the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and processor event-based sampling (PEBS) facilities (see Chapter 25, "Introduction to Virtual Machine Extensions," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C).	Platform
EDX[22]	ACPI	If 1, supports the Thermal Monitor and Software Controlled Clock Facilities. These are internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.	Platform

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

EDX[23]	MMX	If 1, supports the Intel MMX Technology.	Platform
EDX[24]	FXSR	If 1, supports the FXSAVE and FXRSTOR Instructions, which are fast save and restore of the floating-point context, and the availability of CR4.OSFXSR for an operating system to indicate support of same.	Platform
EDX[25]	SSE	If 1, supports SSE.	Platform
EDX[26]	SSE2	If 1, supports SSE2.	Platform
EDX[27]	SELF_SNOOP	If 1, supports Self Snoop which is the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.	Platform
EDX[28]	HTT	If 1, supports Intel Hyper-threading Technology (HTT) and CPUID.01H:EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package). This topology enumeration has been superseded by CPUID.0BH and/or CPUID.1FH on platforms which enumerate either leaf.	Platform
EDX[29]	TM	If 1, supports the Thermal Monitor feature in which the processor implements the thermal monitor automatic thermal control circuitry (TCC).Thermal Monitor.	Platform
EDX[30]	Reserved	Reserved.	
EDX[31]	PBE	If 1, supports the Pending Break Enable feature, which is the use of the FERR#/PBE# pin when the processor is in the stop-clock state (STPCLK# is asserted) to signal the processor that an interrupt is pending and that the processor should return to normal operation to handle the interrupt.	Platform

CPUID.02H -- TLB/Cache/Prefetch Information

CPUID.02H returns TLB, cache, and prefetch information. This leaf has been superseded by CPUID.04H for cache enumeration and CPUID.18H for TLB enumeration. These processors will also report new descriptor values of types 0FEh for 0FFh to refer enumerations to CPUID.04H and CPUID.18H.

- This leaf is valid if MAX_LEAF $\geq$ 02H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-11. Leaf 02H TLB/Cache/Prefetch Information

Register	Field Name	Description	Domain
EAX[7:0]	Reserved	Reserved with a value of 1	
EAX[15:8]	DESCRIPTOR_1	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EAX[23:16]	DESCRIPTOR_2	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EAX[31:24]	DESCRIPTOR_3	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EBX[7:0]	DESCRIPTOR_4	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EBX[15:8]	DESCRIPTOR_5	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EBX[23:16]	DESCRIPTOR_6	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EBX[31:24]	DESCRIPTOR_7	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
ECX[7:0]	DESCRIPTOR_8	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
ECX[15:8]	DESCRIPTOR_9	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
ECX[23:16]	DESCRIPTOR_10	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
ECX[31:24]	DESCRIPTOR_11	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EDX[7:0]	DESCRIPTOR_12	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EDX[15:8]	DESCRIPTOR_13	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EDX[23:16]	DESCRIPTOR_14	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor
EDX[31:24]	DESCRIPTOR_15	See Table “Encoding of CPUID Leaf 2 Descriptors” below this table.	Logical Processor

CPUID.02H returns information about the processor’s internal TLBs, cache, and prefetch hardware in the EAX, EBX, ECX, and EDX registers. The information is reported in encoded form and fall into the following categories:

- The least-significant byte in register EAX (register AL) will always return 01H. Software should ignore this value and not interpret it as an informational descriptor.
- The most significant bit (bit 31) of each register indicates whether the register contains valid information (set to 0) or is reserved (set to 1).
- If a register contains valid information, the information is contained in 1-byte descriptors. There are four types of encoding values for the byte descriptor, the encoding of these descriptors and encoding type are listed in the table

below.

Note that the order of descriptors in the EAX, EBX, ECX, and EDX registers is not defined; that is, specific bytes are not designated to contain descriptors for specific cache, prefetch, or TLB types. The descriptors may appear in any order. Note also a processor may report a general descriptor type (FFH)1 and (FEH)1 and not report any byte descriptor of “cache type” or “*TLB type” via CPUID.02H.

Table 21-12. Encoding of CPUID Leaf 2 Descriptors

Descriptor Value	Type	Cache or TLB Description
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size.
0BH	TLB	Instruction TLB: 4 MByte pages, 4-way set associative, 4 entries.
0CH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 32 byte line size.
0DH	Cache	1st-level data cache: 16 KBytes, 4-way set associative, 64 byte line size.
0EH	Cache	1st-level data cache: 24 KBytes, 6-way set associative, 64 byte line size.
1DH	Cache	2nd-level cache: 128 KBytes, 2-way set associative, 64 byte line size.
21H	Cache	2nd-level cache: 256 KBytes, 8-way set associative, 64 byte line size.
22H	Cache	3rd-level cache: 512 KBytes, 4-way set associative, 64 byte line size, 2 lines per sector.
23H	Cache	3rd-level cache: 1 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
24H	Cache	2nd-level cache: 1 MBytes, 16-way set associative, 64 byte line size.
25H	Cache	3rd-level cache: 2 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
29H	Cache	3rd-level cache: 4 MBytes, 8-way set associative, 64 byte line size, 2 lines per sector.
2CH	Cache	1st-level data cache: 32 KBytes, 8-way set associative, 64 byte line size.
30H	Cache	1st-level instruction cache: 32 KBytes, 8-way set associative, 64 byte line size.
40H	Cache	No 2nd-level cache or, if processor contains a valid 2nd-level cache, no 3rd-level cache.
41H	Cache	2nd-level cache: 128 KBytes, 4-way set associative, 32 byte line size.
42H	Cache	2nd-level cache: 256 KBytes, 4-way set associative, 32 byte line size.
43H	Cache	2nd-level cache: 512 KBytes, 4-way set associative, 32 byte line size.
44H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 32 byte line size.
45H	Cache	2nd-level cache: 2 MByte, 4-way set associative, 32 byte line size.
46H	Cache	3rd-level cache: 4 MByte, 4-way set associative, 64 byte line size.
47H	Cache	3rd-level cache: 8 MByte, 8-way set associative, 64 byte line size.
48H	Cache	2nd-level cache: 3MByte, 12-way set associative, 64 byte line size.
49H	Cache	3rd-level cache: 4MB, 16-way set associative, 64-byte line size (Intel Xeon processor MP, Family 0FH, Model 06H) 2nd-level cache: 4 MByte, 16-way set associative, 64 byte line size.
4AH	Cache	3rd-level cache: 6MByte, 12-way set associative, 64 byte line size.
4BH	Cache	3rd-level cache: 8MByte, 16-way set associative, 64 byte line size.
4CH	Cache	3rd-level cache: 12MByte, 12-way set associative, 64 byte line size.
4DH	Cache	3rd-level cache: 16MByte, 16-way set associative, 64 byte line size.
4EH	Cache	2nd-level cache: 6MByte, 24-way set associative, 64 byte line size.
4FH	TLB	Instruction TLB: 4 KByte pages, 32 entries.
50H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 64 entries.
51H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 128 entries.
52H	TLB	Instruction TLB: 4 KByte and 2-MByte or 4-MByte pages, 256 entries.

55H	TLB	Instruction TLB: 2-MByte or 4-MByte pages, fully associative, 7 entries.
56H	TLB	Data TLB0: 4 MByte pages, 4-way set associative, 16 entries.
57H	TLB	Data TLB0: 4 KByte pages, 4-way associative, 16 entries.
59H	TLB	Data TLB0: 4 KByte pages, fully associative, 16 entries.
5AH	TLB	Data TLB0: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries.
5BH	TLB	Data TLB: 4 KByte and 4 MByte pages, 64 entries.
5CH	TLB	Data TLB: 4 KByte and 4 MByte pages, 128 entries.
5DH	TLB	Data TLB: 4 KByte and 4 MByte pages, 256 entries.
60H	Cache	1st-level data cache: 16 KByte, 8-way set associative, 64 byte line size.
61H	TLB	Instruction TLB: 4 KByte pages, fully associative, 48 entries.
63H	TLB	Data TLB: 2 MByte or 4 MByte pages, 4-way set associative, 32 entries and a separate array with 1 GByte pages, 4-way set associative, 4 entries.
64H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 512 entries.
66H	Cache	1st-level data cache: 8 KByte, 4-way set associative, 64 byte line size.
67H	Cache	1st-level data cache: 16 KByte, 4-way set associative, 64 byte line size.
68H	Cache	1st-level data cache: 32 KByte, 4-way set associative, 64 byte line size.
6AH	Cache	uTLB: 4 KByte pages, 8-way set associative, 64 entries.
6BH	Cache	DTLB: 4 KByte pages, 8-way set associative, 256 entries.
6CH	Cache	DTLB: 2M/4M pages, 8-way set associative, 128 entries.
6DH	Cache	DTLB: 1 GByte pages, fully associative, 16 entries.
70H	Cache	Trace cache: 12 K-?op, 8-way set associative.
71H	Cache	Trace cache: 16 K-?op, 8-way set associative.
72H	Cache	Trace cache: 32 K-?op, 8-way set associative.
76H	TLB	Instruction TLB: 2M/4M pages, fully associative, 8 entries.
78H	Cache	2nd-level cache: 1 MByte, 4-way set associative, 64byte line size.
79H	Cache	2nd-level cache: 128 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7AH	Cache	2nd-level cache: 256 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7BH	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7CH	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size, 2 lines per sector.
7DH	Cache	2nd-level cache: 2 MByte, 8-way set associative, 64byte line size.
7FH	Cache	2nd-level cache: 512 KByte, 2-way set associative, 64-byte line size.
80H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 64-byte line size.
82H	Cache	2nd-level cache: 256 KByte, 8-way set associative, 32 byte line size.
83H	Cache	2nd-level cache: 512 KByte, 8-way set associative, 32 byte line size.
84H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 32 byte line size.
85H	Cache	2nd-level cache: 2 MByte, 8-way set associative, 32 byte line size.
86H	Cache	2nd-level cache: 512 KByte, 4-way set associative, 64 byte line size.
87H	Cache	2nd-level cache: 1 MByte, 8-way set associative, 64 byte line size.
A0H	DTLB	DTLB: 4k pages, fully associative, 32 entries.
B0H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 128 entries.

B1H	TLB	Instruction TLB: 2M pages, 4-way, 8 entries or 4M pages, 4-way, 4 entries.
B2H	TLB	Instruction TLB: 4KByte pages, 4-way set associative, 64 entries.
B3H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 128 entries.
B4H	TLB	Data TLB1: 4 KByte pages, 4-way associative, 256 entries.
B5H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 64 entries.
B6H	TLB	Instruction TLB: 4KByte pages, 8-way set associative, 128 entries.
BAH	TLB	Data TLB1: 4 KByte pages, 4-way associative, 64 entries.
C0H	TLB	Data TLB: 4 KByte and 4 MByte pages, 4-way associative, 8 entries.
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries.
C2H	DTLB	DTLB: 4 KByte/2 MByte pages, 4-way associative, 16 entries.
C3H	STLB	Shared 2nd-Level TLB: 4 KByte /2 MByte pages, 6-way associative, 1536 entries. Also 1GByte pages, 4-way, 16 entries.
C4H	DTLB	DTLB: 2M/4M Byte pages, 4-way associative, 32 entries.
CAH	STLB	Shared 2nd-Level TLB: 4 KByte pages, 4-way associative, 512 entries.
D0H	Cache	3rd-level cache: 512 KByte, 4-way set associative, 64 byte line size.
D1H	Cache	3rd-level cache: 1 MByte, 4-way set associative, 64 byte line size.
D2H	Cache	3rd-level cache: 2 MByte, 4-way set associative, 64 byte line size.
D6H	Cache	3rd-level cache: 1 MByte, 8-way set associative, 64 byte line size.
D7H	Cache	3rd-level cache: 2 MByte, 8-way set associative, 64 byte line size.
D8H	Cache	3rd-level cache: 4 MByte, 8-way set associative, 64 byte line size.
DCH	Cache	3rd-level cache: 1.5 MByte, 12-way set associative, 64 byte line size.
DDH	Cache	3rd-level cache: 3 MByte, 12-way set associative, 64 byte line size.
DEH	Cache	3rd-level cache: 6 MByte, 12-way set associative, 64 byte line size.
E2H	Cache	3rd-level cache: 2 MByte, 16-way set associative, 64 byte line size.
E3H	Cache	3rd-level cache: 4 MByte, 16-way set associative, 64 byte line size.
E4H	Cache	3rd-level cache: 8 MByte, 16-way set associative, 64 byte line size.
EAH	Cache	3rd-level cache: 12MByte, 24-way set associative, 64 byte line size.
EBH	Cache	3rd-level cache: 18MByte, 24-way set associative, 64 byte line size.
ECH	Cache	3rd-level cache: 24MByte, 24-way set associative, 64 byte line size.
F0H	Prefetch	64-Byte prefetching.
F1H	Prefetch	128-Byte prefetching.
FEH	General	CPUID leaf 2 does not report TLB descriptor information; use CPUID leaf 18H to query TLB and other address translation parameters.
FFH	General	CPUID leaf 2 does not report cache descriptor information, use CPUID leaf 4 to query cache parameters.

Example 21-1. Example of Cache and TLB Interpretation

The first member of the family of Pentium 4 processors returns the following information about caches and TLBs when the CPUID executes with an input value of 2:
EAX 66 5B 50 01H

EBX 0H
ECX 0H
EDX 00 7A 70 00H

Which means:

- The least-significant byte (byte 0) of register EAX is set to 01H. This value should be ignored.
- The most-significant bit of all four registers (EAX, EBX, ECX, and EDX) is set to 0, indicating that each register contains valid 1-byte descriptors.
- Bytes 1, 2, and 3 of register EAX indicate that the processor has:
 - 50H - a 64-entry instruction TLB, for mapping 4-KByte and 2-MByte or 4-MByte pages.
 - 5BH - a 64-entry data TLB, for mapping 4-KByte and 4-MByte pages.
 - 66H - an 8-KByte 1st level data cache, 4-way set associative, with a 64-Byte cache line size.
- The descriptors in registers EBX and ECX are valid, but contain NULL descriptors.
- Bytes 0, 1, 2, and 3 of register EDX indicate that the processor has:
 - 00H - NULL descriptor.
 - 70H - Trace cache: 12 K-μop, 8-way set associative.
 - 7AH - a 256-KByte 2nd level cache, 8-way set associative, with a sectored, 64-byte cache line size.
 - 00H - NULL descriptor.

CPUID.03H -- Processor Serial Number

CPUID.03H returns the processor serial number, if available. Processor serial number (PSN) is not supported in the Pentium 4 processor or later.

- This leaf is valid if MAX_LEAF ≥ 03H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-13. Leaf 03H Processor Serial Number

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	PSN_31_0	Bits 00-31 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)	Package
EDX[31:0]	PSN_63_32	Bits 32-63 of 96-bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)	Package

CPUID.04H -- Deterministic Cache Parameters

CPUID.04H returns the deterministic cache parameters for each cache level.

- This leaf is valid if CPUID.04H.00H:EAX[4:0] $\neq$ 0 and MAX_LEAF $\geq$ 04H.
- The sub-leaves are enumerated until sub-leaf n returns 0 in EAX[4:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EAX[4:0] as 0.

Table 21-14. Leaf 04H Deterministic Cache Parameters

Register	Field Name	Description	Domain
EAX[4:0]	CACHE_TYPE	0 = Null, no more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved.	Logical Processor
EAX[7:5]	CACHE_LEVEL	Cache level (starts at 1).	Logical Processor
EAX[8]	SELF_INITIALIZING_CACHE	Self initializing cache level (does not need software initialization).	Logical Processor
EAX[9]	FULLY_ASSOC	Fully associative cache.	Logical Processor
EAX[13:10]	Reserved	Reserved.	
EAX[25:14]	MAX_LP_ADDRESSABLE_IDS	Maximum number of addressable IDs for logical processors sharing this cache. Add one to the return value to get the result. The nearest power-of-2 integer that is not smaller than (1 + EAX[25:14]) is the number of unique initial APIC IDs reserved for addressing different logical processors sharing this cache.	Logical Processor
EAX[31:26]	MAX_CORES_ADDRESSABLE_IDS_PKG	Maximum number of addressable IDs for processor cores in the physical package. Add one to the return value to get the result. The nearest power-of-2 integer that is not smaller than (1 + EAX[31:26]) is the number of unique Core_IDs reserved for addressing different processor cores in a physical package. Core ID is a subset of bits of the initial APIC ID. The returned value is constant for valid initial values in ECX. Valid ECX values start from 0. The maximum number of addressable IDs for processor cores in the physical package field may contain a saturated value and will not correctly identify the addressable ID reservations for cores in this package on processors where CPUID.0BH and/or CPUID.1FH exist. Processors which enumerate topology information in either CPUID.0BH or CPUID.1FH need to use those leaves to obtain the correct topology details.	Platform
EBX[11:0]	LINE_SIZE	System Coherency Line Size. Add one to the return value to get the result.	Platform
EBX[21:12]	PHYS_LINE_PARTITIONS	Physical line partitions. Add one to the return value to get the result.	Logical Processor
EBX[31:22]	NUM_WAYS	Ways of associativity. Add one to the return value to get the result.	Logical Processor

ECX[31:0]	NUM_SETS	Number of sets. Add one to the return value to get the result.	Logical Processor
EDX[0]	NOT_LWR_CACHE_FLUSH	0 = WBINVD/INVD from threads sharing this cache acts upon lower level caches for threads sharing this cache. 1 = WBINVD/INVD is not guaranteed to act upon lower level caches of non-originating threads sharing this cache.	Logical Processor
EDX[1]	INCLUSIVE_CACHE	0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels.	Logical Processor
EDX[2]	COMPLEX_CACHE_INDEXING	0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits.	Logical Processor
EDX[31:3]	Reserved	Reserved.	

When CPUID executes with EAX set to 04H and ECX contains an index value, the processor returns encoded data that describe a set of deterministic cache parameters (for the cache level associated with the input in ECX). Valid index values start from 0. Software can enumerate the deterministic cache parameters for each level of the cache hierarchy starting with an index value of 0, until the parameters report the value associated with the cache type field is 0.

This Cache Size in Bytes

$$= (\text{Ways} + 1) * (\text{Partitions} + 1) * (\text{Line\_Size} + 1) * (\text{Sets} + 1)$$

$$= (\text{EBX}[31:22] + 1) * (\text{EBX}[21:12] + 1) * (\text{EBX}[11:0] + 1) * (\text{ECX} + 1)$$

The CPUID.04H also reports data that can be used to derive the topology of processor cores in a physical package on legacy processors. This information is constant for all valid index values. Software can query the raw data reported by executing CPUID with EAX=04H and ECX=0 and use it as part of the topology enumeration algorithm on processors that do not enumerate either CPUID.0BH or CPUID.1FH as described in Chapter 10, “Multiple-Processor Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

CPUID.05H -- MONITOR and MWAIT Features

CPUID.05H returns the MONITOR and MWAIT feature information.

- This leaf is valid if MAX_LEAF $\geq$ 05H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-15. Leaf 05H MONITOR and MWAIT Features

Register	Field Name	Description	Domain
EAX[15:0]	SMALLEST_MONITOR_LINE_SIZE	Smallest monitor-line size in bytes (default is processor's monitor granularity).	Platform
EAX[31:16]	Reserved	Reserved.	
EBX[15:0]	LARGEST_MONITOR_LINE_SIZE	Largest monitor-line size in bytes (default is processor's monitor granularity).	Platform
EBX[31:16]	Reserved	Reserved.	
ECX[0]	MONITOR_MWAIT_EXTENSIONS	If 1, supports enumeration of MONITOR/MWAIT extensions (beyond EAX and EBX registers).	Platform
ECX[1]	INTERRUPT_AS_BREAK_EVENT	If 1, supports treating interrupts as break-event for MWAIT, even when interrupts are disabled.	Platform
ECX[31:2]	Reserved	Reserved.	
EDX[3:0]	C0_SUB_STATES	Number of C0* sub C-states supported using MWAIT.	Platform
EDX[7:4]	C1_SUB_STATES	Number of C1* sub C-states supported using MWAIT.	Platform
EDX[11:8]	C2_SUB_STATES	Number of C2* sub C-states supported using MWAIT.	Platform
EDX[15:12]	C3_SUB_STATES	Number of C3* sub C-states supported using MWAIT.	Platform
EDX[19:16]	C4_SUB_STATES	Number of C4* sub C-states supported using MWAIT.	Platform
EDX[23:20]	C5_SUB_STATES	Number of C5* sub C-states supported using MWAIT.	Platform
EDX[27:24]	C6_SUB_STATES	Number of C6* sub C-states supported using MWAIT.	Platform
EDX[31:28]	C7_SUB_STATES	Number of C7* sub C-states supported using MWAIT.	Platform

NOTE

The definition of C0 through C7 states for MWAIT extensions are processor-specific C-states, not ACPI C-state.

CPUID.05H returns information about features available to MONITOR/MWAIT instructions. The MONITOR instruction is used for address-range monitoring in conjunction with MWAIT instruction. The MWAIT instruction optionally provides additional extensions for advanced power management.

CPUID.06H -- Thermal and Power Management Features

CPUID.06H returns information about thermal and power management features.

- This leaf is valid if MAX_LEAF ≥ 06H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-16. Leaf 06H Thermal and Power Management Features

Register	Field Name	Description	Domain
EAX[0]	DIGITAL_TEMP_SENSOR	If 1, supports Digital Temperature Sensor.	Platform
EAX[1]	TURBO_BOOST	If 1, supports Intel Turbo Boost Technology. (see description of IA32_MISC_ENABLE[38]).	Platform
EAX[2]	ALWAYS_RUNNING_APIC_TIMER	If 1, supports APIC-Timer-always-running feature.	Platform
EAX[3]	Reserved	Reserved.	
EAX[4]	POWER_LIMIT_NOTIFY	If 1, supports power limit notification controls.	Platform
EAX[5]	EXT_CLOCK_MOD	If 1, supports clock modulation duty cycle extension.	Platform
EAX[6]	PKG_THERM_MGMT	If 1, supports package thermal management.	Platform
EAX[7]	HWP	If 1, supports HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS).	Platform
EAX[8]	HWP_INTERRUPT	If 1, supports IA32_HWP_INTERRUPT MSR.	Platform
EAX[9]	HWP_ACTIVITY_WINDOW	If 1, supports IA32_HWP_REQUEST[bits 41:32].	Platform
EAX[10]	HWP_EPP	If 1, supports IA32_HWP_REQUEST[bits 31:24].	Platform
EAX[11]	HWP_REQUEST_PKG	If 1, supports IA32_HWP_REQUEST_PKG MSR.	Platform
EAX[12]	Reserved	Reserved.	
EAX[13]	HDC	If 1, supports HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, and IA32_THREAD_STALL MSRs.	Platform
EAX[14]	TURBO_BOOST_MAX	If 1, supports Intel® Turbo Boost Max Technology 3.0.	Platform
EAX[15]	HWP_CAP	If 1, supports Highest Performance change capability.	Platform
EAX[16]	HWP_PECI_OVERRIDE	If 1, supports HWP Peci override.	Platform
EAX[17]	FLEXIBLE_HWP	If 1, supports Flexible HWP.	Platform
EAX[18]	HWP_REQUEST_FAST_ACCESS	If 1, supports Fast access mode for the IA32_HWP_REQUEST MSR.	Platform
EAX[19]	HW_FEEDBACK	If 1, supports IA32_HW_FEEDBACK_PTR MSR, IA32_HW_FEEDBACK_CONFIG MSR, IA32_PACKAGE_THERM_STATUS MSR bit 26, and IA32_PACKAGE_THERM_INTERRUPT MSR bit 25.	Platform
EAX[20]	HWP_REQUEST_IGNORE_IDLE	If 1, supports Ignoring Idle Logical Processor HWP request.	Platform
EAX[21]	Reserved	Reserved.	
EAX[22]	HWP_CTL	If 1, supports IA32_HWP_CTL MSR.	Platform

EAX[23]	THREAD_DIRECTOR	If 1, supports Intel® Thread Director. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set.	Platform
EAX[31:24]	Reserved	Reserved.	
EBX[3:0]	DTS_NUM_INT_THRESHOLDS	Number of Interrupt Thresholds in Digital Thermal Sensor.	Platform
EBX[31:4]	Reserved	Reserved.	
ECX[0]	HW_FEEDBACK_CAP	If 1, supports IA32_MPERF and IA32_APERF which provide a measure of delivered processor performance (since last reset of the counters), as a percentage of the expected processor performance when running at the TSC frequency.	Platform
ECX[2:1]	Reserved	Reserved.	
ECX[3]	ENERGY_PERF_BIAS	If 1, supports performance-energy bias preference and a new architectural MSR called IA32_ENERGY_PERF_BIAS (1BOH).	Platform
ECX[7:4]	Reserved	Reserved.	
ECX[15:8]	HW_FEEDBACK_NUM_CLASSES	Number of Intel® Thread Director classes supported by the processor. Information for that many classes is written into the Intel Thread Director Table by the hardware.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[7:0]	HW_FEEDBACK_CAPS	Bitmap of supported hardware feedback interface capabilities. 0 = If 1, supports performance capability reporting. 1 = If 1, supports energy efficiency capability reporting. 2-7 = Reserved. Bits 0 and 1 will always be set together.	Package
EDX[11:8]	HW_FEEDBACK_TABLE_SIZE	Enumerates the size of the hardware feedback interface structure in number of 4 KB pages. Add one to the return value to get the result.	Package
EDX[15:12]	Reserved	Reserved.	
EDX[31:16]	HW_FEEDBACK_TABLE_INDEX	Index (starting at 0) of this logical processor's row in the hardware feedback interface structure. Note that on some parts the index may be same for multiple logical processors. On some parts the indices may not be contiguous, i.e., there may be unused rows in the hardware feedback interface structure.	Logical Processor

Details around these features are described in Chapter 16, “Power and Thermal Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B.

CPUID.07H -- Structured Extended Feature Flags

CPUID.07H returns structured extended feature flags enumeration information. The sub-sections of Section provide leaf 07H information.

- This leaf is valid if MAX_LEAF $\geq$ 07H.
- The maximum sub-leaf value for ECX is specified in CPUID.07H.00H.EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid Sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.07H.00H -- Structured Extended Feature Flags Main Sub-Leaf

CPUID.07H.00H returns the maximum input value of the highest leaf 07H sub-leaf; and EBX, ECX, and EDX contain information of extended feature flags.

Table 21-17. Leaf 07H Sub-Leaf (ECX=0) Output Registers

CPUID Output Registers	Field Name	Description	CPUID Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value for supported leaf 07H subleaves.	Platform
EBX[31:0]		Extended Feature Flags Information in EBX (see "CPUID.07H.00H:EBX—Extended Feature Flags Information")	
ECX[31:0]		Extended Feature Flags Information in ECX (see "CPUID.07H.00H:ECX—Extended Feature Flags Information")	
EDX[31:0]		Extended Feature Flags Information in EDX (see "CPUID.07H.00H:EDX—Extended Feature Flags Information")	

Table 21-18. Leaf 07H.00H Structured Extended Feature Flags Returned in EAX

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value for supported leaf 07H sub-leaves.	Platform

CPUID.07H.00H:EBX Extended Feature Flags Information

The EBX register of CPUID.07H.00H returns the information shown below.

Table 21-19. Leaf 07H.00H Structured Extended Feature Flags Returned in EBX

Register	Field Name	Description	Domain
EBX[0]	FSGSBASE	If 1, supports RDFSBASE/RDGSBASE/WRFS-BASE/WRGSBASE.	Platform
EBX[1]	TSC_ADJUST	If 1, the IA32_TSC_ADJUST MSR is supported.	Platform
EBX[2]	SGX	If 1, supports Intel® Software Guard Extensions (Intel® SGX Extensions).	Platform
EBX[3]	BMI1	If 1, supports the BMI1 instructions.	Platform
EBX[4]	HLE	If 1, supports the Hardware Lock Elision instruction set.	Platform
EBX[5]	AVX2	If 1, supports Intel® Advanced Vector Extensions 2 (Intel® AVX2).	Platform

EBX[6]	FDP_EXCPTN_ONLY	If 1, the x87 FPU Data Pointer is updated only on x87 exceptions.	Platform
EBX[7]	SMEP	If 1, supports Supervisor-Mode Execution Prevention.	Platform
EBX[8]	BMI2	If 1, supports the BMI2 instructions.	Platform
EBX[9]	ENH_REP_MOVSB_STOSB	If 1, supports Enhanced REP MOVSB/STOSB.	Platform
EBX[10]	INVPCID	If 1, supports INVPCID instruction for system software that manages process-context identifiers.	Platform
EBX[11]	RTM	If 1, supports the Restricted Transactional Memory instruction set.	Platform
EBX[12]	RDT_M	If 1, supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability.	Platform
EBX[13]	FCS_FDS_DEPRECATION	If 1, deprecates FPU CS and FPU DS values.	Platform
EBX[14]	MPX	If 1, supports Intel® Memory Protection Extensions.	Platform
EBX[15]	RDT_A	If 1, supports Intel® Resource Director Technology (Intel® RDT) Allocation capability.	Platform
EBX[16]	AVX512F	If 1, supports the AVX512F instructions.	Platform
EBX[17]	AVX512DQ	If 1, supports the AVX512DQ instructions.	Platform
EBX[18]	RDSEED	If 1, supports the RDSEED instruction.	Platform
EBX[19]	ADX	If 1, supports the ADX instructions.	Platform
EBX[20]	SMAP	If 1, supports Supervisor-Mode Access Prevention and the CLAC/STAC instructions.	Platform
EBX[21]	AVX512_IFMA	If 1, supports the AVX512_IFMA instructions.	Platform
EBX[22]	Reserved	Reserved.	
EBX[23]	CLFLUSHOPT	If 1, supports the CLFLUSHOPT instruction.	Platform
EBX[24]	CLWB	If 1, supports the CLWB instruction.	Platform
EBX[25]	INTEL_PROC_TRACE	If 1, supports Intel® Processor Trace.	Platform
EBX[26]	AVX512PF	If 1, supports the AVX512PF instructions. (Intel® Xeon Phi™ only.)	Platform
EBX[27]	AVX512ER	If 1, supports the AVX512ER instructions. (Intel® Xeon Phi™ only.)	Platform
EBX[28]	AVX512CD	If 1, supports the AVX512CD instructions.	Platform
EBX[29]	SHA	If 1, supports Intel® Secure Hash Algorithm Extensions (Intel® SHA Extensions).	Platform
EBX[30]	AVX512BW	If 1, supports the AVX512BW instructions.	Platform
EBX[31]	AVX512VL	If 1, supports the AVX512VL instructions.	Platform

CPUID.07H.00H:ECX Extended Feature Flags Information

The ECX register of CPUID.07H.00H returns the information shown below.

Table 21-20. Leaf 07H.00H Structured Extended Feature Flags Returned in ECX

Register	Field Name	Description	Domain

ECX[0]	PREFETCHWT1	If 1, supports the PREFETCHWT1 instruction. (Intel® Xeon Phi™ only.)	Platform
ECX[1]	AVX512_VBMI	If 1, supports the AVX512_VBMI instructions.	Platform
ECX[2]	UMIP	If 1, supports user-mode instruction prevention.	Platform
ECX[3]	PKU	If 1, supports protection keys for user-mode pages.	Platform
ECX[4]	OSPKE	If 1, the OS has set CR4.PKE to enable protection keys and the RDPKRU/WRPKRU instructions.	Logical Processor
ECX[5]	WAITPKG	If 1, supports the TPAUSE, UMONITOR, and UMWAIT instructions.	Platform
ECX[6]	AVX512_VBMI2	If 1, supports the AVX512_VBMI2 instructions.	Platform
ECX[7]	CET_SS	If 1, supports CET shadow stack features. Processors that set this bit define bits 1:0 of the IA32_U_CET and IA32_S_CET MSRs. Enumerates support for the following MSRs: IA32_INTERRUPT_SPP_TABLE_ADDR, IA32_PL3_SSP, IA32_PL2_SSP, IA32_PL1_SSP, and IA32_PLO_SSP.	Platform
ECX[8]	GFNI	If 1, supports the GFNI instruction set.	Platform
ECX[9]	VAES	If 1 and Intel AVX supported, supports the VEX-encoded AES instruction set.	Platform
ECX[10]	VPCLMULQDQ	If 1 and Intel AVX supported, supports the VPCLMULQDQ instruction.	Platform
ECX[11]	AVX512_VNNI	If 1, supports the AVX512_VNNI instructions.	Platform
ECX[12]	AVX512_BITALG	If 1, supports the AVX512_BITALG instructions.	Platform
ECX[13]	TME_EN	If 1, the following MSRs are supported: IA32_TME_CAPABILITY, IA32_TME_ACTIVATE, IA32_TME_EXCLUDE_MASK, and IA32_TME_EXCLUDE_BASE.	Platform
ECX[14]	AVX512_VPOPCNTDQ	If 1, supports the AVX512_VPOPCNTDQ instructions.	Platform
ECX[15]	Reserved	Reserved.	
ECX[16]	LA57	If 1, supports 57-bit linear addresses and five-level paging.	Platform
ECX[21:17]	MPX_MAWAU	The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode.	Platform
ECX[22]	RDPID	If 1, RDPID and the IA32_TSC_AUX MSR are available.	Platform
ECX[23]	KEY_LOCKER	If 1, supports Key Locker.	Platform
ECX[24]	BUS_LOCK_DETECT	If 1, indicates support for OS bus-lock detection.	Platform
ECX[25]	CLDEMOTE	If 1, supports cache line demote.	Platform
ECX[26]	Reserved	Reserved.	
ECX[27]	MOVDIRI	If 1, supports the MOVDIRI instruction.	Platform
ECX[28]	MOVDIR64B	If 1, supports the MOVDIR64B instruction.	Platform
ECX[29]	ENQCMD	If 1, supports Enqueue Stores.	Platform

ECX[30]	SGX_LC	If 1, supports SGX Launch Configuration.	Platform
ECX[31]	PKS	If 1, supports protection keys for supervisor-mode pages.	Platform

CPUID.07H.00H:EDX Extended Feature Flags Information

The EDX register of CPUID.07H.00H returns the information shown below.

Table 21-21. Leaf 07H.00H Structured Extended Feature Flags Returned in EDX

Register	Field Name	Description	Domain
EDX[0]	Reserved	Reserved.	
EDX[1]	SGX_KEYS	If 1, supports Attestation Services for Intel® SGX.	Platform
EDX[2]	AVX512_4VNNIW	If 1, supports the AVX512_4VNNIW instructions. (Intel® Xeon Phi™ only.)	Platform
EDX[3]	AVX512_4FMAPS	If 1, supports the AVX512_4FMAPS instructions. (Intel® Xeon Phi™ only.)	Platform
EDX[4]	FAST_SHORT_REP_MOVSB	If 1, supports Fast Short REP MOVSB.	Platform
EDX[5]	UINTR	If 1, supports user interrupts.	Platform
EDX[7:6]	Reserved	Reserved.	
EDX[8]	AVX512_VP2INTERSECT	If 1, supports the AVX512_VP2INTERSECT instruction.	Platform
EDX[9]	MCU_OPT_CTRL	If 1, supports both the IA32_MCU_OPT_CTRL MSR and its bit 0 (RNGDS_MITG_DIS).	Platform
EDX[10]	MD_CLEAR	If 1, supports MD_CLEAR.	Platform
EDX[11]	RTM_ALWAYS_ABORT	If 1, any execution of XBEGIN immediately aborts and transitions to the specified fallback address.	Platform
EDX[12]	Reserved	Reserved.	
EDX[13]	RTM_FORCE_ABORT	If 1, supports RTM_FORCE_ABORT and the IA32_TSX_FORCE_ABORT MSR. These allow software to set IA32_TSX_FORCE_ABORT[0] (RTM_FORCE_ABORT).	Platform
EDX[14]	SERIALIZE	If 1, supports SERIALIZE instruction.	Platform
EDX[15]	HYBRID	If 1, the processor is identified as a hybrid part. If CPUID.00H.MAXLEAF ≥ 1AH and CPUID.1AH:EAX <> 0, then the Native Model ID Enumeration Leaf 1AH exists.	Platform
EDX[16]	TSXLDTRK	If 1, supports Intel TSX suspend/resume of load address tracking.	Platform
EDX[17]	Reserved	Reserved.	
EDX[18]	PCONFIG	If 1, supports the PCONFIG instruction.	Platform
EDX[19]	ARCH_LBRS	If 1, supports architectural LBRs.	Platform
EDX[20]	CET_IBT	If 1, supports CET indirect branch tracking features. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs.	Platform

EDX[21]	Reserved	Reserved.	
EDX[22]	AMX_BF16	If 1, supports tile computational operations on bfloat16 numbers.	Platform
EDX[23]	AVX512_FP16	If 1, supports the FP16 data type with AVX512 instructions.	Platform
EDX[24]	AMX_TILE	If 1, supports tile architecture.	Platform
EDX[25]	AMX_INT8	If 1, supports tile computational operations on 8-bit integers.	Platform
EDX[26]	IBRS_IBPB	If 1, supports indirect branch restricted speculation (IBRS) and the indirect branch predictor barrier (IBPB). Processors that set this bit support the IA32_SPEC_CTRL MSR and the IA32_PRED_CMD MSR. They allow software to set IA32_SPEC_CTRL[0] (IBRS) and IA32_PRED_CMD[0] (IBPB).	Platform
EDX[27]	SPEC_CTRL_ST_PREDICTORS	If 1, supports single thread indirect branch predictors (STIBP). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[1] (STIBP).	Platform
EDX[28]	L1D_FLUSH_INTERFACE	If 1, supports L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_- FLUSH).	Platform
EDX[29]	ARCH_CAPABILITIES	If 1, supports the IA32_ARCH_CAPABILITIES MSR.	Platform
EDX[30]	CORE_CAPABILITIES	If 1, supports the IA32_CORE_CAPABILITIES MSR. IA32_CORE_CAPABILITIES is an architectural MSR that enumerates model-specific features. A bit being set in this MSR indicates that a model specific feature is supported; software must still consult CPUID family/model/stepping to determine the behavior of the enumerated feature as features enumerated in IA32_CORE_CAPABILITIES may have different behavior on different processor models. Some of these features may have behavior that is consistent across processor models (and for which consultation of CPUID family/model/stepping is not necessary); such features are identified explicitly where they are documented in this manual.	Platform
EDX[31]	SPEC_CTRL_SSBD	If 1, supports Speculative Store Bypass Disable (SSBD). Processors that set this bit support the IA32_SPEC_CTRL MSR. They allow software to set IA32_SPEC_CTRL[2] (SSBD).	Platform

CPUID.07H.01H -- Structured Extended Feature Sub-Leaf 1

CPUID.07H.01H:EAX Extended Feature Information

The EAX register of CPUID.07H.01H returns the information shown below.

Table 21-22. Leaf 07H.01H Structured Extended Feature Flags Returned in EAX

Register	Field Name	Description	Domain
EAX[0]	SHA512	If 1, supports the SHA512 instructions.	Platform
EAX[1]	SM3	If 1, supports the SM3 instructions.	Platform
EAX[2]	SM4	If 1, supports the SM4 instructions.	Platform
EAX[3]	Reserved	Reserved.	
EAX[4]	AVX_VNNI	If 1, supports the VEX-encoded versions of the Vector Neural Network Instructions.	Platform
EAX[5]	AVX512_BF16	If 1, supports the Vector Neural Network Instructions supporting BFLOAT16 inputs and conversion instructions from IEEE single precision.	Platform
EAX[6]	LASS	If 1, supports Linear Address Space Separation.	Platform
EAX[7]	CMPCCXADD	If 1, supports the CMPccXADD instruction.	Platform
EAX[8]	ARCH_PERFMON_EXT	If 1, supports ArchPerfmonExt. When set, indicates that the Architectural Performance Monitoring Extended Leaf (EAX=23H) is valid.	Platform
EAX[9]	Reserved	Reserved.	
EAX[10]	FAST_REP_MOVSB	If 1, supports fast zero-length REP MOVSB.	Platform
EAX[11]	FAST_REP_STOSB	If 1, supports fast short REP STOSB.	Platform
EAX[12]	FAST_REP_CMPSB_SCASB	If 1, supports fast short REP CMPSB, REP SCASB.	Platform
EAX[18:13]	Reserved	Reserved.	
EAX[19]	WRMSRNS	If 1, supports the WRMSRNS instruction.	Platform
EAX[20]	Reserved	Reserved.	
EAX[21]	AMX_FP16	If 1, supports tile computational operations on FP16 numbers.	Platform
EAX[22]	HRESET	If 1, supports history reset via the HRESET instruction and the IA32_HRESET_ENABLE MSR. When set, indicates that the Processor History Reset Leaf (EAX = 20H) is valid.	Platform
EAX[23]	AVX_IFMA	If 1, supports the AVX-IFMA instructions.	Platform
EAX[25:24]	Reserved	Reserved.	
EAX[26]	LAM	If 1, supports Linear Address Masking.	Platform
EAX[27]	MSRLIST	If 1, supports the RDMSRLIST and WRMSRLIST instructions and the IA32_BARRIER MSRR.	Platform
EAX[29:28]	Reserved	Reserved.	
EAX[30]	INVD_DISABLE_POST_BIOS_DONE	If 1, supports INVD execution prevention after BIOS Done.	Platform
EAX[31]	Reserved	Reserved.	

CPUID.07H.01H:EBX Extended Feature Information

The EBX register of CPUID.07H.01H returns the information shown below.

Table 21-23. Leaf 07H.01H Structured Extended Feature Flags Returned in EBX

Register	Field Name	Description	Domain
EBX[0]	PPIN	If 1, supports the IA32_PPIN and IA32_PPIN_CTL MSRs.	Platform
EBX[2:1]	Reserved	Reserved.	
EBX[3]	CPUIDMAXVAL_LIM_RMV	If 1, IA32_MISC_ENABLE[bit 22] cannot be set to 1 to limit the value returned by CPUID.00H:EAX[bits 7:0].	Platform
EBX[31:4]	Reserved	Reserved.	

CPUID.07H.01H:ECX Extended Feature Information

The ECX register of CPUID.07H.01H returns the information shown below.

Table 21-24. Leaf 07H.01H Structured Extended Feature Flags Returned in ECX

Register	Field Name	Description	Domain
ECX[31:0]	Reserved	Reserved.	

CPUID.07H.01H:EDX Extended Feature Information

The EDX register of CPUID.07H.01H returns the information shown below.

Table 21-25. Leaf 07H.01H Structured Extended Feature Flags Returned in EDX

Register	Field Name	Description	Domain
EDX[3:0]	Reserved	Reserved.	
EDX[4]	AVX_VNNI_INT8	If 1, supports the AVX-VNNI-INT8 instructions.	Platform
EDX[5]	AVX_NE_CONVERT	If 1, supports the AVX-NE-CONVERT instructions.	Platform
EDX[9:6]	Reserved	Reserved.	
EDX[10]	AVX_VNNI_INT16	If 1, supports the AVX-VNNI-INT16 instructions.	Platform
EDX[13:11]	Reserved	Reserved.	
EDX[14]	PREFETCHI	If 1, supports the PREFETCHIT0/1 instructions.	Platform
EDX[16:15]	Reserved	Reserved.	
EDX[17]	UIRET_UIF	If 1, UIRET sets UIF to the value of bit 1 of the RFLAGS image loaded from the stack.	Platform

EDX[18]	CET_SSS	If 1, indicates that an operating system can enable supervisor shadow stacks as long as it ensures that a supervisor shadow stack cannot become prematurely busy due to page faults (see Section 17.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). When emulating the CPUID instruction, a virtual-machine monitor (VMM) should return this bit as 1 only if it ensures that VM exits cannot cause a guest supervisor shadow stack to appear to be prematurely busy. Such a VMM could set the "prematurely busy shadow stack" VM-exit control and use the additional information that it provides.	Platform
EDX[19]	AVX10	If 1, supports the Intel® AVX10 instructions and indicates the presence of CPUID.24H, which enumerates version number and supported vector lengths.	Platform
EDX[23:20]	Reserved	Reserved.	
EDX[24]	SLSM	Static LSM is supported on this platform. If set, IA32_INTEGRITY_STATUS (0x2DC) is available for software use.	Platform
EDX[31:25]	Reserved	Reserved.	

CPUID.07H.02H -- Structured Extended Feature Sub-Leaf 2

CPUID.07H.02H returns the structured extended feature information contained in the sub-sections of this section.

Table 21-26. Leaf 07H Sub-Leaf (ECX=2) Output Registers

CPUID Output Registers	Description
EAX[31:0]	Reserved
EBX[31:0]	Reserved
ECX[31:0]	Reserved
EDX[31:0]	Extended Feature Information (see "CPUID.07H.02H:EDX—Extended Feature Information")

CPUID.07H.02H:EDX Extended Feature Information

The EDX register of CPUID.07H.01H returns the information shown below.

Table 21-27. CPUID.07H.02H Extended Feature Information Provided in EDX¹

Register	Field Name	Description	Domain
EDX[0]	PSFD	If 1, supports bit 7 of the IA32_SPEC_CTRL MSR. Bit 7 of this MSR disables Fast Store Forwarding Predictor without disabling Speculative Store Bypass.	Platform
EDX[1]	IPRED_CTRL	If 1, supports bits 3 and 4 of the IA32_SPEC_CTRL MSR. Bit 3 of this MSR enables IPRED_DIS control for CPL3. Bit 4 of this MSR enables IPRED_DIS control for CPL0/1/2.	Platform

EDX[2]	RRSBA_CTRL	If 1, supports bits 5 and 6 of the IA32_SPEC_CTRL MSR. Bit 5 of this MSR disables RRSBA behavior for CPL3. Bit 6 of this MSR disables RRSBA behavior for CPL0/1/2.	Platform
EDX[3]	DDPD_U	If 1, supports bit 8 of the IA32_SPEC_CTRL MSR. Bit 8 of this MSR disables Data Dependent Prefetcher.	Platform
EDX[4]	BHI_CTRL	If 1, supports bit 10 of the IA32_SPEC_CTRL MSR. Bit 10 of this MSR enables BHI_DIS_S behavior.	Platform
EDX[5]	MCDT_NO	If 1, the processor does not exhibit MXCSR Configuration Dependent Timing (MCDT) behavior and does not need to be mitigated to avoid data- dependent behavior for certain instructions.	Platform
EDX[6]	UC_LOCK_DISABLE	If 1, supports the UC-lock disable feature and it causes #AC.	Platform
EDX[7]	MONITOR_MITG_NO	If 1, the MONITOR/UMONITOR instructions are not affected by performance or power issues due to MONITOR/UMONITOR instructions exceeding the capacity of an internal monitor tracking table. If 0, then the product may be affected by this issue.	Platform
EDX[31:8]	Reserved	Reserved.	

NOTE

1. Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub-leaf index, EDX returns 0.

CPUID.08H -- Reserved

This leaf is reserved.

Table 21-28. Leaf 08H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.09H -- Direct Cache Access Information

- CPUID.09H returns information about Direct Cache Access capabilities.
- This leaf is valid if CPUID.01H:ECX.DCA[18] = 1 and MAX_LEAF ≥ 09H.
 - This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-29. Leaf 09H Direct Cache Access Information

Register	Field Name	Description	Domain
EAX[31:0]	PLATFORM_DCA_CAP	Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

The IA32_PLATFORM_DCA_CAP MSR is valid when CPUID.01H:ECX.DCA[18] = 1.

CPUID.0AH -- Architectural Performance Monitoring

CPUID.0AH returns information about support for architectural performance monitoring capabilities.

- This leaf is valid if CPUID.0AH:EAX[7:0] (Version ID) > 0 and MAX_LEAF ≥ 09H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-30. Leaf 0AH Architectural Performance Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	VERSION	Version ID of architectural performance monitoring.	Platform
EAX[15:8]	NUM_GP_CTRS	Number of general-purpose performance monitoring counter(s) per logical processor.	Platform
EAX[23:16]	GP_CTR_WIDTH	Bit width of general-purpose, performance monitoring counter.	Platform
EAX[31:24]	EVENT_ENUM_LENGTH	Length of EBX bit vector to enumerate architectural performance monitoring events. Architectural event x is supported if EBX[x]=0 && EAX[31:24]>x.	Platform
EBX[0]	CORE_CYC_NA	Core cycle event not available if 1 or if EAX[31:24]<1.	Platform
EBX[1]	INTR_RET_NA	Instruction retired event not available if 1 or if EAX[31:24]<2.	Platform
EBX[2]	REF_CYC_NA	Reference cycles event not available if 1 or if EAX[31:24]<3.	Platform
EBX[3]	LLC_CYC_NA	Last-level cache reference event not available if 1 or if EAX[31:24]<4.	Platform
EBX[4]	LLC_MISSES_NA	Last-level cache misses event not available if 1 or if EAX[31:24]<5.	Platform
EBX[5]	BR_INSTR_RET_NA	Branch instruction retired event not available if 1 or if EAX[31:24]<6.	Platform
EBX[6]	BR_MISPRED_RET_NA	Branch mispredict retired event not available if 1 or if EAX[31:24]<7.	Platform
EBX[7]	SLOTS_NA	Top-down slots event not available if 1 or if EAX[31:24]<8.	Platform
EBX[9:8]	Reserved	Reserved.	
EBX[10]	FRONTEND_NA	Topdown frontend bound not available if 1 or if EAX[31:24] < 11.	Platform
EBX[11]	RETIRING_NA	Topdown retiring not available if 1 or if EAX[31:24] < 12.	Platform
EBX[12]	LBR_INSERTS_NA	LBR inserts not available if 1 or if EAX[31:24] < 13.	Platform
EBX[31:13]	Reserved	Reserved.	
ECX[31:0]	FIXED_CTR_MASK	Supported fixed counters bit mask. Fixed-function performance counter 'i' is supported if bit 'i' is 1 (first counter index starts at zero). It is recommended to use the following logic to determine if a Fixed Counter is supported: FxCtr[i]_is_supported := ECX[i] (EDX[4:0] > i);	Platform

EDX[4:0]	NUM_FIXED_CTR	Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1).	Platform
EDX[12:5]	FIXED_CTR_WIDTH	Bit width of fixed-function performance counters (if Version ID > 1).	Platform
EDX[14:13]	Reserved	Reserved.	
EDX[15]	ANYTHREAD_DEPRECATION	Starting with Architectural Performance Monitoring Version 5, this field indicates that a processor supports AnyThread mode deprecation. If this field is set, software can choose to ignore guidelines in “AnyThread Counting and Software Evolution” of Chapter 21, “Performance Monitoring,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B	Platform
EDX[19:16]	SLOTS_PER_CYC	If this field is non-zero, it represents the number of Top-down Microarchitecture Analysis (TMA) slots per cycle. This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of slots. If this field is zero, IA32_FIXED_CTR3 should be used to determine the total number of slots.	Platform
EDX[31:20]	Reserved	Reserved.	

For each version of architectural performance monitoring capability, software must enumerate this leaf to discover the programming facilities and the architectural performance events available in the processor. The details are described in Chapter 25, “Introduction to Virtual Machine Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C.

CPUID.0BH -- Extended Topology

CPUID.0BH returns information about Extended Topology. CPUID.1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of leaf 1FH before using leaf 0BH.

- This leaf is valid if CPUID.0BH.00H:EBX[15:0] $\neq$ 0 and MAX_LEAF $\geq$ 0BH.
- The sub-leaves are enumerated until sub-leaf n returns 0 in EBX[15:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EBX[15:0] as 0.

CPUID.0BH -- ECX $\geq$ 0

Table 21-31. Leaf 0BH Extended Topology

Register	Field Name	Description	Domain
EAX[4:0]	SHIFT_COUNT	The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[15:0]	NEXT_LEVEL_NUM_LP	The number of logical processors across all instances of this domain within the next higher-scoped domain. (For example, in a processor socket/package comprising “M” cores of “N” logical processors each, the “core” domain sub-leaf value of this field would be M*N.) This number reflects configuration as shipped by Intel. This field may also contain asymmetric values across different logical processors, as an example of a mix of cores that support more than one logical processor with cores that support only one logical processor. Note, software must not use this field to enumerate processor topology. Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[7:0]	LEVEL_NUM	The input ECX sub-leaf index.	Platform
ECX[15:8]	DOMAIN_TYPE	This field provides an identification value which indicates the domain shown in the table. Although domains are ordered, their assigned identification values are not and software should not depend on it. Note that enumeration values of 0 and 3-255 are reserved.	Platform
ECX[31:16]	Reserved	Reserved.	

EDX[31:0]	X2APIC_ID	The X2APIC ID of this logical processor.	Logical Processor

The sub-leaves of CPUID.0BH describe an ordered hierarchy of logical processors starting from the smallestscoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largestscoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket). The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated. For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor. If sub-leaf index “N” returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than “N” shall also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.

Table 21-32. Hierarchy of Valid Domain Enumerations in CPUID.0BH:ECX[15:8]

Hierarchy	Domain	Domain Type ID Value
Invalid	Invalid	0
Lowest	Logical Processor	1
...	Core	2
Highest	Package/Socket	(Implied)
Reserved	Reserved	3-255

CPUID.0CH -- Reserved

Table 21-33. Leaf 0CH Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0DH -- Processor Extended State

CPUID.0DH returns a bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area.

- This leaf is valid if CPUID.01H:ECX.XSAVE[26] = 1 and MAX_LEAF ≥ 0DH.
- Sub-leaves 0 and 1 are always valid; consult them to determine which other sub-leaves are present as described in “CPUID.0DH.n, n>01H—State Sub-Leaves”.

CPUID.0DH.00H -- Processor Extended State Main Sub-Leaf

CPUID.0DH.00H returns the processor extended state information.

Table 21-34. Leaf 0DH.00H Processor Extended State

Register	Field Name	Description	Domain
EAX[0]	X87	x87 state.	Platform
EAX[1]	SSE	SSE state.	Platform
EAX[2]	AVX	AVX state.	Platform
EAX[3]	MPX_BNDREGS	MPX state.	Platform
EAX[4]	MPX_BNDCSR	MPX state.	Platform
EAX[5]	AVX512_OPMASK	AVX-512 Opmask state.	Platform
EAX[6]	AVX512_ZMM_HI256	AVX-512 ZMM upper 256 data state.	Platform
EAX[7]	AVX512_HI16_ZMM	AVX-512 upper 16 ZMM registers state.	Platform
EAX[8]	PT	PT state.	Platform
EAX[9]	PKRU	PKRU state.	Platform
EAX[10]	PASID	PASID state.	Platform
EAX[11]	CET_U	CET user state.	Platform
EAX[12]	CET_S	CET supervisor state.	Platform
EAX[13]	HDC	HDC state.	Platform
EAX[14]	UINTR	UINTR state.	Platform
EAX[15]	LBR	LBR state (only for the architectural LBR feature).	Platform
EAX[16]	HWP	HWP state.	Platform
EAX[17]	AMX_TILECFG	TILECFG state.	Platform
EAX[18]	AMX_TILEDATA	TILEDATA state.	Platform
EAX[31:19]	Reserved	Reserved.	
EBX[31:0]	XSAVE_BYTES_ENABLED_FEATURES	Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) required by enabled features in XCR0. May be different than ECX if some features at the end of the XSAVE save area are not enabled.	Logical Processor
ECX[31:0]	XSAVE_BYTES_SUPPORTED_FEATURES	Maximum size (bytes, from the beginning of the XSAVE/XRSTOR save area) of the XSAVE/XRSTOR save area required by all supported features in the processors, i.e. all the valid bit fields in XCR0.	Platform

EDX[31:0]	VALID_XCRO_UPPER_32	Reports the supported bits of the upper 32 bits of XCRO. XCRO[n+32] can be set to 1 only if EDX[n] is 1.	Platform

The EAX register of CPUID.0DH.00H reports the supported bits of the lower 32 bits of XCRO. XCRO[n] can be set to 1 only if EAX[n] is 1.

CPUID.0DH.01H -- Feature and Supervisor State Sub-Leaf

CPUID.0DH.01H returns feature and supervisor state information.

Table 21-35. Leaf 0DH.01H Processor Extended State

Register	Field Name	Description	Domain
EAX[0]	XSAVEOPT	If 1, supports XSAVEOPT.	Platform
EAX[1]	XSAVEC	If 1, supports XSAVEC and the compacted form of XRSTOR.	Platform
EAX[2]	XGETBV1	If 1, supports XGETBV with ECX = 1.	Platform
EAX[3]	XSAVES	If 1, supports XSAVES/XRSTORS and IA32_XSS.	Platform
EAX[4]	XFD	If 1, supports extended feature disable (XFD).	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	XSAVES_BYTES_ENABLED_FEATURES	The size in bytes of the XSAVE area containing all states enabled by XCRO IA32_XSS. If EAX[3] is enumerated as 0 and EAX[1] is enumerated as 1, EBX enumerates the size of the XSAVE area containing all states enabled by XCRO. If EAX[1] and EAX[3] are both enumerated as 0, EBX enumerates zero.	Logical Processor
ECX[0]	X87	x87 state.	Platform
ECX[1]	SSE	SSE state.	Platform
ECX[2]	AVX	AVX state.	Platform
ECX[3]	MPX_BNDREGS	MPX state.	Platform
ECX[4]	MPX_BNDCSR	MPX state.	Platform
ECX[5]	AVX512_OPMASK	AVX-512 Opmask state.	Platform
ECX[6]	AVX512_ZMM_HI256	AVX-512 ZMM upper 256 data state.	Platform
ECX[7]	AVX512_HI16_ZMM	AVX-512 upper 16 ZMM registers state.	Platform
ECX[8]	PT	PT state.	Platform
ECX[9]	PKRU	PKRU state.	Platform
ECX[10]	PASID	PASID state.	Platform
ECX[11]	CET_U	CET user state.	Platform
ECX[12]	CET_S	CET supervisor state.	Platform
ECX[13]	HDC	HDC state.	Platform
ECX[14]	UINTR	UINTR state.	Platform
ECX[15]	LBR	LBR state (only for the architectural LBR feature).	Platform
ECX[16]	HWP	HWP state.	Platform

ECX[17]	TILECFG	TILECFG state.	Platform
ECX[18]	TILEDATA	TILEDATA state.	Platform
ECX[31:19]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved	

NOTE

ECX reports the supported bits of the lower 32 bits of the IA32_XSS MSR. IA32_XSS[n] can be set to 1 only if ECX[n] is 1. EDX reports the supported bits of the upper 32 bits of the IA32_XSS MSR. IA32_XSS[n+32] can be set to 1 only if EDX[n] is 1.

CPUID.0DH.ECX > 1 -- ECX > 1

CPUID.0DH.n, where n > 1, returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. Software can use the forward-extendable technique depicted below to query in bulk the valid sub-leaves and obtain size and offset information for each processor extended state save area that is supported:

```

/* For each supported feature indicated by sub-leaf 0 and 1, read the size and offset sub-leaf */
For j= 2 to 62
  If (CPUID.0DH.00H: <EDX:EAX>[j] == 1 or // Use 64-bit value of EDX:EAX
      CPUID.0DH.01H: <EDX:ECX>[j] == 1) // Use 64-bit value of EDX:ECX
    Read(CPUID.0DH.j) // Examine the size and offset.
  END IF
END FOR

```

Table 21-36. Leaf 0DH.ECX > 1 Processor Extended State

Register	Field Name	Description	Domain
EAX[31:0]	COMP_SIZE	The size in bytes (from the offset specified in EBX) of the save area for an extended state feature associated with a valid sub-leaf index n.	Platform
EBX[31:0]	COMP_OFFSET	The offset in bytes of this extended state component's save area from the beginning of the XSAVE/XRSTOR area. This field reports 0 if the sub-leaf index, n, does not map to a valid bit in the XCRO register. If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n ($0 \leq n \leq 31$) is invalid if sub-leaf 0 returns 0 in EAX[n] and sub-leaf 1 returns 0 in ECX[n]. Sub-leaf n ($32 \leq n \leq 63$) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].	Platform
ECX[0]	COMP_SUP	This bit is set if the bit n (corresponding to the sub-leaf index) is supported in the IA32_XSS MSR; it is clear if bit n is instead supported in XCRO.	Platform
ECX[1]	COMP_64B_ALIGNED	This bit is set if, when the compacted format of an XSAVE area is used, this extended state component located on the next 64-byte boundary following the preceding state component (otherwise, it is located immediately following the preceding state component)	Platform
ECX[2]	COMP_XFD	is set to indicate support for XFD faulting.	Platform

ECX[31:3]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0EH -- Reserved

This leaf is reserved.

Table 21-37. Leaf 0EH Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.0FH -- Intel® Resource Director Technology (Intel® RDT) Monitoring

CPUID.0FH returns information for the Intel Resource Director Technology Monitoring capabilities. As described below, software uses the bit vector returned in EDX by sub-leaf 00H to determine the available resource types (ResID) that can be monitored. This information is necessary for software to program the IA32_PQR_ASSOC and IA32_QM_EVTSEL MSRs such that Quality-of-Service data can be read afterwards from the IA32_QM_CTR MSR.

- This leaf is valid if CPUID.07H.00H:EBX.RDT_M[12] = 1 and MAX_LEAF ≥ 0FH.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.0FH.00H:EDX[n] == 1).

CPUID.0FH.00H -- Intel® RDT Monitoring Main Sub-Leaf

CPUID.0FH.00H returns information about Intel RDT Monitoring.

Table 21-38. Leaf 0FH.00H Intel® Resource Director Technology (Intel® RDT) Monitoring

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	MAX_RMID	Maximum range (zero-based) of RMID within this physical processor of all types.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[0]	Reserved	Reserved.	
EDX[1]	CMT_L3	If 1, supports L3 Cache Intel RDT Monitoring. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.	Platform
EDX[31:2]	Reserved	Reserved.	

CPUID.0FH.00H returns information about the bit-vector representation of QoS monitoring resource types that are supported in the processor and maximum range of RMID values the processor can use to monitor of any supported resource types. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS monitoring capability available for that type.

CPUID.0FH.01H -- L3 Cache Intel® Resource Director Technology Monitoring

CPUID.0FH.01H returns information about L3 Cache Intel RDT monitoring.

Table 21-39. Leaf 0FH.01H Intel® Resource Director Technology (Intel® RDT) Monitoring

Register	Field Name	Description	Domain
EAX[7:0]	CTR_WIDTH	The counter width is encoded as an offset from 24b. A value of zero in this field indicates that 24-bit counters are supported. A value of 8 in this field indicates that 32-bit counters are supported.	Platform
EAX[8]	RDT_M_OVF	If 1, supports an overflow bit in the IA32_QM_CTR MSR (bit 61).	Platform
EAX[9]	IO_QOS_CMT	If 1, indicates the presence of non-CPU agent supporting Intel RDT CMT.	Platform
EAX[10]	IO_QOS_MBM	If 1, indicates the presence of non-CPU agent supporting Intel RDT MBM support.	Platform

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

EAX[31:11]	Reserved	Reserved.	
EBX[31:0]	CONV_FACTOR	Factor used to convert from reported IA32_QM_CTR value to derived occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.	Platform
ECX[31:0]	MAX_RMID_L3	Maximum range (zero-based) of RMID of this resource type.	Platform
EDX[0]	CMT_L3_OCCUP	If 1, supports L3 occupancy monitoring.	Platform
EDX[1]	CMT_L3_TOTAL	If 1, supports L3 total bandwidth monitoring.	Platform
EDX[2]	CMT_L3_LOCAL	If 1, supports L3 local bandwidth monitoring.	Platform
EDX[31:3]	Reserved	Reserved.	

CPUID.10H -- Intel® Resource Director Technology (Intel® RDT) Allocation

CPUID.10H returns information for Intel Resource Director Technology Allocation. This leaf is valid when CPUID.07H.00H:EBX.RDT_A[15] = 1. As described below, software uses the bit vector returned in EBX by subleaf 00H to determine the available QoS Enforcement (allocation) resource types that are supported in the processor. This information is necessary for software to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

- This leaf is valid if CPUID.07H.00H:EBX.RDT_A[15] = 1 and MAX_LEAF ≥ 10H.
- If the leaf is valid, sub-leaf 00H is always valid. Sub-leaf n (n ≥ 1) is only valid when (CPUID.10H.00H:EBX[n] == 1).

CPUID.10H.00H -- Intel® RDT Allocation Main Sub-Leaf

CPUID.10H.00H returns information about Intel RDT Allocation.

Table 21-40. Leaf 10H.00H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	Reserved	Reserved.	
EBX[1]	CAT_L3	If 1, supports L3 Cache Allocation Technology. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX.	Platform
EBX[2]	CAT_L2	If 1, supports L2 Cache Allocation Technology.	Platform
EBX[3]	MBA	If 1, supports Memory Bandwidth Allocation.	Platform
EBX[31:4]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.10H.00H returns information about the bit-vector representation of QoS Enforcement resource types that are supported in the processor. Each bit, starting from bit 1, corresponds to a specific resource type if the bit is set. The bit position corresponds to the sub-leaf index (or ResID) that software must use to query QoS enforcement capability available for that type.

CPUID.10H.01H -- L3 Cache Allocation Technology

CPUID.10H.ResID=1 returns information about L3 Cache Allocation Technology.

Table 21-41. Leaf 10H.01H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L3_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L3_MAP	Bit-granular map of isolation/contention of allocation units.	Platform
ECX[0]	Reserved	Reserved.	
ECX[1]	CAT_L3_NONCPU	If 1, supports L3 CAT for non-CPU agents.	Platform

ECX[2]	CAT_L3_CDP	If 1, supports L3 Code and Data Prioritization Technology.	Platform
ECX[3]	CAT_L3_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L3_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.02H -- L2 Cache Allocation Technology

CPUID.10H.ResID=2 returns information about L2 Cache Allocation Technology.

Table 21-42. Leaf 10H.02H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[4:0]	CAT_L2_BITMASK_LENGTH	Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[31:0]	CAT_L2_MAP	Bit-granular map of isolation/contention of allocation units.	Platform
ECX[1:0]	Reserved	Reserved.	
ECX[2]	CAT_L2_CDP	If 1, supports L2 Code and Data Prioritization Technology.	Platform
ECX[3]	CAT_L2_NONCONTIG	If 1, supports non-contiguous capacity bitmasks. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous.	Platform
ECX[31:4]	Reserved	Reserved.	
EDX[15:0]	CAT_L2_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.10H.03H -- Memory Bandwidth Allocation

CPUID.10H.ResID=3 returns information aboutt Memory Bandwidth Allocation.

Table 21-43. Leaf 10H.03H Intel® Resource Director Technology (Intel® RDT) Allocation

Register	Field Name	Description	Domain
EAX[11:0]	MBA_MAX	Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result.	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[0]	PER_THREAD_MBA	Per-thread MBA controls are supported.	Platform

ECX[1]	Reserved	Reserved.	
ECX[2]	MBA_LINEAR	If 1, the response of the delay values is linear.	Platform
ECX[31:3]	Reserved	Reserved.	
EDX[15:0]	MBA_MAX_CLOS	Highest Class of Service (COS) number supported for this ResID.	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.11H -- Reserved

This leaf is reserved.

Table 21-45. Leaf 11H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.12H -- Intel® Software Guard Extensions (Intel® SGX) Capability

CPUID.12H returns information about Intel® SGX capabilities. More details can be found in Chapter 35, “Introduction to Intel® Software Guard Extensions,” and Chapter 36, “Enclave Access Control and Data Structures,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

- This leaf is valid when CPUID.07H.00H:EBX.SGX[2] = 1 and MAX_LEAF ≥ 12H.
- If the leaf is valid, sub-leaf 00H and 01H are always valid. Sub-leaf n (n ≥ 2) is only valid when CPUID.12H.n:EAX[3:0] != 0.

CPUID.12H.00H -- Intel® SGX Main Sub-Leaf

CPUID.12H.00H returns information about Intel® SGX capabilities. It is only valid when CPUID.07H.00H:EBX.SGX = 1.

Table 21-46. Leaf 12H.00H Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[0]	SGX1	If 1, supports the collection of SGX1 Leaf functions.	Platform
EAX[1]	SGX2	If 1, supports the collection of SGX2 Leaf functions.	Platform
EAX[5:2]	Reserved	Reserved.	
EAX[6]	Reserved	Reserved.	
EAX[7]	EVERIFYREPORT2	If 1, supports the ENCLU instruction Leaf EVERIFYREPORT2.	Platform
EAX[9:8]	Reserved	Reserved.	
EAX[10]	EUPDATESVN	If 1, supports the ENCLS instruction Leaf EUPDATESVN.	Platform
EAX[11]	EDECCSSA	If 1, supports the ENCLU instruction Leaf EDECCSSA.	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	MISCSELECT	Bit vector of supported extended SGX features. The definition of MISCSELECT can be found in Section 36.7.2, “SECS.MISCSELECT Field,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[7:0]	MAX_ENCLAVE_SIZE_NOT_64	The maximum supported enclave size in non-64-bit mode is 2 ^(EDX[7:0]) .	Platform
EDX[15:8]	MAX_ENCLAVE_SIZE_64	The maximum supported enclave size in 64-bit mode is 2 ^(EDX[15:8]) .	Platform
EDX[31:16]	Reserved	Reserved.	

CPUID.12H.01H -- Intel® SGX Attributes

CPUID.12H.01H returns information about Intel® SGX Attributes. It is only valid when CPUID.07H.00H:EBX.SGX = 1.

Table 21-47. Leaf 12H.01H Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain

EAX[31:0]	ECREATE_SECS_ATTRIBUTES_31_0	Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.	Platform
EBX[31:0]	ECREATE_SECS_ATTRIBUTES_63_32	Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.	Platform
ECX[31:0]	ECREATE_SECS_ATTRIBUTES_95_64	Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.	Platform
EDX[31:0]	ECREATE_SECS_ATTRIBUTES_127_96	Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.	Platform

The definition of the attributes can be found in Section 36.7.1, "ATTRIBUTES," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D

CPUID.12H.N≥2 - INTEL® SGX ENCLAVE PAGE CACHE -- n≥2 - Intel® SGX Enclave Page Cache

CPUID.12H with ECX≥2 returns information about Intel® SGX Enclave Page Cache and is supported if CPUID.07H.00H:EBX.SGX = 1.

For sub-leaves where ECX≥2, the definition of EAX[31:4], EBX, ECX, and EDX depends on the sub-leaf type listed below.

Table 21-48. Leaf 12H.N≥2 - INTEL® SGX ENCLAVE PAGE CACHE Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[3:0]	SUB_LEAF_TYPE	0000b: Indicates this sub-leaf is invalid. - EDX:ECX:EBX:EAX return 0. 0001b: This sub-leaf enumerates an EPC section. - EBX:EAX and EDX:ECX provide information on the Enclave Page Cache (EPC) section. All other type encodings are reserved.	Platform
EAX[31:4]	Reserved	Output dependent on EAX[3:0] subtype.	
EBX[31:0]	Reserved	Output dependent on EAX[3:0] subtype.	
ECX[31:0]	Reserved	Output dependent on EAX[3:0] subtype.	
EDX[31:0]	Reserved	Output dependent on EAX[3:0] subtype.	

Sub-Leaf Encoding Type EAX[3:0] = 0000b (Invalid)

This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.

CPUID.12H -- Sub-Leaf Encoding Type EAX[3:0] = 0001b

This sub-leaf enumerates an EPC section with EDX:ECX, EBX:EAX defined as follows.

Table 21-49. Leaf 12H.SUB-LEAF ENCODING TYPE EAX[3:0] = 0001B Intel® Software Guard Extensions (Intel® SGX) Capability

Register	Field Name	Description	Domain
EAX[3:0]	SUB_LEAF_TYPE	Value is 0001b.	Platform
EAX[11:4]	Reserved	Reserved.	
EAX[31:12]	EPC_SECTION_ADDR_31_12	Bits 31:12 of the physical address of the base of the EPC section.	Platform
EBX[19:0]	EPC_SECTION_ADDR_51_32	Bits 51:32 of the physical address of the base of the EPC section.	Platform
EBX[31:20]	Reserved	Reserved.	
ECX[3:0]	EPC_SECTION_PROPERTY	EPC Section Property Encoding Definitions, as follows: 0000b - All bits in EDX:ECX are enumerated as 0. 0001b - This section has confidentiality, integrity, and replay protection. 0010b - This section has confidentiality protection only. 0011b - This section has confidentiality and integrity protection. All other encodings are reserved.	Platform
ECX[11:4]	Reserved	Reserved.	
ECX[31:12]	EPC_SECTION_SIZE_31_12	Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory.	Platform
EDX[19:0]	EPC_SECTION_SIZE_51_32	Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory.	Platform
EDX[31:20]	Reserved	Reserved.	

CPUID.13H -- Reserved

This leaf is reserved.

Table 21-50. Leaf 13H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.14H -- Intel® Processor Trace (Intel® PT)

CPUID.14H returns information about Intel® Processor Trace (PT).

CPUID.14H.00H returns information about Intel Processor Trace extensions.

CPUID.14H.n (n > 0 and less than the number of non-zero bits in CPUID.14H.00H:EAX) returns information about packet generation in Intel Processor Trace.

For more details on Intel PT, see Chapter 34, “Intel® Processor Trace,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3D.

- This leaf is valid when CPUID.07H.00H:EBX.INTEL_PROC_TRACE[25] = 1 and MAX_LEAF ≥ 14H.
- The maximum sub-leaf value for ECX is specified in CPUID.14H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.14H.00H -- Intel® PT Main Sub-Leaf

CPUID.14H.00H returns information about Intel Processor Trace extensions.

Table 21-51. Leaf 14H.00H Intel® Processor Trace (Intel® PT)

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum sub-leaf supported in leaf 14H.	Platform
EBX[0]	CR3_FILTER	If 1, supports that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed.	Platform
EBX[1]	CYC_ACC	If 1, supports Configurable PSB and Cycle-Accurate Mode.	Platform
EBX[2]	IP_FILTER	If 1, supports IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset.	Platform
EBX[3]	MTC	If 1, supports MTC timing packet and suppression of COFI-based packets.	Platform
EBX[4]	PTWRITE	If 1, supports PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets.	Platform
EBX[5]	PWR_EVT_TRACE	If 1, supports Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation.	Platform
EBX[6]	PMI_PRESERVE	If 1, supports the PSB and PMI preservation. Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEn-able), enabling the processor to set IA32_RTIT_STATUS[7] (PendToPaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB.	Platform
EBX[7]	EVENT_TRACE	If 1, supports that writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation.	Platform
EBX[8]	TNT_DIS	If 1, supports that writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation.	Platform

EBX[31:9]	Reserved	Reserved.	
ECX[0]	TOPAOUT	If 1, supports that tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUPUT_MASK_PTRS MSRs can be accessed.	Platform
ECX[1]	MENTRY	If 1, supports that ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOffsetTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS.	Platform
ECX[2]	SNGL_RNG_OUT	If 1, supports the Single-Range Output scheme.	Platform
ECX[3]	TRACE_TRANSPORT_SUBSYSTEM	If 1, indicates support of output to Trace Transport Subsystem.	
ECX[30:4]	Reserved	Reserved.	
ECX[31]	LIP	If 1, the generated packets which contain IP payloads values contain LIP. If 0, the generated packets which contain IP payloads contain Effective IP. Trace segments using a flat memory model will generate the same information regardless of how a logical processor reports this value since LIP=EIP.	Logical Processor
EDX[31:0]	Reserved	Reserved.	

CPUID.14H.01H -- Feature Information Sub-Leaf

CPUID.14H.01H returns information about packet generation in Intel Processor Trace.

Table 21-51. Leaf 14H.01H Intel® Processor Trace (Intel® PT)

Register	Field Name	Description	Domain
EAX[2:0]	RANGECNT	Number of configurable Address Ranges for filtering.	Platform
EAX[15:3]	Reserved	Reserved.	
EAX[31:16]	MTC_RATE	Bitmap of supported MTC period encodings.	Platform
EBX[15:0]	CYC_THRESHOLDS	Bitmap of supported Cycle Threshold value encodings.	Platform
EBX[31:16]	PSB_RATE	Bitmap of supported Configurable PSB frequency encodings.	Platform
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.15H -- Time Stamp Counter and Nominal Core Crystal Clock

CPUID.15H returns information about the Time Stamp Counter and the Nominal Core Crystal Clock.

- This leaf is valid if MAX_LEAF $\geq$ 15H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-53. Leaf 15H Time Stamp Counter and Nominal Core Crystal Clock

Register	Field Name	Description	Domain
EAX[31:0]	DENOMINATOR	An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.	Platform
EBX[31:0]	NUMERATOR	An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio. If 0, the TSC/"core crystal clock" ratio is not enumerated.	Platform
ECX[31:0]	NOMINAL_ART_FREQUENCY	An unsigned integer which is the nominal frequency of the core crystal clock in Hz. If 0, the nominal core crystal clock frequency is not enumerated. Note, the core crystal clock may differ from the reference clock, bus clock or core clock frequencies.	Platform
EDX[31:0]	Reserved	Reserved.	

Dividing EBX[31:0] by EAX[31:0] provides the ratio of the TSC frequency to the core crystal clock frequency. The Timestamp Counter frequency is computed by multiplying that ratio by ECX[31:0], such as $TSC\_frequency = ECX * EBX/EAX$.

CPUID.16H -- Processor Frequency Information

CPUID.16H returns information about processor frequency information. Data is returned from this interface in accordance with the processor's specification and does not reflect actual values. Suitable use of this data includes the display of processor information in like manner to the processor brand string and for determining the appropriate range to use when displaying processor information e.g. frequency history graphs. The returned information should not be used for any other purpose as the returned information does not accurately correlate to information / counters returned by other processor interfaces. While a processor may support the Processor Frequency Information leaf, fields that return a value of zero are not supported.

- This leaf is valid if MAX_LEAF ≥ 16H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-54. Leaf 16H Processor Frequency Information

Register	Field Name	Description	Domain
EAX[15:0]	PROCESSOR_BASE_FREQUENCY	Processor Base Frequency (in MHz).	Logical Processor
EAX[31:16]	Reserved	Reserved.	
EBX[15:0]	MAXIMUM_FREQUENCY	Maximum Frequency (in MHz).	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[15:0]	BUS_FREQUENCY	Bus (Reference) Frequency (in MHz).	Logical Processor
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.17H -- System-on-Chip Vendor Attribute

CPUID.17H returns System-on-Chip vendor attribute information.

- This leaf is valid if CPUID.17H.00H:EAX[31:0] (MaxSOCID_Index) ≥ 3 and MAX_LEAF ≥ 17 H.
- The maximum sub-leaf value for ECX is specified in CPUID.17H.00H:EAX[31:0] MaxSOCID_Index.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.17H.00H -- Main Sub-Leaf

CPUID.17H.00H returns System-on-Chip vendor attribute information.

Table 21-55. Leaf 17H.00H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SOCID_INDEX	Reports the maximum input value of supported Sub-leaf in Leaf 17H.	Platform
EBX[15:0]	SOC_VENDOR_ID	SOC Vendor ID.	Platform
EBX[16]	IS_VENDOR_SCHEME	If 1, the SOC Vendor ID field is assigned via an industry standard enumeration scheme. Otherwise, the SOC Vendor ID field is assigned by Intel.	Platform
EBX[31:17]	Reserved	Reserved.	
ECX[31:0]	PROJECT_ID	A unique number an SOC vendor assigns to its SOC projects.	Platform
EDX[31:0]	STEPPING_ID	A unique number within an SOC project that an SOC vendor assigns.	Package

CPUID.17H.01H -- Vendor Brand String Sub-Leaf (Bytes 0 to 15)

CPUID.17H with ECX=1, 2, or 3 returns information for System-on-Chip vendor branding string.

SOC Vendor Brand String is a UTF-8 encoded string padded with trailing bytes of 00H. The complete SOC Vendor Brand String is constructed by concatenating in ascending order the output of each sub-leaf 1 to 3. For the output of each sub-leaf, byte are ordered in ascending order of EAX:EBX:ECX:EDX.

Leaf 17H sub-leaves 4 and above are reserved.

Table 21-56. Leaf 17H.01H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_0_to_3	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_4_to_7	SOC Vendor Brand String. UTF-8 encoded string.	Platform
ECX[31:0]	VENDOR_BRAND_STRING_BYTES_8_to_11	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_12_to_15	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.02H -- Vendor Brand String Sub-Leaf (Bytes 16 to 31)

Table 21-57. Leaf 17H.02H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_16_to_19	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_20_to_23	SOC Vendor Brand String. UTF-8 encoded string.	Platform

ECX[31:0]	VENDOR_BRAND_STRING_BYTES_24_to_27	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_28_to_31	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.03H -- Vendor Brand String Sub-Leaf (Bytes 32 to 47)

Table 21-58. Leaf 17H.03H System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	VENDOR_BRAND_STRING_BYTES_32_to_35	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EBX[31:0]	VENDOR_BRAND_STRING_BYTES_36_to_39	SOC Vendor Brand String. UTF-8 encoded string.	Platform
ECX[31:0]	VENDOR_BRAND_STRING_BYTES_40_to_43	SOC Vendor Brand String. UTF-8 encoded string.	Platform
EDX[31:0]	VENDOR_BRAND_STRING_BYTES_44_to_47	SOC Vendor Brand String. UTF-8 encoded string.	Platform

CPUID.17H.M>MAXSOCID_INDEX—RESERVED SUB-LEAVES -- m>MaxSOCID_Index—Reserved Sub-Leaves

CPUID.17H with ECX > MaxSOCID_Index is reserved and returns all zeroes.

Table 21-59. Leaf 17H.M>MAXSOCID_INDEX—RESERVED SUB-LEAVES System-on-Chip Vendor Attribute

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved	
EBX[31:0]	Reserved	Reserved	
ECX[31:0]	Reserved	Reserved	
EDX[31:0]	Reserved	Reserved	

CPUID.18H -- Deterministic Address Translation Parameters

CPUID.18H returns information about the Deterministic Address Translation Parameters. Each sub-leaf enumerates a different address translation structure.

- This leaf is valid if CPUID.18H.00H:EDX[4:0] $\neq 0$ and MAX_LEAF $\geq 18H$.
- The maximum sub-leaf value for ECX is specified in CPUID.18H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX. A sub-leaf index is also invalid if EDX[4:0] returns 0.
- Valid sub-leaves do not need to be contiguous or in any particular order. A valid sub-leaf may be in a higher input ECX value than an invalid sub-leaf or than a valid sub-leaf of a higher or lower-level structure.

CPUID.18H.00H -- Main Sub-Leaf

Deterministic Address Translation Parameters Main Leaf

Table 21-60. Leaf 18H.00H Deterministic Address Translation Parameters

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum input value of supported sub-leaf in leaf 18H.	Logical Processor
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[4:0]	Reserved	Will always return 0.	Logical Processor
EDX[31:5]	Reserved	Reserved.	

CPUID.18H.ECX ≥ 1 -- Sub-Leaves

CPUID.18H with $ECX \geq 1$ returns Deterministic Address Translation Parameters information.

Table 21-61. Leaf 18H.ECX ≥ 1 Deterministic Address Translation Parameters

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[0]	4KB_ENTRIES	If 1, supports 4K page size entries in this structure.	Logical Processor
EBX[1]	2MB_ENTRIES	If 1, supports 2MB page size entries in this structure.	Logical Processor
EBX[2]	4MB_ENTRIES	If 1, supports 4MB page size entries in this structure.	Logical Processor
EBX[3]	1GB_ENTRIES	If 1, supports 1 GB page size entries in this structure.	Logical Processor
EBX[7:4]	Reserved	Reserved.	
EBX[10:8]	PARTITIONING	Partitioning (0:Soft partitioning between the logical processors sharing this structure).	Logical Processor
EBX[15:11]	Reserved	Reserved.	
EBX[31:16]	NUM_WAYS	W = ways of associativity.	Logical Processor
ECX[31:0]	NUM_SETS	S = number of sets.	Logical Processor

EDX[4:0]	TYPE	00000b: Null (indicates this Sub-leaf is not valid). 00001b: Data TLB. 00010b: Instruction TLB. 00011b: Unified TLB.1 00100b: Load Only TLB. Hit on loads; fills on both loads and stores. 00101b: Store Only TLB. Hit on stores; fill on stores. All other encodings are reserved. Some unified TLBs will allow a single TLB entry to satisfy data read/write and instruction fetches. Others will require separate entries (e.g., one loaded on data read/write and another loaded on an instruction fetch). See the Intel® 64 and IA-32 Architectures Optimization Reference Manual for details of a particular product.	Logical Processor
EDX[7:5]	LEVEL_NUM	Translation cache level (starts at 1).	Logical Processor
EDX[8]	FULLY_ASSOC	Fully associative structure.	Logical Processor
EDX[13:9]	Reserved	Reserved.	
EDX[25:14]	MAX_LP_ADDRESSABLE_IDS	Maximum number of addressable IDs for logical processors sharing this translation cache. Add one to the return value to get the result.	Logical Processor
EDX[31:26]	Reserved	Reserved.	

CPUID.19H -- Key Locker

CPUID.19H returns Key Locker information.

- This leaf is valid if CPUID.07H.00H:ECX.KEY_LOCKER[23] = 1 and MAX_LEAF ≥ 19H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-62. Leaf 19H Key Locker

Register	Field Name	Description	Domain
EAX[0]	CPLD_RESTRICT	If 1, supports the Key Locker restriction of CPLD-only. ¹	Platform
EAX[1]	NO_ENCRYPT_RESTRICT	If 1, supports the Key Locker restriction of no-encrypt. ¹	Platform
EAX[2]	NO_DECRYPT_RESTRICT	If 1, supports the Key Locker restriction of no-decrypt. ¹	Platform
EAX[31:3]	Reserved	Reserved.	
EBX[0]	AESKLE	If 1, the AES Key Locker instructions are fully enabled. CPUID.19H:EBX.AESKLE[0] is enumerated as 1 if the AES Key Locker instructions have been activated by system firmware and CR4.KL[bit 19] = 1. Software can check this bit after setting CR4.KL to determine whether AES Key Locker instructions have been enabled. Note that some processors may allow enabling of those instructions without activation by system firmware. Some processors may not support the use of AES Key Locker instructions in system-management-mode (SMM). Those processors enumerate CPUID.19H:EBX.AESKLE[0] as 0 in SMM regardless of the setting of CR4.KL.	Logical Processor
EBX[1]	Reserved	Reserved.	
EBX[2]	AES_WIDE	If 1, supports the AES wide Key Locker instructions. ¹	Platform
EBX[3]	Reserved	Reserved.	
EBX[4]	IWKEYBACKUP	If 1, supports the Key Locker MSRs (IA32_COPY_LOCAL_TO_PLATFORM, IA23_COPY_PLATFORM_TO_LOCAL, IA32_COPY_STATUS, and IA32_IWKEYBACKUP_STATUS) and backing up the internal wrapping key. ¹	Platform
EBX[31:5]	Reserved	Reserved.	
ECX[0]	NOBACKUP	If 1, supports the NoBackup parameter to LAADIWKEY. ¹	Platform
ECX[1]	RAND_IWKEY	If 1, supports KeySource encoding of 1 (randomization of the internal wrapping key). ¹	Platform
ECX[31:2]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

NOTE

1. This field is valid only if CPUID.19H:EBX.AESKLE[0] = 1.

CPUID.1AH -- Native Model ID Enumeration

CPUID.1AH returns Native Model ID information.
This leaf exists on all logical processors in a hybrid package, it may also be present in other processor configurations.

- This leaf is valid if CPUID.1AH.00H:EAX[31:0] <> 0 and MAX_LEAF ≥ 1AH.
- The only valid sub-leaf is 0 and ECX must be set to 0.

Table 21-63. Leaf 1AH Native Model ID Enumeration

Register	Field Name	Description	Domain
EAX[23:0]	CORE_NATIVE_MODEL_ID	The core-type and native model ID can be used to uniquely identify the microarchitecture of the core. This native model ID is not unique across core types, and not related to the model ID reported in CPUID.01H, and does not identify the SOC.	Logical Processor
EAX[31:24]	CORE_TYPE	10H: Reserved 20H: Intel® Atom® 30H: Reserved 40H: Intel® Core The core type may only be used as an identification of the microarchitecture for this logical processor and its numeric value has no significance, neither large nor small. This field neither implies nor expresses any other attribute to this logical processor and software should not assume any.	Logical Processor
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1BH -- PCONFIG Information

CPUID.1BH -- Output Registers Format for All Sub-Leaves

CPUID.1BH returns information for PCONFIG capabilities. This information is enumerated in sub-leaves selected by the value of ECX (starting with 0).

- This leaf is valid if CPUID.07H.00H:EDX.PCONFIG[18] = 1 and MAX_LEAF ≥ 1BH.
- Sub-leaves are enumerated until sub-leaf n, where EAX[11:0] returns 0.

Table 21-64. Leaf 1BH PCONFIG Information

Register	Field Name	Description	Domain
EAX[11:0]	SUB_LEAF_TYPE	1 (Target Identifier)	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

Each sub-leaf of CPUID.1BH enumerates its sub-leaf type in EAX. If a sub-leaf type is 0, the sub-leaf is invalid and zero is returned in EBX, ECX, and EDX. In this case, all subsequent sub-leaves (selected by larger input values of ECX) are also invalid.

The only valid sub-leaf type currently defined is 1, indicating that the sub-leaf enumerates target identifiers for the PCONFIG instruction. Any non-zero value returned in EBX, ECX, or EDX indicates a valid target identifier of the PCONFIG instruction (any value of zero should be ignored). The only target identifier currently defined is 1, indicating TME-MK. See the “PCONFIG—Platform Configuration” instruction in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B, for more information.

CPUID.1BH.OUTPUT REGISTERS FOR SUB-LEAF TYPE TARGET IDENTIFIER (1) -- Output Registers for Sub-Leaf Type Target Identifier (1)

Table 21-65. Leaf 1BH.OUTPUT REGISTERS FOR SUB-LEAF TYPE TARGET IDENTIFIER (1) PCONFIG Information

Register	Field Name	Description	Domain
EAX[11:0]	SUB_LEAF_TYPE	1 (Target Identifier)	Platform
EAX[31:12]	Reserved	Reserved.	
EBX[31:0]	TARGET_IDENTIFIER_1	Target identifier	Platform
ECX[31:0]	TARGET_IDENTIFIER_2	Target identifier	Platform
EDX[31:0]	TARGET_IDENTIFIER_3	Target identifier	Platform

The only current valid target identifier is 1 for MK-TME.

CPUID.1CH -- Last Branch Records (LBR) Information

CPUID.1CH returns information about architectural Last Branch Records (LBR). For details on LBR, see Chapter 20, "Last Branch Records," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

- This leaf is valid if CPUID.07H.00H:EDX.ARCH_LBRS[19] = 1 and MAX_LEAF ≥ 1CH.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-66. Leaf 1CH Last Branch Records (LBR) Information

Register	Field Name	Description	Domain
EAX[7:0]	LBR_DEPTH_VALUES	For each bit <i>n</i> set in this field, the IA32_LBR_DEPTH.DEPTH value $8 \cdot (n+1)$ is supported.	Platform
EAX[29:8]	Reserved	Reserved.	
EAX[30]	DEEP_C_STATE_RESET	If 1, supports that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1.	Platform
EAX[31]	IP_VALUES_CONTAIN_LIP	If 1, the LBR IP values contain LIP. If 0, IP values contain Effective IP. Trace segments using a flat memory model will generate the same information regardless of how a logical processor reports this value since LIP = EIP	Logical Processor
EBX[0]	CPL_FILTERING	If 1, supports setting IA32_LBR_CTL[2:1] to a non-zero value.	Platform
EBX[1]	BRANCH_FILTERING	If 1, supports setting IA32_LBR_CTL[22:16] to a non-zero value.	Platform
EBX[2]	CALL_STACK_MODE	If 1, supports setting IA32_LBR_CTL[3] to 1.	Platform
EBX[31:3]	Reserved	Reserved.	
ECX[0]	MISPREDICT_BIT	If 1, IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED).	Platform
ECX[1]	TIMED_LBRS	If 1, IA32_LBR_x_INFO[15:0] holds CPU cycles since last LBR entry (CYC_CNT), and IA32_LBR_x_INFO[60] holds an indication of whether the value held there is valid (CYC_CNT_VALID).	Platform
ECX[2]	BRANCH_TYPE_FIELD_SUPPORTED	If 1, IA32_LBR_INFO_x[59:56] holds indication of the recorded operation's branch type (BR_TYPE).	Platform
ECX[15:3]	Reserved	Reserved.	
ECX[19:16]	EVENT_LOGGING_BITMAP	The event logging bitmap is supported.	Platform
ECX[31:20]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1DH -- Tile Information

CPUID.1DH returns information about tile architecture and tile palette 1 (see Chapter 19, “Programming with Intel® Advanced Matrix Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

- This leaf is valid if CPUID.07H.00H:EDX.AMX_TILE[24] = 1 and MAX_LEAF ≥ 1DH.
- The maximum sub-leaf value for ECX is specified in CPUID.1DH.00H:EAX[31:0] max_palette.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.1DH.00H -- Tile Information Main Sub-Leaf

CPUID.1DH.00H returns the tile architecture information.

Table 21-67. Leaf 1DH.00H Tile Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_PALETTE	Highest numbered palette sub-leaf. Value = 1.	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1DH.01H -- Tile Palette 1

CPUID.1DH.01H returns tile palette information.

Table 21-68. Leaf 1DH.01H Tile Information

Register	Field Name	Description	Domain
EAX[15:0]	TOTAL_TILE_BYTES	Palette 1 total_tile_bytes. Value = 8192.	Platform
EAX[31:16]	BYTES_PER_TILE	Palette 1 bytes_per_tile. Value = 1024.	Platform
EBX[15:0]	BYTES_PER_ROW	Palette 1 bytes_per_row. Value = 64.	Platform
EBX[31:16]	MAX_NAMES	Palette 1 max_names (number of tile registers). Value = 8.	Platform
ECX[15:0]	MAX_ROWS	Palette 1 max_rows. Value = 16.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1EH -- TMUL Information

CPUID.1EH returns information about TMUL capabilities (see Chapter 19, “Programming with Intel® Advanced Matrix Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

- This leaf is valid if CPUID.07H.00H:EDX.AMX_TILE[24] = 1 and MAX_LEAF ≥ 1EH.
- The only valid sub-leaf is 0 and ECX must be set to 0.

CPUID.1EH.00H -- TMUL Information Main Leaf

TMUL Main Leaf Information

Table 21-69. Leaf 1EH.00H TMUL Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	The Maximum number of subleaves.	Platform
EBX[7:0]	TMUL_MAXK	tmul_maxk (rows or columns). Value = 16.	Platform
EBX[23:8]	TMUL_MAXN	tmul_maxn (column bytes). Value = 64.	Platform
EBX[31:24]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.1FH -- V2 Extended Topology

CPUID.1FH returns information about V2 Extended Topology.

CPUID.1FH is a preferred superset to leaf 0BH. Intel recommends using leaf 1FH when available rather than leaf 0BH and ensuring that any leaf 0BH algorithms are updated to support leaf 1FH.

- This leaf is valid if CPUID.1FH.00H:EBX[15:0] $\neq$ 0 and MAX_LEAF $\geq$ 1FH.
- The sub-leaves are enumerated until sub-leaf n returns 0 in EBX[15:0].
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n+1 is invalid if sub-leaf n returns EBX[15:0] as 0.

CPUID.1FH -- ECX $\geq$ 0

Table 21-70. Leaf 1FH V2 Extended Topology

Register	Field Name	Description	Domain
EAX[4:0]	SHIFT_COUNT	The number of bits that the x2APIC ID must be shifted to the right to address instances of the next higher-scoped domain. When logical processor is not supported by the processor, the value of this field at the Logical Processor domain sub-leaf may be returned as either 0 (no allocated bits in the x2APIC ID) or 1 (one allocated bit in the x2APIC ID); software should plan accordingly.	Platform
EAX[31:5]	Reserved	Reserved.	
EBX[15:0]	NEXT_LEVEL_NUM_LP	The number of logical processors across all instances of this domain within the next higher-scoped domain relative to this current logical processor. (For example, in a processor socket/package comprising "M" dies of "N" cores each, where each core has "L" logical processors, the "die" domain sub-leaf value of this field would be M*N*L. In an asymmetric topology this would be the summation of the value across the lower domain level instances to create each upper domain level instance.) This number reflects configuration as shipped by Intel. Note that the number of logical processors can be asymmetric in which case "L" may be different on different logical processors, as an example a core with 2 logical processors on the same platform as a core with 1 logical processor. Note, software must not use this field to enumerate processor topology. Software must not use the value of EBX[15:0] to enumerate processor topology of the system. The value is only intended for display and diagnostic purposes. The actual number of logical processors available to BIOS/OS/Applications may be different from the value of EBX[15:0], depending on software and platform hardware configurations.	Logical Processor
EBX[31:16]	Reserved	Reserved.	
ECX[7:0]	LEVEL_NUM	The input ECX sub-leaf index.	Platform

ECX[15:8]	DOMAIN_TYPE	This field provides an identification value which indicates the domain as shown in the table. Although domains are ordered, their assigned identification values are not and software should not depend on it. (For example, if a new domain between core and module is specified, it will have an identification value higher than 5.) See the table below for the current list of valid enumerations. Note that enumeration values of 0 and 7-255 are reserved.	Platform
ECX[31:16]	Reserved	Reserved.	
EDX[31:0]	X2APIC_ID	The x2APIC ID of the current logical processor is always valid and does not vary with the sub-leaf index in ECX.	Logical Processor

The sub-leaves of CPUID.1FH describe an ordered hierarchy of logical processors starting from the smallest scoped domain of a Logical Processor (sub-leaf index 0) to the Core domain (sub-leaf index 1) to the largest scoped domain (the last valid sub-leaf index) that is implicitly subordinate to the unenumerated highest-scoped domain of the processor package (socket).

The details of each valid domain is enumerated by a corresponding sub-leaf. Details for a domain include its type and how all instances of that domain determine the number of logical processors and x2 APIC ID partitioning at the next higher-scoped domain. The ordering of domains within the hierarchy is fixed architecturally as shown below. For a given processor, not all domains may be relevant or enumerated; however, the logical processor and core domains are always enumerated. As an example, a processor may report an ordered hierarchy consisting only of "Logical Processor," "Core," and "Die."

For two valid sub-leaves N and N+1, sub-leaf N+1 represents the next immediate higher-scoped domain with respect to the domain of sub-leaf N for the given processor.

If sub-leaf index "N" returns an invalid domain type in ECX[15:08] (00H), then all sub-leaves with an index greater than "N" shall also return an invalid domain type. A sub-leaf returning an invalid domain always returns 0 in EAX and EBX.

Table 21-71. Hierarchy of Valid Domain Enumerations in CPUID.1FH:ECX[15:8]

Hierarchy	Domain	Domain Type ID Value
Invalid	Invalid	0
Lowest	Logical Processor	1
...	Core	2
...	Module	3
...	Tile	4
...	Die	5
...	DieGrp	6
Highest	Package/Socket	(Implied)
Reserved	Reserved	7-255

CPUID.20H -- Processor History Reset Information

CPUID.20H returns information about processor history reset when CPUID.07H.01H:EAX.HRESET[22] = 1.

- This leaf is valid if CPUID.07H.01H:EAX.HRESET[22] = 1 and MAX_LEAF ≥ 20H.
- The maximum sub-leaf value for ECX is specified in CPUID.20H.00H:EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.20H.00H -- Processor History Reset Sub-leaf

Table 21-72. Leaf 20H.00H Processor History Reset Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum number of sub-leaves that are supported in leaf 20H.	Platform
EBX[0]	THREAD_DIRECTOR_HRESET	Indicates support for both HRESET's EAX[0] parameter, and IA32_HRESET_ENABLE[0] set by the OS to enable reset of Intel® Thread Director history.	Platform
EBX[31:1]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.21H -- Reserved

This leaf is reserved.

Table 21-73. Leaf 21H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.22H -- Reserved

This leaf is reserved.

Table 21-74. Leaf 22H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H -- Architectural Performance Monitoring Extended

CPUID.23H returns architectural performance monitoring extended information.

- This leaf is valid if CPUID.07H.01H:EAX.ARCH_PERFMON_EXT[8] = 1 and MAX_LEAF ≥ 23H.
- The sub-leaves of this leaf are enumerated by a bitmask specified in CPUID.23H.00H.EAX[31:0] Available SubLeaf Mask. The bit numbers of set bits in the bitmask represent valid sub-leaf indexes.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index is invalid if the index as a bit number is clear in the Available Sub-Leaf Mask or is greater than 31.

CPUID.23H.00H -- Main Sub-Leaf

Table 21-75. Leaf 23H.00H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	SUBLEAF_MASK	If bit n is set, sub-leaf n is supported. (For unsupported sub-leaves, 0 is returned in the registers EAX, EBX, ECX, and EDX.)	Logical Processor
EBX[0]	UNITMASK2	If 1, supports the UnitMask2 field in the IA32_PERFEVTSELx MSRs.	Logical Processor
EBX[1]	EQ	If 1, supports the equal flag in the IA32_PERFEVTSELx MSRS.	Logical Processor
EBX[31:2]	Reserved	Reserved.	
ECX[7:0]	SLOTS_PER_CYC	If this field is non-zero, it represents the number of Top-down Microarchitecture Analysis (TMA) slots per cycle. This number can be multiplied by the number of cycles (from CPU_CLK_UNHALTED.THREAD / CPU_CLK_UNHALTED.CORE or IA32_FIXED_CTR1) to determine the total number of slots. If this field is zero, IA32_FIXED_CTR3 should be used to determine the total number of slots.	Logical Processor
ECX[31:8]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.01H -- Counter Information Sub-Leaf

Table 21-76. Leaf 23H.01H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	GP_COUNTERS	For each bit n set in this field, the processor supports general-purpose performance monitoring counter n.	Logical Processor
EBX[31:0]	FIXED_COUNTERS	For each bit m set in this field, the processor supports fixed-function performance monitoring counter m. The valid range of fixed-function counters is 0 through 15.	Logical Processor
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.23H.02H -- Bitmap of Auto Counter Reload Sub-Leaf

Table 21-77. Leaf 23H.02H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[31:0]	ACR_GP_RELOAD	General counters that can be reloaded. For each bit n set in this field, the processor supports ACR for general-purpose performance monitoring counter n.	Logical Processor
EBX[31:0]	ACR_FIXED_RELOAD	Fixed counters that can be reloaded. For each bit m set in this field, the processor supports ACR for fixed-function performance monitoring counter m.	Logical Processor
ECX[31:0]	ACR_GP_TRIGGER	General counters that can cause reloads. For each bit y set in this field, the processor allows general-purpose performance monitoring counter y to reload all existing general-purpose performance monitoring counters capable of being reloaded.	Logical Processor
EDX[31:0]	ACR_FIXED_TRIGGER	Fixed counters that can cause reloads. For each bit x set in this field, the processor allows fixed-function performance monitoring counter x to reload all existing fixed-function performance monitoring counters capable of being reloaded.	Logical Processor

CPUID.23H.03H -- Architectural Performance Monitoring Events Bitmap Sub-Leaf

For each bit n set in this field, the processor supports Architectural Performance Monitoring Event of index n.

Table 21-78. Leaf 23H.03H Architectural Performance Monitoring Extended

Register	Field Name	Description	Domain
EAX[0]	CORE_CYC	If 1, supports architectural index 0.	Logical Processor
EAX[1]	INSTR_RET	If 1, supports architectural index 1.	Logical Processor
EAX[2]	REF_CYC	If 1, supports architectural index 2.	Logical Processor
EAX[3]	LLC_REF	If 1, supports architectural index 3.	Logical Processor
EAX[4]	LLC_MISSES	If 1, supports architectural index 4.	Logical Processor
EAX[5]	BR_INSTR_RET	If 1, supports architectural index 5	Logical Processor
EAX[6]	BR_MISPRED_RET	If 1, supports architectural index 6	Logical Processor
EAX[7]	SLOTS	If 1, supports architectural index 7	Logical Processor
EAX[8]	BACKEND	If 1, supports architectural index 8	Logical Processor
EAX[9]	BADSPEC	If 1, supports architectural index 9	Logical Processor

PROCESSOR IDENTIFICATION AND FEATURE DETERMINATION

EAX[10]	FRONTEND	If 1, supports architectural index 10	Logical Processor
EAX[11]	RETIRING	If 1, supports architectural index 11	Logical Processor
EAX[12]	LBR_INSERTS	If 1, supports architectural index 12	Logical Processor
EAX[31:13]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.24H -- Converged Vector ISA

When CPUID.24H, the processor returns Intel AVX10 converged vector ISA information.

This leaf is supported when CPUID.07H.01H:EDX.AVX10[19] = 1.

- This leaf is valid if CPUID.07H.01H:EDX.AVX10[19] = 1 and MAX_LEAF ≥ 24H.
- The maximum sub-leaf value for ECX is specified in CPUID.24H.00H.EAX[31:0] MAX_SUBLEAF.
- If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf index n is invalid if n exceeds the value that sub-leaf 0 returns in EAX.

CPUID.24H.00H -- Converged Vector ISA Main Sub-Leaf

Table 21-79. Leaf 24H.00H Converged Vector ISA

Register	Field Name	Description	Domain
EAX[31:0]	MAX_SUBLEAF	Reports the maximum number of sub-leaves that are supported in leaf 24H.	Platform
EBX[7:0]	VECTOR_ISA_VERSION	Reports the Intel® AVX10 Converged Vector ISA version.	Platform
EBX[15:8]	Reserved	Reserved.	
EBX[18:16]	Reserved at 111	Always 111b. Earlier versions of this specification documented these bits as enumerating support for different vector lengths. Processors enumerating Intel® AVX10 support all vector widths.	Platform
EBX[31:19]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000000H -- Maximum Input Value for Extended Function CPUID Information

CPUID.80000000H returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

- This leaf is supported starting with Pentium 4.
- Processors prior to Pentium 4 treat bit 31 as 0, and this leaf returns the values from CPUID.00H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-80. Leaf 80000000H Maximum Input Value for Extended Function CPUID Information

Register	Field Name	Description	Domain
EAX[31:0]	MAX_EXTENDED_LEAF	Maximum input value for Extended Function CPUID Information.	Platform
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000001H -- Extended Processor Signature and Feature Bits

CPUID.80000001H returns information about extended processor signature and features bits.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000001H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-81. Leaf 80000001H Extended Processor Signature and Feature Bits

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[0]	LAHF_SAHF_64	If 1, supports the LAHF/SAHF instructions in 64-bit mode. LAHF and SAHF are always available in other modes, regardless of the enumeration of this feature flag.	Platform
ECX[4:1]	Reserved	Reserved.	
ECX[5]	LZCNT	If 1, supports the LZCNT instruction.	Platform
ECX[7:6]	Reserved	Reserved.	
ECX[8]	PREFETCHW	If 1, supports the PREFETCHW instruction.	Platform
ECX[31:9]	Reserved	Reserved.	
EDX[10:0]	Reserved	Reserved.	
EDX[11]	SYSCALL_SYSRET_64	If 1, supports SYSCALL/SYSRET. Intel processors support SYSCALL and SYSRET only in 64-bit mode. This feature flag is always enumerated as 0 outside 64-bit mode.	Platform
EDX[19:12]	Reserved	Reserved.	
EDX[20]	EXECUTE_DIS	If 1, supports Execute Disable Bit.	Platform
EDX[25:21]	Reserved	Reserved.	
EDX[26]	PAGE_1GB	If 1, supports 1-GByte pages.	Platform
EDX[27]	RDTSCP	If 1, supports RDTSCP and IA32_TSC_AUX.	Platform
EDX[28]	Reserved	Reserved.	
EDX[29]	INTEL64	If 1, supports Intel® 64 Architecture.	Platform
EDX[31:30]	Reserved	Reserved.	

CPUID.80000002H -- Processor Brand String (Bytes 0 to 15)

CPUID.80000002H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000002H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-82. Leaf 80000002H Processor Brand String (Bytes 0 to 15)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_0	Processor brand string.	Platform
EBX[31:0]	BRAND_NAME_1	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_2	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_3	Processor brand string continued.	Platform

CPUID.80000003H -- Processor brand string (Bytes 16 to 31)

CPUID.80000003H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000003H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-83. Leaf 80000003H Processor brand string (Bytes 16 to 31)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_4	Processor brand string continued.	Platform
EBX[31:0]	BRAND_NAME_5	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_6	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_7	Processor brand string continued.	Platform

CPUID.80000004H -- Processor brand string (Bytes 32 to 47)

CPUID.80000004H returns information about the Processor Brand String. For additional details on Processor Brand String, see Section 21.2, “Methods for Returning Branding Information Using CPUID.”

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000004H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-84. Leaf 80000004H Processor brand string (Bytes 32 to 47)

Register	Field Name	Description	Domain
EAX[31:0]	BRAND_NAME_8	Processor brand string continued.	Platform
EBX[31:0]	BRAND_NAME_9	Processor brand string continued.	Platform
ECX[31:0]	BRAND_NAME_10	Processor brand string continued.	Platform
EDX[31:0]	BRAND_NAME_11	Processor brand string continued.	Platform

CPUID.80000005H -- Reserved

This leaf is reserved and returns all zeroes.

Table 21-85. Leaf 80000005H Reserved

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

CPUID.80000006H -- Extended Function CPUID Information

CPUID.80000006H returns Extended Function CPUID information. The preferred method to enumerate caching information description>is to use CPUID.04H—Deterministic Cache Parameters.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000006H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX

Table 21-86. Leaf 80000006H Extended Function CPUID Information

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[7:0]	L2_LINE_SIZE	Cache line size in bytes.	Logical Processor
ECX[11:8]	Reserved	Reserved.	
ECX[15:12]	L2_ASSOC	L2 associativity field. The L2 associativity field encodings are listed in the table below.	Logical Processor
ECX[31:16]	L2_SIZE	Cache size in 1K units.	Logical Processor
EDX[7:0]	Reserved	Reserved.	
EDX[31:8]	Reserved	Reserved.	

Table 21-87. L2 Associativity Field Encodings

Encoding Value	Description	Encoding Value	Description
00H	Disabled	08H	16 Ways
01H	1 Way (direct mapped)	09H	Reserved
02H	2 Ways	0AH	32 Ways
03H	Reserved	0BH	48 Ways
04H	4 Ways	0CH	64 Ways
05H	Reserved	0DH	96 Ways
06H	8 Ways	0EH	128 Ways
07H	See CPUID leaf 4 sub-leaf 2 ¹	0FH	Fully Associative

CPUID.80000007H -- Extended Function CPUID Information 1

CPUID.80000007H returns Extended Function CPUID information.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000007H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-88. Leaf 80000007H Extended Function CPUID Information 1

Register	Field Name	Description	Domain
EAX[31:0]	Reserved	Reserved.	
EBX[31:0]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[7:0]	Reserved	Reserved.	
EDX[8]	TSC_INVARIANT	If 1, supports Invariant TSC.	Platform
EDX[31:9]	Reserved	Reserved.	

CPUID.80000008H -- Extended Function CPUID Information 2

CPUID.80000008H returns Extended Function CPUID information.

- This leaf is valid if MAX_EXTENDED_LEAF ≥ 80000008H.
- This leaf does not contain sub-leaves and provides the same information regardless of the value of ECX.

Table 21-89. Leaf 80000008H Extended Function CPUID Information 2

Register	Field Name	Description	Domain
EAX[7:0]	PHYS_ADDR_SIZE	Number of physical address bits. If CPUID.80000008H:EAX[7:0] is supported, the maximum physical address number supported should come from this field. If TME-MK is enabled, the number of bits that can be used to address physical memory is CPUID.80000008H:EAX[7:0] - IA32_TME_ACTIVATE[35:32].	Platform
EAX[15:8]	LIN_ADDR_SIZE	Number of linear address bits.	Platform
EAX[23:16]	GUEST_PHYS_ADDR_SIZE	This value applies only to software operating in a virtual machine (Intel processors enumerate this value as zero). When this field is zero, refer to Physical Address Bits for the number of guest physical address bits	Platform
EAX[31:24]	Reserved	Reserved.	
EBX[8:0]	Reserved	Reserved.	
EBX[9]	WBNOINVD	If 1, supports the WBNOINVD instruction.	Platform
EBX[31:10]	Reserved	Reserved.	
ECX[31:0]	Reserved	Reserved.	
EDX[31:0]	Reserved	Reserved.	

4. Updates to Chapter 3, Volume 2A

Change bars and violet text show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-L*.

Changes to this chapter:

- For instruction CPUID—CPU Identification, several sections were moved to Chapter 21 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*.
- Instruction INVD—Invalidate Internal Caches was updated for UDB.

CHAPTER 3 INSTRUCTION SET REFERENCE, A-L

This chapter describes the instruction set for the Intel 64 and IA-32 architectures (A-L) in IA-32e, protected, virtual-8086, and real-address modes of operation. The set includes general-purpose, x87 FPU, MMX, SSE/SSE2/SSE3/SSSE3/SSE4, AESNI/PCLMULQDQ, AVX, and system instructions. See also Chapter 4, “Instruction Set Reference, M-U,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B; Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C; and Chapter 6, “Instruction Set Reference, W-Z,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

For each instruction, each operand combination is described. A description of the instruction and its operand, an operational description, a description of the effect of the instructions on flags in the EFLAGS register, and a summary of exceptions that can be generated are also provided.

3.1 INTERPRETING THE INSTRUCTION REFERENCE PAGES

This section describes the format of information contained in the instruction reference pages in this chapter. It explains notational conventions and abbreviations used in these sections.

3.1.1 Instruction Format

The following is an example of the format used for each instruction description in this chapter. The heading below introduces the example. The table below provides an example summary table.

CMC—Complement Carry Flag [this is an example]

Opcode	Instruction	Op/En	64/32-bit Mode	CPUID Feature Flag	Description
F5	CMC	Z0	V/V	N/A	Complement carry flag.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

3.1.1.1 Opcode Column in the Instruction Summary Table (Instructions without VEX Prefix)

The “Opcode” column in the table above shows the object code produced for each form of the instruction. When possible, codes are given as hexadecimal bytes in the same order in which they appear in memory. Definitions of entries other than hexadecimal bytes are as follows:

- **NP** — Indicates the use of 66/F2/F3 prefixes (beyond those already part of the instructions opcode) are not allowed with the instruction. Such use will either cause an invalid-opcode exception (#UD) or result in the encoding for a different instruction.
- **NF_x** — Indicates the use of F2/F3 prefixes (beyond those already part of the instructions opcode) are not allowed with the instruction. Such use will either cause an invalid-opcode exception (#UD) or result in the encoding for a different instruction.
- **REX.W** — Indicates the use of a REX prefix that affects operand size or instruction semantics. The ordering of the REX prefix and other optional/mandatory instruction prefixes are discussed Chapter 2. Note that REX prefixes that promote legacy instructions to 64-bit behavior are not listed explicitly in the opcode column.
- **/digit** — A digit between 0 and 7 indicates that the ModR/M byte of the instruction uses only the r/m (register or memory) operand. The reg field contains the digit that provides an extension to the instruction's opcode.
- **/r** — Indicates that the ModR/M byte of the instruction contains a register operand and an r/m operand.
- **cb, cw, cd, cp, co, ct** — A 1-byte (cb), 2-byte (cw), 4-byte (cd), 6-byte (cp), 8-byte (co) or 10-byte (ct) value following the opcode. This value is used to specify a code offset and possibly a new value for the code segment register.
- **ib, iw, id, io** — A 1-byte (ib), 2-byte (iw), 4-byte (id) or 8-byte (io) immediate operand to the instruction that follows the opcode, ModR/M bytes or scale-indexing bytes. The opcode determines if the operand is a signed value. All words, doublewords, and quadwords are given with the low-order byte first.
- **+rb, +rw, +rd, +ro** — Indicated the lower 3 bits of the opcode byte is used to encode the register operand without a modR/M byte. The instruction lists the corresponding hexadecimal value of the opcode byte with low 3 bits as 000b. In non-64-bit mode, a register code, from 0 through 7, is added to the hexadecimal value of the opcode byte. In 64-bit mode, indicates the four bit field of REX.b and opcode[2:0] field encodes the register operand of the instruction. “+ro” is applicable only in 64-bit mode. See Table 3-1 for the codes.
- **+i** — A number used in floating-point instructions when one of the operands is ST(i) from the FPU register stack. The number i (which can range from 0 to 7) is added to the hexadecimal byte given at the left of the plus sign to form a single opcode byte.

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
AL	None	0	AX	None	0	EAX	None	0	RAX	None	0
CL	None	1	CX	None	1	ECX	None	1	RCX	None	1
DL	None	2	DX	None	2	EDX	None	2	RDX	None	2
BL	None	3	BX	None	3	EBX	None	3	RBX	None	3
AH	Not encodable (N.E.)	4	SP	None	4	ESP	None	4	N/A	N/A	N/A
CH	N.E.	5	BP	None	5	EBP	None	5	N/A	N/A	N/A
DH	N.E.	6	SI	None	6	ESI	None	6	N/A	N/A	N/A
BH	N.E.	7	DI	None	7	EDI	None	7	N/A	N/A	N/A
SPL	Yes	4	SP	None	4	ESP	None	4	RSP	None	4
BPL	Yes	5	BP	None	5	EBP	None	5	RBP	None	5

Table 3-1. Register Codes Associated With +rb, +rw, +rd, +ro (Contd.)

byte register			word register			dword register			quadword register (64-Bit Mode only)		
Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field	Register	REX.B	Reg Field
SIL	Yes	6	SI	None	6	ESI	None	6	RSI	None	6
DIL	Yes	7	DI	None	7	EDI	None	7	RDI	None	7
Registers R8 - R15 (see below): Available in 64-Bit Mode Only											
R8B	Yes	0	R8W	Yes	0	R8D	Yes	0	R8	Yes	0
R9B	Yes	1	R9W	Yes	1	R9D	Yes	1	R9	Yes	1
R10B	Yes	2	R10W	Yes	2	R10D	Yes	2	R10	Yes	2
R11B	Yes	3	R11W	Yes	3	R11D	Yes	3	R11	Yes	3
R12B	Yes	4	R12W	Yes	4	R12D	Yes	4	R12	Yes	4
R13B	Yes	5	R13W	Yes	5	R13D	Yes	5	R13	Yes	5
R14B	Yes	6	R14W	Yes	6	R14D	Yes	6	R14	Yes	6
R15B	Yes	7	R15W	Yes	7	R15D	Yes	7	R15	Yes	7

3.1.1.2 Opcode Column in the Instruction Summary Table (Instructions with VEX prefix)

In the Instruction Summary Table, the Opcode column presents each instruction encoded using the VEX prefix in following form (including the modR/M byte if applicable, the immediate byte if applicable):

VEX.[128,256].[66,F2,F3].0F/0F3A/0F38.[W0,W1] opcode [/r] [/ib,/is4]

- **VEX** — Indicates the presence of the VEX prefix is required. The VEX prefix can be encoded using the three-byte form (the first byte is C4H), or using the two-byte form (the first byte is C5H). The two-byte form of VEX only applies to those instructions that do not require the following fields to be encoded: VEX.mmmmm, VEX.W, VEX.X, VEX.B. Refer to Section 2.3 for more detail on the VEX prefix.

The encoding of various sub-fields of the VEX prefix is described using the following notations:

- **128,256:** VEX.L field can be 0 (denoted by VEX.128, VEX.L0, or VEX.LZ) or 1 (denoted by VEX.256 or VEX.L1). The VEX.L field can be encoded using either the 2-byte or 3-byte form of the VEX prefix. The presence of the notation VEX.256 or VEX.128 in the opcode column should be interpreted as follows:
 - If VEX.256 is present in the opcode column: The semantics of the instruction must be encoded with VEX.L = 1. An attempt to encode this instruction with VEX.L = 0 can result in one of two situations: (a) if VEX.128 version is defined, the processor will behave according to the defined VEX.128 behavior; (b) an #UD occurs if there is no VEX.128 version defined.
 - If VEX.128 is present in the opcode column but there is no VEX.256 version defined for the same opcode byte: Two situations apply: (a) For VEX-encoded, 128-bit SIMD integer instructions, software must encode the instruction with VEX.L = 0. The processor will treat the opcode byte encoded with VEX.L = 1 by causing an #UD exception; (b) For VEX-encoded, 128-bit packed floating-point instructions, software must encode the instruction with VEX.L = 0. The processor will treat the opcode byte encoded with VEX.L = 1 by causing an #UD exception (e.g., VMOVLPS).
 - If VEX.L0 or VEX.L1 is present in the opcode column: The specified VEX.L value is required for encoding this instruction but does not have the connotation of specifying vector length.
 - If VEX.LIG is present in the opcode column: The VEX.L value is ignored. This generally applies to VEX-encoded scalar SIMD floating-point instructions. Scalar SIMD floating-point instruction can be distinguished from the mnemonic of the instruction. Generally, the last two letters of the instruction mnemonic would be either "SS", "SD", or "SI" for SIMD floating-point conversion instructions.
 - If VEX.LZ is present in the opcode column: The VEX.L must be encoded to be 0B, an #UD occurs if VEX.L is not zero.

- **66,F2,F3:** The presence or absence of these values map to the VEX.pp field encodings. If absent, this corresponds to VEX.pp=00B. If present, the corresponding VEX.pp value affects the “opcode” byte in the same way as if a SIMD prefix (66H, F2H or F3H) does to the ensuing opcode byte. Thus a non-zero encoding of VEX.pp may be considered as an implied 66H/F2H/F3H prefix. The VEX.pp field may be encoded using either the 2-byte or 3-byte form of the VEX prefix.
- **0F,0F3A,0F38:** The presence maps to a valid encoding of the VEX.mmmmm field. Only three encoded values of VEX.mmmmm are defined as valid, corresponding to the escape byte sequence of 0FH, 0F3AH, and 0F38H. The effect of a valid VEX.mmmmm encoding on the ensuing opcode byte is same as if the corresponding escape byte sequence on the ensuing opcode byte for non-VEX encoded instructions. Thus a valid encoding of VEX.mmmmm may be considered as an implied escape byte sequence of either 0FH, 0F3AH or 0F38H. The VEX.mmmmm field must be encoded using the 3-byte form of VEX prefix.
- **0F,0F3A,0F38 and 2-byte/3-byte VEX:** The presence of 0F3A and 0F38 in the opcode column implies that opcode can only be encoded by the three-byte form of VEX. The presence of 0F in the opcode column does not preclude the opcode to be encoded by the two-byte of VEX if the semantics of the opcode does not require any subfield of VEX not present in the two-byte form of the VEX prefix.
- **W0:** VEX.W=0.
- **W1:** VEX.W=1.
- The presence of W0/W1 in the opcode column applies to two situations: (a) it is treated as an extended opcode bit, (b) the instruction semantics support an operand size promotion to 64-bit of a general-purpose register operand or a 32-bit memory operand. The presence of W1 in the opcode column implies the opcode must be encoded using the 3-byte form of the VEX prefix. The presence of W0 in the opcode column does not preclude the opcode to be encoded using the C5H form of the VEX prefix, if the semantics of the opcode does not require other VEX subfields not present in the two-byte form of the VEX prefix. Please see Section 2.3 on the subfield definitions within VEX.
- **WIG:** can use C5H form (if not requiring VEX.mmmmm) or VEX.W value is ignored in the C4H form of VEX prefix.
- If WIG is present, the instruction may be encoded using either the two-byte form or the three-byte form of VEX. When encoding the instruction using the three-byte form of VEX, the value of VEX.W is ignored.
- **opcode** — Instruction opcode.
- **/is4** — An 8-bit immediate byte is present containing a source register specifier in either imm8[7:4] (for 64-bit mode) or imm8[6:4] (for 32-bit mode), and instruction-specific payload in imm8[3:0].
- In general, the encoding of VEX.R, VEX.X, VEX.B field are not shown explicitly in the opcode column. The encoding scheme of VEX.R, VEX.X, VEX.B fields must follow the rules defined in Section 2.3.

EVEX.[128,256,512,LLIG].[66,F2,F3].0F/0F3A/0F38.[W0,W1,WIG] opcode [/r] [/ib]

- **EVEX** — The EVEX prefix is encoded using the four-byte form (the first byte is 62H). Refer to Section 2.7.1 for more detail on the EVEX prefix.

The encoding of various sub-fields of the EVEX prefix is described using the following notations:

- **128, 256, 512, LLIG:** This corresponds to the vector length; three values are allowed by EVEX: 512-bit, 256-bit and 128-bit. Alternatively, vector length is ignored (LIG) for certain instructions; this typically applies to scalar instructions operating on one data element of a vector register.
- **66,F2,F3:** The presence of these value maps to the EVEX.pp field encodings. The corresponding VEX.pp value affects the “opcode” byte in the same way as if a SIMD prefix (66H, F2H or F3H) does to the ensuing opcode byte. Thus a non-zero encoding of VEX.pp may be considered as an implied 66H/F2H/F3H prefix.
- **0F,0F3A,0F38:** The presence maps to a valid encoding of the EVEX.mmm field. Only three encoded values of EVEX.mmm are defined as valid, corresponding to the escape byte sequence of 0FH, 0F3AH, and 0F38H. The effect of a valid EVEX.mmm encoding on the ensuing opcode byte is the same as if the corresponding escape byte sequence on the ensuing opcode byte for non-EVEX encoded instructions. Thus a valid encoding of EVEX.mmm may be considered as an implied escape byte sequence of either 0FH, 0F3AH or 0F38H.
- **W0:** EVEX.W=0.

- **W1:** EVEX.W=1.
- **WIG:** EVEX.W bit ignored
- **opcode** — Instruction opcode.
- In general, the encoding of EVEX.R and R', EVEX.X and X', and EVEX.B and B' fields are not shown explicitly in the opcode column.

NOTE

Previously, the terms NDS, NDD, and DDS were used in instructions with an EVEX (or VEX) prefix. These terms indicated that the vvvv field was valid for encoding, and specified register usage. These terms are no longer necessary and are redundant with the instruction operand encoding tables provided with each instruction. The instruction operand encoding tables give explicit details on all operands, indicating where every operand is stored and if they are read or written. If vvvv is not listed as an operand in the instruction operand encoding table, then EVEX (or VEX) vvvv must be 0b1111.

3.1.1.3 Instruction Column in the Opcode Summary Table

The “Instruction” column gives the syntax of the instruction statement as it would appear in an ASM386 program. The following is a list of the symbols used to represent operands in the instruction statements:

- **rel8** — A relative address in the range from 128 bytes before the end of the instruction to 127 bytes after the end of the instruction.
- **rel16, rel32** — A relative address within the same code segment as the instruction assembled. The rel16 symbol applies to instructions with an operand-size attribute of 16 bits; the rel32 symbol applies to instructions with an operand-size attribute of 32 bits.
- **ptr16:16, ptr16:32** — A far pointer, typically to a code segment different from that of the instruction. The notation 16:16 indicates that the value of the pointer has two parts. The value to the left of the colon is a 16-bit selector or value destined for the code segment register. The value to the right corresponds to the offset within the destination segment. The ptr16:16 symbol is used when the instruction's operand-size attribute is 16 bits; the ptr16:32 symbol is used when the operand-size attribute is 32 bits.
- **r8** — One of the byte general-purpose registers: AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL, and SIL; or one of the byte registers (R8B - R15B) available when using REX.R and 64-bit mode.
- **r16** — One of the word general-purpose registers: AX, CX, DX, BX, SP, BP, SI, DI; or one of the word registers (R8-R15) available when using REX.R and 64-bit mode.
- **r32** — One of the doubleword general-purpose registers: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI; or one of the doubleword registers (R8D - R15D) available when using REX.R in 64-bit mode.
- **r64** — One of the quadword general-purpose registers: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15. These are available when using REX.R and 64-bit mode.
- **imm8** — An immediate byte value. The imm8 symbol can be a signed number between -128 and +127 inclusive; an unsigned number between 0 and 255 inclusive; or a bitmap when an instruction uses its individual bits. For instructions in which imm8 is combined with a word or doubleword operand, the immediate value is sign-extended to form a word or doubleword. The upper byte of the word is filled with the topmost bit of the immediate value.
- **imm16** — An immediate word value used for instructions whose operand-size attribute is 16 bits. This is a number between -32,768 and +32,767 inclusive.
- **imm32** — An immediate doubleword value used for instructions whose operand-size attribute is 32 bits. It allows the use of a number between +2,147,483,647 and -2,147,483,648 inclusive.
- **imm64** — An immediate quadword value used for instructions whose operand-size attribute is 64 bits. The value allows the use of a number between +9,223,372,036,854,775,807 and -9,223,372,036,854,775,808 inclusive.
- **/ib** — A single-byte value.

- **r/m8** — A byte operand that is either the contents of a byte general-purpose register (AL, CL, DL, BL, AH, CH, DH, BH, BPL, SPL, DIL, and SIL) or a byte from memory. Byte registers R8B - R15B are available using REX.R in 64-bit mode.
- **r/m16** — A word general-purpose register or memory operand used for instructions whose operand-size attribute is 16 bits. The word general-purpose registers are: AX, CX, DX, BX, SP, BP, SI, DI. The contents of memory are found at the address provided by the effective address computation. Word registers R8W - R15W are available using REX.R in 64-bit mode.
- **r/m32** — A doubleword general-purpose register or memory operand used for instructions whose operand-size attribute is 32 bits. The doubleword general-purpose registers are: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI. The contents of memory are found at the address provided by the effective address computation. Doubleword registers R8D - R15D are available when using REX.R in 64-bit mode.
- **r/m64** — A quadword general-purpose register or memory operand used for instructions whose operand-size attribute is 64 bits when using REX.W. Quadword general-purpose registers are: RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8-R15; these are available only in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **reg** — A general-purpose register used for instructions when the width of the register does not matter to the semantics of the operation of the instruction. The register can be r16, r32, or r64.
- **m** — A 16-, 32- or 64-bit operand in memory.
- **m8** — A byte operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. In 64-bit mode, it is pointed to by the RSI or RDI registers.
- **m16** — A word operand in memory, usually expressed as a variable or array name, but pointed to by the DS:(E)SI or ES:(E)DI registers. This nomenclature is used only with the string instructions.
- **m32** — A doubleword operand in memory. The contents of memory are found at the address provided by the effective address computation.
- **m64** — A memory quadword operand in memory.
- **m128** — A memory double quadword operand in memory.
- **m16:16, m16:32 & m16:64** — A memory operand containing a far pointer composed of two numbers. The number to the left of the colon corresponds to the pointer's segment selector. The number to the right corresponds to its offset.
- **m16&32, m16&16, m32&32, m16&64** — A memory operand consisting of data item pairs whose sizes are indicated on the left and the right side of the ampersand. All memory addressing modes are allowed. The m16&16 and m32&32 operands are used by the BOUND instruction to provide an operand containing an upper and lower bounds for array indices. The m16&32 operand is used by LIDT and LGDT to provide a word with which to load the limit field, and a doubleword with which to load the base field of the corresponding GDTR and IDTR registers. The m16&64 operand is used by LIDT and LGDT in 64-bit mode to provide a word with which to load the limit field, and a quadword with which to load the base field of the corresponding GDTR and IDTR registers.
- **m80bcd** — A Binary Coded Decimal (BCD) operand in memory, 80 bits.
- **mooffs8, mooffs16, mooffs32, mooffs64** — A simple memory variable (memory offset) of type byte, word, or doubleword used by some variants of the MOV instruction. The actual address is given by a simple offset relative to the segment base. No ModR/M byte is used in the instruction. The number shown with mooffs indicates its size, which is determined by the address-size attribute of the instruction.
- **Sreg** — A segment register. The segment register bit assignments are ES = 0, CS = 1, SS = 2, DS = 3, FS = 4, and GS = 5.
- **m32fp, m64fp, m80fp** — A single precision, double precision, and double extended-precision (respectively) floating-point operand in memory. These symbols designate floating-point values that are used as operands for x87 FPU floating-point instructions.
- **m16int, m32int, m64int** — A word, doubleword, and quadword integer (respectively) operand in memory. These symbols designate integers that are used as operands for x87 FPU integer instructions.
- **ST or ST(0)** — The top element of the FPU register stack.
- **ST(i)** — The i^{th} element from the top of the FPU register stack ($i := 0$ through 7).
- **mm** — An MMX register. The 64-bit MMX registers are: MM0 through MM7.

- **mm/m32** — The low order 32 bits of an MMX register or a 32-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- **mm/m64** — An MMX register or a 64-bit memory operand. The 64-bit MMX registers are: MM0 through MM7. The contents of memory are found at the address provided by the effective address computation.
- **xmm** — An XMM register. The 128-bit XMM registers are: XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode.
- **xmm/m32** — An XMM register or a 32-bit memory operand. The 128-bit XMM registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **xmm/m64** — An XMM register or a 64-bit memory operand. The 128-bit SIMD floating-point registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **xmm/m128** — An XMM register or a 128-bit memory operand. The 128-bit XMM registers are XMM0 through XMM7; XMM8 through XMM15 are available using REX.R in 64-bit mode. The contents of memory are found at the address provided by the effective address computation.
- **<XMM0>** — Indicates implied use of the XMM0 register.
 When there is ambiguity, xmm1 indicates the first source operand using an XMM register and xmm2 the second source operand using an XMM register.
 Some instructions use the XMM0 register as the third source operand, indicated by <XMM0>. The use of the third XMM register operand is implicit in the instruction encoding and does not affect the ModR/M encoding.
- **ymm** — A YMM register. The 256-bit YMM registers are: YMM0 through YMM7; YMM8 through YMM15 are available in 64-bit mode.
- **m256** — A 32-byte operand in memory. This nomenclature is used only with AVX instructions.
- **ymm/m256** — A YMM register or 256-bit memory operand.
- **<YMM0>** — Indicates use of the YMM0 register as an implicit argument.
- **bnd** — A 128-bit bounds register. BND0 through BND3.
- **mib** — A memory operand using SIB addressing form, where the index register is not used in address calculation, Scale is ignored. Only the base and displacement are used in effective address calculation.
- **m512** — A 64-byte operand in memory.
- **zmm/m512** — A ZMM register or 512-bit memory operand.
- **{k1}{z}** — A mask register used as instruction writemask. The 64-bit k registers are: k1 through k7. Writemask specification is available exclusively via EVEX prefix. The masking can either be done as a merging-masking, where the old values are preserved for masked out elements or as a zeroing masking. The type of masking is determined by using the EVEX.z bit.
- **{k1}** — Without {z}: a mask register used as instruction writemask for instructions that do not allow zeroing-masking but support merging-masking. This corresponds to instructions that require the value of the aaa field to be different than 0 (e.g., gather) and store-type instructions which allow only merging-masking.
- **k1** — A mask register used as a regular operand (either destination or source). The 64-bit k registers are: k0 through k7.
- **mV** — A vector memory operand; the operand size is dependent on the instruction.
- **vm32{x,y,z}** — A vector array of memory operands specified using VSIB memory addressing. The array of memory addresses are specified using a common base register, a constant scale factor, and a vector index register with individual elements of 32-bit index value in an XMM register (vm32x), a YMM register (vm32y) or a ZMM register (vm32z).
- **vm64{x,y,z}** — A vector array of memory operands specified using VSIB memory addressing. The array of memory addresses are specified using a common base register, a constant scale factor, and a vector index register with individual elements of 64-bit index value in an XMM register (vm64x), a YMM register (vm64y) or a ZMM register (vm64z).

- **zmm/m512/m32bcst** — An operand that can be a ZMM register, a 512-bit memory location or a 512-bit vector loaded from a 32-bit memory location.
- **zmm/m512/m64bcst** — An operand that can be a ZMM register, a 512-bit memory location or a 512-bit vector loaded from a 64-bit memory location.
- **<ZMM0>** — Indicates use of the ZMM0 register as an implicit argument.
- **{er}** — Indicates support for embedded rounding control, which is only applicable to the register-register form of the instruction. This also implies support for SAE (Suppress All Exceptions).
- **{sae}** — Indicates support for SAE (Suppress All Exceptions). This is used for instructions that support SAE, but do not support embedded rounding control.
- **SRC1** — Denotes the first source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having two or more source operands.
- **SRC2** — Denotes the second source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having two or more source operands.
- **SRC3** — Denotes the third source operand in the instruction syntax of an instruction encoded with the VEX/EVEX prefix and having three source operands.
- **SRC** — The source in a single-source instruction.
- **DST** — The destination in an instruction. This field is encoded by reg_field.

In the instruction encoding, the MODRM byte is represented several ways depending on the role it plays. The MODRM byte has 3 fields: 2-bit MODRM.MOD field, a 3-bit MODRM.REG field and a 3-bit MODRM.RM field. When all bits of the MODRM byte have fixed values for an instruction, the 2-hex nibble value of that byte is presented after the opcode in the encoding boxes on the instruction description pages. When only some fields of the MODRM byte must contain fixed values, those values are specified as follows:

- If only the MODRM.MOD must be 0b11, and MODRM.REG and MODRM.RM fields are unrestricted, this is denoted as **11:rrr:bbb**. The **rrr** correspond to the 3-bits of the MODRM.REG field and the **bbb** correspond to the 3-bits of the MODRM.RM field.
- If the MODRM.MOD field is constrained to be a value other than 0b11, i.e., it must be one of 0b00, 0b01, or 0b10, then we use the notation **!(11)**.
- If the MODRM.REG field had a specific required value, e.g., 0b101, that would be denoted as **mm:101:bbb**.

3.1.1.4 Operand Encoding Column in the Instruction Summary Table

The “operand encoding” column is abbreviated as Op/En in the Instruction Summary table heading. Instruction operand encoding information is provided for each assembly instruction syntax using a letter to cross reference to a row entry in the operand encoding definition table that follows the instruction summary table. The operand encoding table in each instruction reference page lists each instruction operand (according to each instruction syntax and operand ordering shown in the instruction column) relative to the ModRM byte, VEX.vvvv field or additional operand encoding placement.

EVEX encoded instructions employ compressed $\text{disp8} \times N$ encoding of the displacement bytes, where N is defined in Table 2-36 and Table 2-37, according to tuple types. The tuple type for an instruction is listed in the operand encoding definition table where applicable.

NOTES

- The letters in the Op/En column of an instruction apply ONLY to the encoding definition table immediately following the instruction summary table.
- In the encoding definition table, the letter ‘r’ within a pair of parenthesis denotes the content of the operand will be read by the processor. The letter ‘w’ within a pair of parenthesis denotes the content of the operand will be updated by the processor.

3.1.1.5 64/32-bit Mode Column in the Instruction Summary Table

The “64/32-bit Mode” column indicates whether the opcode sequence is supported in (a) 64-bit mode or (b) the Compatibility mode and other IA-32 modes that apply in conjunction with the CPUID feature flag associated specific instruction extensions.

The 64-bit mode support is to the left of the 'slash' and has the following notation:

- **V** — Supported.
- **I** — Not supported.
- **N.E.** — Indicates an instruction syntax is not encodable in 64-bit mode (it may represent part of a sequence of valid instructions in other modes).
- **N.P.** — Indicates the REX prefix does not affect the legacy instruction in 64-bit mode.
- **N.I.** — Indicates the opcode is treated as a new instruction in 64-bit mode.
- **N.S.** — Indicates an instruction syntax that requires an address override prefix in 64-bit mode and is not supported. Using an address override prefix in 64-bit mode may result in model-specific execution behavior.

The Compatibility/Legacy Mode support is to the right of the 'slash' and has the following notation:

- **V** — Supported.
- **I** — Not supported.
- **N.E.** — Indicates an Intel 64 instruction mnemonics/syntax that is not encodable; the opcode sequence is not applicable as an individual instruction in compatibility mode or IA-32 mode. The opcode may represent a valid sequence of legacy IA-32 instructions.

3.1.1.6 CPUID Support Column in the Instruction Summary Table

The fourth column holds abbreviated CPUID feature flags (e.g., appropriate bit in CPUID.01H:ECX, CPUID.01H:EDX for SSE/SSE2/SSE3/SSSE3/SSE4.1/SSE4.2/AESNI/PCLMULQDQ/AVX/RDRAND support) that indicate processor support for the instruction. If the corresponding flag is '0', the instruction will #UD.

3.1.1.7 Description Column in the Instruction Summary Table

The "Description" column briefly explains forms of the instruction.

3.1.1.8 Description Section

Each instruction is then described by number of information sections. The "Description" section describes the purpose of the instructions and required operands in more detail.

Summary of terms that may be used in the description section:

- **Legacy SSE** — Refers to SSE, SSE2, SSE3, SSSE3, SSE4, AESNI, PCLMULQDQ, and any future instruction sets referencing XMM registers and encoded without a VEX prefix.
- **VEX.vvvv** — The VEX bit field specifying a source or destination register (in 1's complement form).
- **rm_field** — shorthand for the ModR/M *r/m* field and any REX.B.
- **reg_field** — shorthand for the ModR/M *reg* field and any REX.R.

3.1.1.9 Operation Section

The "Operation" section contains an algorithm description (frequently written in pseudo-code) for the instruction. Algorithms are composed of the following elements:

- Comments are enclosed within the symbol pairs "(" and ")".
- Compound statements are enclosed in keywords, such as: IF, THEN, ELSE, and FI for an if statement; DO and OD for a do statement; or CASE... OF for a case statement.
- A register name implies the contents of the register. A register name enclosed in brackets implies the contents of the location whose address is contained in that register. For example, ES:[DI] indicates the contents of the location whose ES segment relative address is in register DI. [SI] indicates the contents of the address contained in register SI relative to the SI register's default segment (DS) or the overridden segment.
- Parentheses around the "E" in a general-purpose register name, such as (E)SI, indicates that the offset is read from the SI register if the address-size attribute is 16, from the ESI register if the address-size attribute is 32.

Parentheses around the “R” in a general-purpose register name, (R)SI, in the presence of a 64-bit register definition such as (R)SI, indicates that the offset is read from the 64-bit RSI register if the address-size attribute is 64.

- Brackets are used for memory operands where they mean that the contents of the memory location is a segment-relative offset. For example, [SRC] indicates that the content of the source operand is a segment-relative offset.
- $A := B$ indicates that the value of B is assigned to A.
- The symbols $=$, $\neq$, $>$, $<$, $\geq$, and $\leq$ are relational operators used to compare two values: meaning equal, not equal, greater or equal, less or equal, respectively. A relational expression such as $A = B$ is TRUE if the value of A is equal to B; otherwise it is FALSE.
- The expression “ $\ll$ COUNT” and “ $\gg$ COUNT” indicates that the destination operand should be shifted left or right by the number of bits indicated by the count operand.

The following identifiers are used in the algorithmic descriptions:

- **OperandSize and AddressSize** — The OperandSize identifier represents the operand-size attribute of the instruction, which is 16, 32 or 64-bits. The AddressSize identifier represents the address-size attribute, which is 16, 32 or 64-bits. For example, the following pseudo-code indicates that the operand-size attribute depends on the form of the MOV instruction used.

```
IF Instruction = MOVW
    THEN OperandSize := 16;
ELSE
    IF Instruction = MOVD
        THEN OperandSize := 32;
    ELSE
        IF Instruction = MOVQ
            THEN OperandSize := 64;
        FI;
    FI;
FI;
```

See “Operand-Size and Address-Size Attributes” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for guidelines on how these attributes are determined.

- **StackAddrSize** — Represents the stack address-size attribute associated with the instruction, which has a value of 16, 32 or 64-bits. See “Address-Size Attribute for Stack” in Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.
- **SRC** — Represents the source operand.
- **DEST** — Represents the destination operand.
- **MAXVL** — The maximum vector register width pertaining to the instruction. This is not the vector-length encoding in the instruction’s encoding but is instead determined by the current value of XCR0. For details, refer to the table below. Note that the value of MAXVL is the largest of the features enabled. Future processors may define new bits in XCR0 whose setting may imply other values for MAXVL.

MAXVL Definition

XCR0 Component	MAXVL
XCR0.SSE	128
XCR0.AVX	256
XCR0.{ZMM_Hi256, Hi16_ZMM, OPMASK}	512

The following functions are used in the algorithmic descriptions:

- **ZeroExtend(value)** — Returns a value zero-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, zero extending a byte value of -10 converts the byte from F6H to

a doubleword value of 000000F6H. If the value passed to the ZeroExtend function and the operand-size attribute are the same size, ZeroExtend returns the value unaltered.

- **SignExtend(value)** — Returns a value sign-extended to the operand-size attribute of the instruction. For example, if the operand-size attribute is 32, sign extending a byte containing the value -10 converts the byte from F6H to a doubleword value of FFFFFFF6H. If the value passed to the SignExtend function and the operand-size attribute are the same size, SignExtend returns the value unaltered.
- **SaturateSignedWordToSignedByte** — Converts a signed 16-bit value to a signed 8-bit value. If the signed 16-bit value is less than -128, it is represented by the saturated value -128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateSignedDwordToSignedWord** — Converts a signed 32-bit value to a signed 16-bit value. If the signed 32-bit value is less than -32768, it is represented by the saturated value -32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateSignedWordToUnsignedByte** — Converts a signed 16-bit value to an unsigned 8-bit value. If the signed 16-bit value is less than zero, it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToSignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than -128, it is represented by the saturated value -128 (80H); if it is greater than 127, it is represented by the saturated value 127 (7FH).
- **SaturateToSignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than -32768, it is represented by the saturated value -32768 (8000H); if it is greater than 32767, it is represented by the saturated value 32767 (7FFFH).
- **SaturateToUnsignedByte** — Represents the result of an operation as a signed 8-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 255, it is represented by the saturated value 255 (FFH).
- **SaturateToUnsignedWord** — Represents the result of an operation as a signed 16-bit value. If the result is less than zero it is represented by the saturated value zero (00H); if it is greater than 65535, it is represented by the saturated value 65535 (FFFFH).
- **LowOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the least significant word of the doubleword result in the destination operand.
- **HighOrderWord(DEST * SRC)** — Multiplies a word operand by a word operand and stores the most significant word of the doubleword result in the destination operand.
- **Push(value)** — Pushes a value onto the stack. The number of bytes pushed is determined by the operand-size attribute of the instruction. See the “Operation” subsection of the “PUSH—Push Word, Doubleword, or Quadword Onto the Stack” section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.
- **Pop()** — removes the value from the top of the stack and returns it. The statement `EAX := Pop();` assigns to EAX the 32-bit value from the top of the stack. Pop will return either a word, a doubleword or a quadword depending on the operand-size attribute. See the “Operation” subsection in the “POP—Pop a Value From the Stack” section of Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B.
- **PopRegisterStack** — Marks the FPU ST(0) register as empty and increments the FPU register stack pointer (TOP) by 1.
- **Switch-Tasks** — Performs a task switch.
- **Bit(BitBase, BitOffset)** — Returns the value of a bit within a bit string. The bit string is a sequence of bits in memory or a register. Bits are numbered from low-order to high-order within registers and within memory bytes. If the BitBase is a register, the BitOffset can be in the range 0 to [15, 31, 63] depending on the mode and register size. See Figure 3-1: the function `Bit[RAX, 21]` is illustrated.

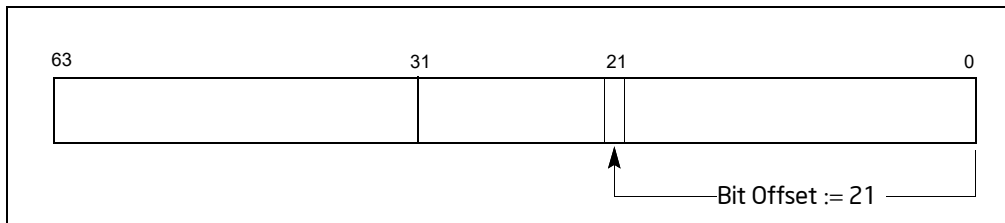


Figure 3-1. Bit Offset for BIT[RAX, 21]

If BitBase is a memory address, the BitOffset has different ranges depending on the operand size (see Table 3-2).

Table 3-2. Range of Bit Positions Specified by Bit Offset Operands

Operand Size	Immediate BitOffset	Register BitOffset
16	0 to 15	-2^{15} to $2^{15} - 1$
32	0 to 31	-2^{31} to $2^{31} - 1$
64	0 to 63	-2^{63} to $2^{63} - 1$

The addressed bit is numbered (Offset MOD 8) within the byte at address (BitBase + (BitOffset DIV 8)) where DIV is signed division with rounding towards negative infinity and MOD returns a positive number (see Figure 3-2).

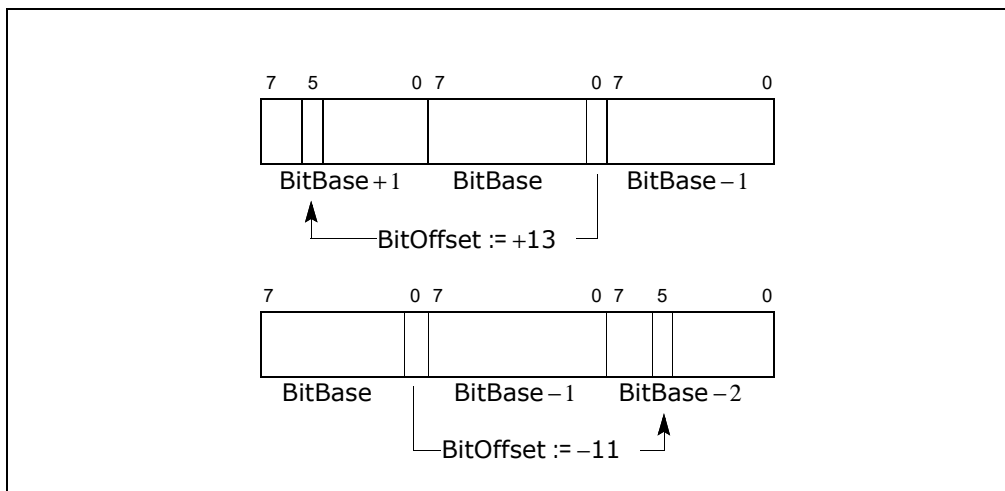


Figure 3-2. Memory Bit Indexing

3.1.1.10 Intel® C/C++ Compiler Intrinsics Equivalents Section

The Intel C/C++ compiler intrinsic functions give access to the full power of the Intel Architecture Instruction Set, while allowing the compiler to optimize register allocation and instruction scheduling for faster execution. Most of these functions are associated with a single IA instruction, although some may generate multiple instructions or different instructions depending upon how they are used. In particular, these functions are used to invoke instructions that perform operations on vector registers that can hold multiple data elements. These SIMD instructions use the following data types.

- `__m128`, `__m256`, and `__m512` can represent 4, 8, or 16 packed single precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The `__m128` data type is also used with various single precision floating-point scalar instructions that perform calculations using

only the lowest 32 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.

- `__m128d`, `__m256d`, and `__m512d` can represent 2, 4, or 8 packed double precision floating-point values, and are used with the vector registers and SSE, AVX, or AVX-512 instruction set extension families. The `__m128d` data type is also used with various double precision floating-point scalar instructions that perform calculations using only the lowest 64 bits of a vector register; the remaining bits of the result come from one of the sources or are set to zero depending upon the instruction.
- `__m128i`, `__m256i`, and `__m512i` can represent integer data in bytes, words, doublewords, quadwords, and occasionally larger data types.

Each of these data types incorporates in its name the number of bits it can hold. For example, the `__m128` type holds 128 bits, and because each single precision floating-point value is 32 bits long the `__m128` type holds (128/32) or four values. Normally the compiler will allocate memory for these data types on an even multiple of the size of the type. Such aligned memory locations may be faster to read and write than locations at other addresses.

These SIMD data types are not basic Standard C data types or C++ objects, so they may be used only with the assignment operator, passed as function arguments, and returned from a function call. If you access the internal members of these types directly, or indirectly by using them in a union, there may be side effects affecting optimization, so it is recommended to use them only with the SIMD instruction intrinsic functions described in this manual or the Intel C/C++ compiler documentation.

Many intrinsic functions names are prefixed with an indicator of the vector length and suffixed by an indicator of the vector element data type, although some functions do not follow the rules below. The prefixes are:

- `_mm_` indicates that the function operates on 128-bit (or sometimes 64-bit) vectors.
- `_mm256_` indicates the function operates on 256-bit vectors.
- `_mm512_` indicates that the function operates on 512-bit vectors.

The suffixes include:

- `_ps`, which indicates a function that operates on packed single precision floating-point data. Packed single precision floating-point data corresponds to arrays of the C/C++ type *float* with either 4, 8 or 16 elements. Values of this type can be loaded from an array using the `_mm_loadu_ps`, `_mm256_loadu_ps`, or `_mm512_loadu_ps` functions, or created from individual values using `_mm_set_ps`, `_mm256_set_ps`, or `_mm512_set_ps` functions, and they can be stored in an array using `_mm_storeu_ps`, `_mm256_storeu_ps`, or `_mm512_storeu_ps`.
- `_ss`, which indicates a function that operates on scalar single precision floating-point data. Single precision floating-point data corresponds to the C/C++ type *float*, and values of type *float* can be converted to type `__m128` for use with these functions using the `_mm_set_ss` function, and converted back using the `_mm_cvtss_f32` function. When used with functions that operate on packed single precision floating-point data the scalar element corresponds with the first packed value.
- `_pd`, which indicates a function that operates on packed double precision floating-point data. Packed double precision floating-point data corresponds to arrays of the C/C++ type *double* with either 2, 4, or 8 elements. Values of this type can be loaded from an array using the `_mm_loadu_pd`, `_mm256_loadu_pd`, or `_mm512_loadu_pd` functions, or created from individual values using `_mm_set_pd`, `_mm256_set_pd`, or `_mm512_set_pd` functions, and they can be stored in an array using `_mm_storeu_pd`, `_mm256_storeu_pd`, or `_mm512_storeu_pd`.
- `_sd`, which indicates a function that operates on scalar double precision floating-point data. Double precision floating-point data corresponds to the C/C++ type *double*, and values of type *double* can be converted to type `__m128d` for use with these functions using the `_mm_set_sd` function, and converted back using the `_mm_cvtsd_f64` function. When used with functions that operate on packed double precision floating-point data the scalar element corresponds with the first packed value.
- `_epi8`, which indicates a function that operates on packed 8-bit signed integer values. Packed 8-bit signed integers correspond to an array of *signed char* with 16, 32 or 64 elements. Values of this type can be created from individual elements using `_mm_set_epi8`, `_mm256_set_epi8`, or `_mm512_set_epi8` functions.
- `_epi16`, which indicates a function that operates on packed 16-bit signed integer values. Packed 16-bit signed integers correspond to an array of *short* with 8, 16 or 32 elements. Values of this type can be created from individual elements using `_mm_set_epi16`, `_mm256_set_epi16`, or `_mm512_set_epi16` functions.

- `_epi32`, which indicates a function that operates on packed 32-bit signed integer values. Packed 32-bit signed integers correspond to an array of *int* with 4, 8 or 16 elements. Values of this type can be created from individual elements using `_mm_set_epi32`, `_mm256_set_epi32`, or `_mm512_set_epi32` functions.
- `_epi64`, which indicates a function that operates on packed 64-bit signed integer values. Packed 64-bit signed integers correspond to an array of *long long* (or *long* if it is a 64-bit data type) with 2, 4 or 8 elements. Values of this type can be created from individual elements using `_mm_set_epi32`, `_mm256_set_epi32`, or `_mm512_set_epi32` functions.
- `_epu8`, which indicates a function that operates on packed 8-bit unsigned integer values. Packed 8-bit unsigned integers correspond to an array of *unsigned char* with 16, 32 or 64 elements.
- `_epu16`, which indicates a function that operates on packed 16-bit unsigned integer values. Packed 16-bit unsigned integers correspond to an array of *unsigned short* with 8, 16 or 32 elements.
- `_epu32`, which indicates a function that operates on packed 32-bit unsigned integer values. Packed 32-bit unsigned integers correspond to an array of *unsigned* with 4, 8 or 16 elements.
- `_epu64`, which indicates a function that operates on packed 64-bit unsigned integer values. Packed 64-bit unsigned integers correspond to an array of *unsigned long long* (or *unsigned long* if it is a 64-bit data type) with 2, 4 or 8 elements.
- `_si128`, which indicates a function that operates on a single 128-bit value of type `__m128i`.
- `_si256`, which indicates a function that operates on a single a 256-bit value of type `__m256i`.
- `_si512`, which indicates a function that operates on a single a 512-bit value of type `__m512i`.

Values of any packed integer type can be loaded from an array using the `_mm_loadu_si128`, `_mm256_loadu_si256`, or `_mm512_loadu_si512` functions, and they can be stored in an array using `_mm_storeu_si128`, `_mm256_storeu_si256`, or `_mm512_storeu_si512`.

These functions and data types are used with the SSE, AVX, and AVX-512 instruction set extension families. In addition there are similar functions that correspond to MMX instructions. These are less frequently used because they require additional state management, and only operate on 64-bit packed integer values.

The declarations of Intel C/C++ compiler intrinsic functions may reference some non-standard data types, such as `__int64`. The C Standard header `stdint.h` defines similar platform-independent types, and the documentation for that header gives characteristics that apply to corresponding non-standard types according to the following table.

Table 3-3. Standard and Non-Standard Data Types

Non-standard Type	Standard Type (from <code>stdint.h</code>)
<code>__int64</code>	<code>int64_t</code>
<code>unsigned __int64</code>	<code>uint64_t</code>
<code>__int32</code>	<code>int32_t</code>
<code>unsigned __int32</code>	<code>uint32_t</code>
<code>__int16</code>	<code>int16_t</code>
<code>unsigned __int16</code>	<code>uint16_t</code>

For a more detailed description of each intrinsic function and additional information related to its usage, refer to the online Intel Intrinsics Guide, <https://software.intel.com/sites/landingpage/IntrinsicsGuide>.

3.1.1.11 Flags Affected Section

The “Flags Affected” section lists the flags in the EFLAGS register that are affected by the instruction. When a flag is cleared, it is equal to 0; when it is set, it is equal to 1. The arithmetic and logical instructions usually assign values to the status flags in a uniform manner (see Appendix A, “EFLAGS Cross-Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). Non-conventional assignments are described in the “Operation” section. The values of flags listed as **undefined** may be changed by the instruction in an indeterminate manner. Flags that are not listed are unchanged by the instruction.

3.1.1.12 FPU Flags Affected Section

The floating-point instructions have an “FPU Flags Affected” section that describes how each instruction can affect the four condition code flags of the FPU status word.

3.1.1.13 Protected Mode Exceptions Section

The “Protected Mode Exceptions” section lists the exceptions that can occur when the instruction is executed in protected mode and the reasons for the exceptions. Each exception is given a mnemonic that consists of a pound sign (#) followed by two letters and an optional error code in parentheses. For example, #GP(0) denotes a general protection exception with an error code of 0. Table 3-4 associates each two-letter mnemonic with the corresponding exception vector and name. See Chapter 6, “Procedure Calls, Interrupts, and Exceptions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for a detailed description of the exceptions.

Application programmers should consult the documentation provided with their operating systems to determine the actions taken when exceptions occur.

Table 3-4. Intel 64® and IA-32 General Exceptions

Vector	Name	Source	Protected Mode ¹	Real Address Mode	Virtual 8086 Mode
0	#DE—Divide Error	DIV and IDIV instructions.	Yes	Yes	Yes
1	#DB—Debug	Any code or data reference.	Yes	Yes	Yes
3	#BP—Breakpoint	INT3 instruction.	Yes	Yes	Yes
4	#OF—Overflow	INTO instruction.	Yes	Yes	Yes
5	#BR—BOUND Range Exceeded	BOUND instruction.	Yes	Yes	Yes
6	#UD—Invalid Opcode (Undefined Opcode)	UD instruction or reserved opcode.	Yes	Yes	Yes
7	#NM—Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes
8	#DF—Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.	Yes	Yes	Yes
10	#TS—Invalid TSS	Task switch or TSS access.	Yes	No	Yes
11	#NP—Segment Not Present	Loading segment registers or accessing system segments.	Yes	No	Yes
12	#SS—Stack Segment Fault	Stack operations and SS register loads.	Yes	Yes	Yes
13	#GP—General Protection ²	Any memory reference and other protection checks.	Yes	Yes	Yes
14	#PF—Page Fault	Any memory reference.	Yes	No	Yes
16	#MF—Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.	Yes	Yes	Yes
17	#AC—Alignment Check	Any data reference in memory.	Yes	No	Yes
18	#MC—Machine Check	Model dependent machine check errors.	Yes	Yes	Yes
19	#XM—SIMD Floating-Point Numeric Error	SSE/SSE2/SSE3 floating-point instructions.	Yes	Yes	Yes
20	#VE—Virtualization Exception	EPT violations ³	Yes	No	No

Table 3-4. Intel 64® and IA-32 General Exceptions (Contd.)

Vector	Name	Source	Protected Mode ¹	Real Address Mode	Virtual 8086 Mode
21	#CP—Control Protection Exception	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.	Yes	No	No

NOTES:

1. Apply to protected mode, compatibility mode, and 64-bit mode.
2. In the real-address mode, vector 13 is the segment overrun exception.
3. This exception can occur only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.

3.1.1.14 Real-Address Mode Exceptions Section

The “Real-Address Mode Exceptions” section lists the exceptions that can occur when the instruction is executed in real-address mode (see Table 3-4).

3.1.1.15 Virtual-8086 Mode Exceptions Section

The “Virtual-8086 Mode Exceptions” section lists the exceptions that can occur when the instruction is executed in virtual-8086 mode (see Table 3-4).

3.1.1.16 Floating-Point Exceptions Section

The “Floating-Point Exceptions” section lists exceptions that can occur when an x87 FPU floating-point instruction is executed. All of these exception conditions result in a floating-point error exception (#MF, exception 16) being generated. Table 3-5 associates a one- or two-letter mnemonic with the corresponding exception name. See “Floating-Point Exception Conditions” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for a detailed description of these exceptions.

Table 3-5. x87 FPU Floating-Point Exceptions

Mnemonic	Name	Source
#IS #IA	Floating-point invalid operation: - Stack overflow or underflow - Invalid arithmetic operation	- x87 FPU stack overflow or underflow - Invalid FPU arithmetic operation
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result (precision)	Inexact result (precision)

3.1.1.17 SIMD Floating-Point Exceptions Section

The “SIMD Floating-Point Exceptions” section lists exceptions that can occur when an SSE/SSE2/SSE3 floating-point instruction is executed. All of these exception conditions result in a SIMD floating-point error exception (#XM, exception 19) being generated. Table 3-6 associates a one-letter mnemonic with the corresponding exception name. For a detailed description of these exceptions, refer to “SSE and SSE2 Exceptions”, in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

Table 3-6. SIMD Floating-Point Exceptions

Mnemonic	Name	Source
#I	Floating-point invalid operation	Invalid arithmetic operation or source operand
#Z	Floating-point divide-by-zero	Divide-by-zero
#D	Floating-point denormal operand	Source operand that is a denormal number
#O	Floating-point numeric overflow	Overflow in result
#U	Floating-point numeric underflow	Underflow in result
#P	Floating-point inexact result	Inexact result (precision)

3.1.1.18 Compatibility Mode Exceptions Section

This section lists exceptions that occur within compatibility mode.

3.1.1.19 64-Bit Mode Exceptions Section

This section lists exceptions that occur within 64-bit mode.

3.2 INTEL® AMX CONSIDERATIONS

The following implementation parameters and helper functions are applicable to the Intel® AMX instructions.

3.2.1 Implementation Parameters

The parameters are reported via CPUID.1DH. Index 0 reports all zeros for all fields.

```
define palette_table[id]:
    uint16_t total_tile_bytes
    uint16_t bytes_per_tile
    uint16_t bytes_per_row
    uint16_t max_names
    uint16_t max_rows
```

The tile parameters are set by LDTILECFG or XRSTOR* of TILECFG:

```
define tile[tid]:
    byte rows
    word colsb // bytes_per_row
    bool valid
```

3.2.2 Helper Functions

The helper functions used in Intel AMX instructions are defined below.

```

define write_row_and_zero(treg, r, data, nbytes):
    for j in 0 ...nbytes-1:
        treg.row[r].byte[j] := data.byte[j]

    // zero the rest of the row
    for j in nbytes ... palette_table[tilecfg.palette_id].bytes_per_row-1:
        treg.row[r].byte[j] := 0

define zero_upper_rows(treg, r):
    for i in r ... palette_table[tilecfg.palette_id].max_rows-1:
        for j in 0 ... palette_table[tilecfg.palette_id].bytes_per_row-1:
            treg.row[i].byte[j] := 0

define zero_tilecfg_start():
    tilecfg.start_row :=0

define zero_all_tile_data():
    if XCR0[TILEDATA]:
        b:=CPUID.0DH.TILEDATA:EAX // size of feature
        for j in 0 ... b:
            TILEDATA.byte[j] := 0

define xcr0_supports_palette(palette_id):
    if palette_id == 0:
        return 1
    elif palette_id == 1:
        if XCR0[TILECFG] and XCR0[TILEDATA]:
            return 1
    return 0

```

3.3 INSTRUCTIONS (A-L)

The remainder of this chapter provides descriptions of Intel 64 and IA-32 instructions (A-L). See also: Chapter 4, “Instruction Set Reference, M-U,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B; Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2C; and Chapter 6, “Instruction Set Reference, W-Z,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

CPUID—CPU Identification

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF A2	CPUID	Z0	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX and, in some cases, ECX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

The ID flag (bit 21) in the EFLAGS register indicates support for the CPUID instruction. If a software procedure can set and clear this flag, the processor executing the procedure supports the CPUID instruction. This instruction operates the same in non-64-bit modes and 64-bit mode.

CPUID returns processor identification and feature information in the EAX, EBX, ECX, and EDX registers.¹ The instruction's output is dependent on the contents of the EAX register upon execution and, in some cases, ECX.

Chapter 21, "Processor Identification and Feature Determination," in Volume 1 of the Intel® 64 and IA-32 Architectures Software Developer's Manual provides CPUID leaf information and shows information returned, depending on the initial value loaded into the EAX and ECX registers.

CPUID can be executed at any privilege level to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed. Although the CPUID instruction provides serialization, it is not the preferred method on newer processors that support the SERIALIZE instruction. See "Serializing Instructions" in Chapter 10 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A for more details.

Execution of CPUID causes a VM exit when executed in VMX non-root operation. See Chapter 27, "VMX Non-Root Operation," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C for more details.

IA-32 Architecture Compatibility

CPUID is not supported in early models of the Intel486 processor or in any IA-32 processor earlier than the Intel486 processor.

Operation

IA32_BIOS_SIGN_ID MSR := Update with installed microcode revision number;

(* Note that for some leaf values in EAX, the subleaf value in ECX is ignored. *)

(* Note that for invalid CPUID leaves and subleaves, the output values returned in EAX, EBX, ECX, and EDX are "Reserved" *)

(* Refer to Volume 1, Chapter 21 for details surrounding CPUID_INFO() *)

(EAX, EBX, ECX, EDX) := CPUID_INFO(EAX, ECX)

Flags Affected

None.

Exceptions (All Operating Modes)

#UD If the LOCK prefix is used.

In earlier IA-32 processors that do not support the CPUID instruction, execution of the instruction results in an invalid opcode (#UD) exception being generated.

1. On Intel 64 processors, CPUID clears the high 32 bits of the RAX/RBX/RCX/RDX registers in all modes.

INVD—Invalidate Internal Caches

Opcode ¹	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 08	INVD	Z0	Valid	Valid	Flush internal caches; initiate flushing of external caches.

NOTES:

1. See the IA-32 Architecture Compatibility section below.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

Invalidates (flushes) the processor's internal caches and issues a special-function bus cycle that directs external caches to also flush themselves. Data held in internal caches is not written back to main memory.

After executing this instruction, the processor does not wait for the external caches to complete their flushing operation before proceeding with instruction execution. It is the responsibility of hardware to respond to the cache flush signal.

The INVD instruction is a privileged instruction. When the processor is running in protected mode, the CPL of a program or procedure must be 0 to execute this instruction.

The INVD instruction may be used when the cache is used as temporary memory and the cache contents need to be invalidated rather than written back to memory. When the cache is used as temporary memory, no external device should be actively writing data to main memory.

Use this instruction with care. Data cached internally and not written back to main memory will be lost. Note that any data from an external device to main memory (for example, via a PCIWrite) can be temporarily stored in the caches; these data can be lost when an INVD instruction is executed. Unless there is a specific requirement or benefit to flushing caches without writing back modified cache lines (for example, temporary memory, testing, or fault recovery where cache coherency with main memory is not a concern), software should instead use the WBINVD instruction.

On processors that support processor reserved memory, the INVD instruction cannot be executed when processor reserved memory protections are activated. See Section 38.5, "EPC and Management of EPC Pages," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D.

On processors that support SEAM, the INVD instruction cannot be executed when the SEAM range is protected. See Section 34.4, "Memory Protection," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3.

Some processors prevent execution of INVD after BIOS execution is complete. They report this by enumerating CPUID.07H.01H:EAX[30] as 1. On such processors, INVD cannot be executed if bit 0 of SR_BIOS_DONE (MSR address 151H) is 1.

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

IA-32 Architecture Compatibility

The INVD instruction is implementation dependent; it may be implemented differently on different families of Intel 64 or IA-32 processors. This instruction is not supported on IA-32 processors earlier than the Intel486 processor.

Operation

Flush(InternalCaches);
SignalFlush(ExternalCaches);
Continue (* Continue execution *)

Flags Affected

None.

Protected Mode Exceptions

- #GP(0) If the current privilege level is not 0.
If the processor reserved memory protections are activated.
If the SEAM range is protected.
If CPUID.07H.01H:EAX[30] = 1 and bit 0 is set in MSR_BIOS_DONE (MSR address 151H).
- #UD If the LOCK prefix is used.

Real-Address Mode Exceptions

- #GP(0) If CPUID.07H.01H:EAX[30] = 1 and bit 0 is set in MSR_BIOS_DONE (MSR address 151H).
If the processor reserved memory protections are activated.
If the SEAM range is protected.
- #UD If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

- #GP(0) The INVD instruction cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

5. Updates to Chapter 4, Volume 2B

Change bars and violet text show changes to Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, M-U*.

Changes to this chapter:

- Updated instructions MOV—Move to/from Control Registers and PCONFIG—Platform Configuration with SEAM information.
- Corrected footnote reference for instruction MOVXSD r32, r/m32 and opcode 63 /r.
- For instruction UD—Undefined Instruction, added opcode D6 and instruction UDB.

4.1 IMM8 CONTROL BYTE OPERATION FOR PCMPESTRI / PCMPESTRM / PCMPISTRI / PCMPISTRM

The notations introduced in this section are referenced in the reference pages of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM. The operation of the immediate control byte is common to these four string text processing instructions of SSE4.2. This section describes the common operations.

4.1.1 General Description

The operation of PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM is defined by the combination of the respective opcode and the interpretation of an immediate control byte that is part of the instruction encoding.

The opcode controls the relationship of input bytes/words to each other (determines whether the inputs terminated strings or whether lengths are expressed explicitly) as well as the desired output (index or mask).

The imm8 control byte for PCMPESTRM/PCMPESTRI/PCMPISTRM/PCMPISTRI encodes a significant amount of programmable control over the functionality of those instructions. Some functionality is unique to each instruction while some is common across some or all of the four instructions. This section describes functionality which is common across the four instructions.

The arithmetic flags (ZF, CF, SF, OF, AF, PF) are set as a result of these instructions. However, the meanings of the flags have been overloaded from their typical meanings in order to provide additional information regarding the relationships of the two inputs.

PCMPxSTRx instructions perform arithmetic comparisons between all possible pairs of bytes or words, one from each packed input source operand. The boolean results of those comparisons are then aggregated in order to produce meaningful results. The imm8 control byte is used to affect the interpretation of individual input elements as well as control the arithmetic comparisons used and the specific aggregation scheme.

Specifically, the imm8 Control Byte consists of bit fields that control the following attributes:

- **Source data format** — Byte/word data element granularity, signed or unsigned elements.
- **Aggregation operation** — Encodes the mode of per-element comparison operation and the aggregation of per-element comparisons into an intermediate result.
- **Polarity** — Specifies intermediate processing to be performed on the intermediate result.
- **Output selection** — Specifies final operation to produce the output (depending on index or mask) from the intermediate result.

4.1.2 Source Data Format

Table 4-1. Source Data Format

Imm8[1:0]	Meaning	Description
00b	Unsigned bytes	Both 128-bit sources are treated as packed, unsigned bytes.
01b	Unsigned words	Both 128-bit sources are treated as packed, unsigned words.
10b	Signed bytes	Both 128-bit sources are treated as packed, signed bytes.
11b	Signed words	Both 128-bit sources are treated as packed, signed words.

If the imm8 control byte has bit[0] cleared, each source contains 16 packed bytes. If the bit is set each source contains 8 packed words. If the imm8 control byte has bit[1] cleared, each input contains unsigned data. If the bit is set each source contains signed data.

4.1.3 Aggregation Operation

Table 4-2. Aggregation Operation

Imm8[3:2]	Mode	Comparison
00b	Equal any	The arithmetic comparison is “equal.”
01b	Ranges	Arithmetic comparison is “greater than or equal” between even indexed bytes/words of reg and each byte/word of reg/mem. Arithmetic comparison is “less than or equal” between odd indexed bytes/words of reg and each byte/word of reg/mem. (reg/mem[m] >= reg[n] for n = even, reg/mem[m] <= reg[n] for n = odd)
10b	Equal each	The arithmetic comparison is “equal.”
11b	Equal ordered	The arithmetic comparison is “equal.”

All 256 (64) possible comparisons are always performed. The individual Boolean results of those comparisons are referred by “BoolRes[Reg/Mem element index, Reg element index].” Comparisons evaluating to “True” are represented with a 1, False with a 0 (positive logic). The initial results are then aggregated into a 16-bit (8-bit) intermediate result (IntRes1) using one of the modes described in the table below, as determined by imm8 control byte bits[3:2].

See Section 4.1.6 for a description of the overrideIfDataInvalid() function used in Table 4-3.

Table 4-3. Aggregation Operation

Mode	Pseudocode
Equal any (find characters from a set)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i++ IntRes1[j] OR= overrideIfDataInvalid(BoolRes[j,i])
Ranges (find characters from ranges)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For j = 0 to UpperBound, j++ For i = 0 to UpperBound, i+=2 IntRes1[j] OR= (overrideIfDataInvalid(BoolRes[j,i]) AND overrideIfDataInvalid(BoolRes[j,i+1]))
Equal each (string compare)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = 0; For i = 0 to UpperBound, i++ IntRes1[i] = overrideIfDataInvalid(BoolRes[i,i])
Equal ordered (substring search)	UpperBound = imm8[0] ? 7 : 15; IntRes1 = imm8[0] ? FFH : FFFFH For j = 0 to UpperBound, j++ For i = 0 to UpperBound-j, k=j to UpperBound, k++, i++ IntRes1[j] AND= overrideIfDataInvalid(BoolRes[k,i])

4.1.4 Polarity

IntRes1 may then be further modified by performing a 1’s complement, according to the value of the imm8 control byte bit[4]. Optionally, a mask may be used such that only those IntRes1 bits which correspond to “valid” reg/mem input elements are complemented (note that the definition of a valid input element is dependent on the specific opcode and is defined in each opcode’s description). The result of the possible negation is referred to as IntRes2.

Table 4-4. Polarity

Imm8[5:4]	Operation	Description
00b	Positive Polarity (+)	IntRes2 = IntRes1
01b	Negative Polarity (-)	IntRes2 = -1 XOR IntRes1
10b	Masked (+)	IntRes2 = IntRes1
11b	Masked (-)	IntRes2[i] = IntRes1[i] if reg/mem[i] invalid, else = ~IntRes1[i]

4.1.5 Output Selection

Table 4-5. Output Selection

Imm8[6]	Operation	Description
0b	Least significant index	The index returned to ECX is of the least significant set bit in IntRes2.
1b	Most significant index	The index returned to ECX is of the most significant set bit in IntRes2.

For PCMPESTRI/PCMPISTRI, the imm8 control byte bit[6] is used to determine if the index is of the least significant or most significant bit of IntRes2.

Table 4-6. Output Selection

Imm8[6]	Operation	Description
0b	Bit mask	IntRes2 is returned as the mask to the least significant bits of XMM0 with zero extension to 128 bits.
1b	Byte/word mask	IntRes2 is expanded into a byte/word mask (based on imm8[1]) and placed in XMM0. The expansion is performed by replicating each bit into all of the bits of the byte/word of the same index.

Specifically for PCMPESTRM/PCMPISTRM, the imm8 control byte bit[6] is used to determine if the mask is a 16 (8) bit mask or a 128 bit byte/word mask.

4.1.6 Valid/Invalid Override of Comparisons

PCMPxSTRx instructions allow for the possibility that an end-of-string (EOS) situation may occur within the 128-bit packed data value (see the instruction descriptions below for details). Any data elements on either source that are determined to be past the EOS are considered to be invalid, and the treatment of invalid data within a comparison pair varies depending on the aggregation function being performed.

In general, the individual comparison result for each element pair BoolRes[i..j] can be forced true or false if one or more elements in the pair are invalid. See Table 4-7.

Table 4-7. Comparison Result for Each Element Pair BoolRes[i..j]

xmm1 byte/ word	xmm2/ m128 byte/word	Imm8[3:2] = 00b (equal any)	Imm8[3:2] = 01b (ranges)	Imm8[3:2] = 10b (equal each)	Imm8[3:2] = 11b (equal ordered)
Invalid	Invalid	Force false	Force false	Force true	Force true
Invalid	Valid	Force false	Force false	Force false	Force true
Valid	Invalid	Force false	Force false	Force false	Force false
Valid	Valid	Do not force	Do not force	Do not force	Do not force

4.1.7 Summary of Im8 Control byte

Table 4-8. Summary of Imm8 Control Byte

Imm8	Description
-----0b	128-bit sources treated as 16 packed bytes.
-----1b	128-bit sources treated as 8 packed words.
-----0-b	Packed bytes/words are unsigned.
-----1-b	Packed bytes/words are signed.
---00--b	Mode is equal any.
---01--b	Mode is ranges.
---10--b	Mode is equal each.
---11--b	Mode is equal ordered.
---0---b	IntRes1 is unmodified.
---1---b	IntRes1 is negated (1's complement).
--0----b	Negation of IntRes1 is for all 16 (8) bits.
--1----b	Negation of IntRes1 is masked by reg/mem validity.
-0-----b	Index of the least significant, set, bit is used (regardless of corresponding input element validity). IntRes2 is returned in least significant bits of XMM0.
-1-----b	Index of the most significant, set, bit is used (regardless of corresponding input element validity). Each bit of IntRes2 is expanded to byte/word.
0-----b	This bit currently has no defined effect, should be 0.
1-----b	This bit currently has no defined effect, should be 0.

4.1.8 Diagram Comparison and Aggregation Process

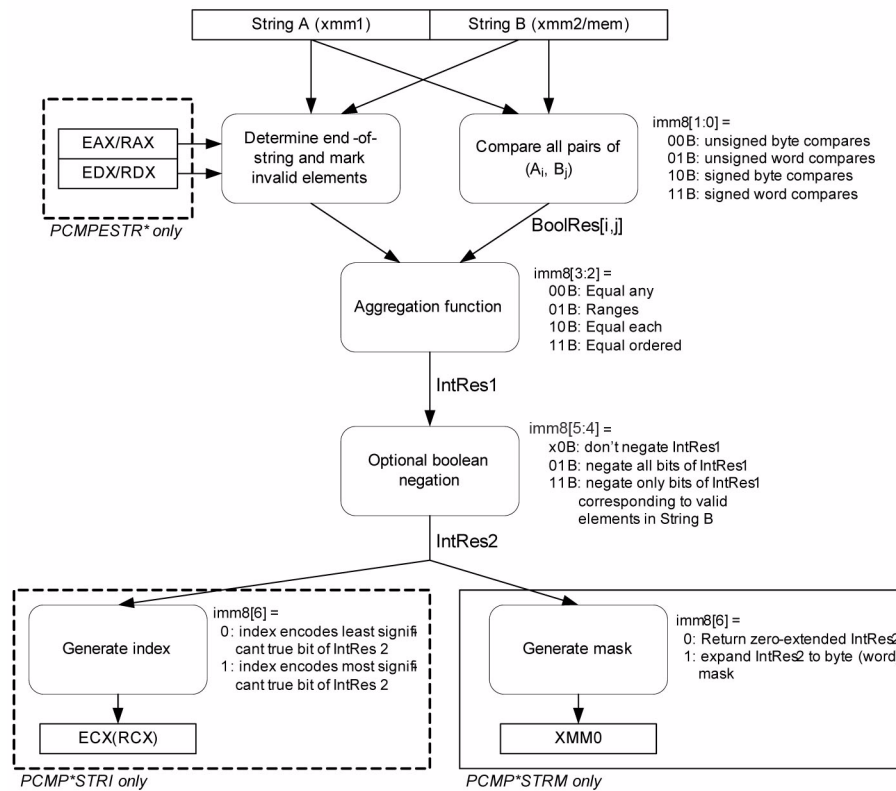


Figure 4-1. Operation of PCMPSTRx and PCMPSTRx

4.2 COMMON TRANSFORMATION AND PRIMITIVE FUNCTIONS FOR SHA1XXX AND SHA256XXX

The following primitive functions and transformations are used in the algorithmic descriptions of SHA1 and SHA256 instruction extensions SHA1NEXTE, SHA1RND4, SHA1MSG1, SHA1MSG2, SHA256RND4, SHA256MSG1, and SHA256MSG2. The operands of these primitives and transformation are generally 32-bit DWORD integers.

- **f0()**: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 1 to 20 processing.

$$f0(B,C,D) := (B \text{ AND } C) \text{ XOR } ((\text{NOT}(B) \text{ AND } D))$$
- **f1()**: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 21 to 40 processing.

$$f1(B,C,D) := B \text{ XOR } C \text{ XOR } D$$
- **f2()**: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 41 to 60 processing.

$$f2(B,C,D) := (B \text{ AND } C) \text{ XOR } (B \text{ AND } D) \text{ XOR } (C \text{ AND } D)$$
- **f3()**: A bit oriented logical operation that derives a new dword from three SHA1 state variables (dword). This function is used in SHA1 round 61 to 80 processing. It is the same as f1().

$$f3(B,C,D) := B \text{ XOR } C \text{ XOR } D$$
- **Ch()**: A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).

$\text{Ch}(E,F,G) := (E \text{ AND } F) \text{ XOR } ((\text{NOT } E) \text{ AND } G)$

- $\text{Maj}()$: A bit oriented logical operation that derives a new dword from three SHA256 state variables (dword).
 $\text{Maj}(A,B,C) := (A \text{ AND } B) \text{ XOR } (A \text{ AND } C) \text{ XOR } (B \text{ AND } C)$

ROR is rotate right operation

$(A \text{ ROR } N) := A[N-1:0] \parallel A[\text{Width}-1:N]$

ROL is rotate left operation

$(A \text{ ROL } N) := A \text{ ROR } (\text{Width}-N)$

SHR is the right shift operation

$(A \text{ SHR } N) := \text{ZEROES}[N-1:0] \parallel A[\text{Width}-1:N]$

- $\Sigma_0()$: A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.
 $\Sigma_0(A) := (A \text{ ROR } 2) \text{ XOR } (A \text{ ROR } 13) \text{ XOR } (A \text{ ROR } 22)$
- $\Sigma_1()$: A bit oriented logical and rotational transformation performed on a dword SHA256 state variable.
 $\Sigma_1(E) := (E \text{ ROR } 6) \text{ XOR } (E \text{ ROR } 11) \text{ XOR } (E \text{ ROR } 25)$
- $\sigma_0()$: A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.
 $\sigma_0(W) := (W \text{ ROR } 7) \text{ XOR } (W \text{ ROR } 18) \text{ XOR } (W \text{ SHR } 3)$
- $\sigma_1()$: A bit oriented logical and rotational transformation performed on a SHA256 message dword used in the message scheduling.
 $\sigma_1(W) := (W \text{ ROR } 17) \text{ XOR } (W \text{ ROR } 19) \text{ XOR } (W \text{ SHR } 10)$
- K_i : SHA1 Constants dependent on immediate i.
 $K0 = 0x5A827999$
 $K1 = 0x6ED9EBA1$
 $K2 = 0x8F1BBCDC$
 $K3 = 0xCA62C1D6$

4.3 INSTRUCTIONS (M-U)

Chapter 4 continues an alphabetical discussion of Intel® 64 and IA-32 instructions (M-U). See also: Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A; Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C; and Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

MOV—Move to/from Control Registers

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF 20/r	MOV r32, CR0–CR7	MR	N.E.	Valid	Move control register to r32.
OF 20/r	MOV r64, CR0–CR7	MR	Valid	N.E.	Move extended control register to r64.
REX.R + OF 20 /0	MOV r64, CR8	MR	Valid	N.E.	Move extended CR8 to r64. ¹
OF 22 /r	MOV CR0–CR7, r32	RM	N.E.	Valid	Move r32 to control register.
OF 22 /r	MOV CR0–CR7, r64	RM	Valid	N.E.	Move r64 to extended control register.
REX.R + OF 22 /0	MOV CR8, r64	RM	Valid	N.E.	Move r64 to extended CR8. ¹

NOTES:

1. MOV CR* instructions, except for MOV CR8, are serializing instructions. MOV CR8 is not architecturally defined as a serializing instruction. For more information, see Chapter 10 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	N/A	N/A
RM	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Moves the contents of a control register (CR0, CR2, CR3, CR4, or CR8) to a general-purpose register or the contents of a general-purpose register to a control register. The operand size for these instructions is always 32 bits in non-64-bit modes, regardless of the operand-size attribute. On a 64-bit capable processor, an execution of MOV to CR outside of 64-bit mode zeros the upper 32 bits of the control register. (See "Control Registers" in Chapter 2 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for a detailed description of the flags and fields in the control registers.) This instruction can be executed only when the current privilege level is 0.

At the opcode level, the *reg* field within the ModR/M byte specifies which of the control registers is loaded or read. The 2 bits in the *mod* field are ignored. The *r/m* field specifies the general-purpose register loaded or read. Some of the bits in CR0, CR3, and CR4 are reserved and must be written with zeros. Attempting to set any reserved bits in CR0[31:0] is ignored. Attempting to set any reserved bits in CR0[63:32] results in a general-protection exception, #GP(0). When PCIDs are not enabled, bits 2:0 and bits 11:5 of CR3 are not used and attempts to set them are ignored. [See the next paragraph for treatment of bits reserved in CR3.](#) Attempting to set any reserved bits in CR4 results in #GP(0). On Pentium 4, Intel Xeon and P6 family processors, CR0.ET remains set after any load of CR0; attempts to clear this bit have no impact.

Normally, MAXPHYADDR is the value enumerated in CPUID.80000008H:EAX[7:0]. However, if IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3*); the value is not reduced in SEAM. An attempt to set any reserved bit in CR3[63:MAXPHYADDR] results in #GP(0).

In certain cases, these instructions have the side effect of invalidating entries in the TLBs and the paging-structure caches. See Section 5.10.4.1, "Operations that Invalidate TLBs and Paging-Structure Caches," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, for details.

The following side effects are implementation-specific for the Pentium 4, Intel Xeon, and P6 processor family: when modifying PE or PG in register CR0, or PSE or PAE in register CR4, all TLB entries are flushed, including global entries. Software should not depend on this functionality in all Intel 64 or IA-32 processors.

In 64-bit mode, the instruction's default operation size is 64 bits. The REX.R prefix must be used to access CR8. Use of REX.B permits access to additional registers (R8–R15). Use of the REX.W prefix or 66H prefix is ignored. Use of the REX.R prefix to specify a register other than CR8 causes an invalid-opcode exception. See the summary chart at the beginning of this section for encoding data and limits.

If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 determines whether the instruction invalidates entries in the TLBs and the paging-structure caches (see Section 5.10.4.1, “Operations that Invalidate TLBs and Paging-Structure Caches,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). The instruction does not modify bit 63 of CR3, which is reserved and always 0.

See “Changes to Instruction Behavior in VMX Non-Root Operation” in Chapter 26 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C, for more information about the behavior of this instruction in VMX non-root operation.

Operation

DEST := SRC;

Flags Affected

The OF, SF, ZF, AF, PF, and CF flags are undefined.

Protected Mode Exceptions

- #GP(0) If the current privilege level is not 0.
 If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
 If an attempt is made to write a 1 to any reserved bit in CR4.
 If an attempt is made to write 1 to CR4.PCIDE.
 If any of the reserved bits are set in the page-directory pointers table (PDPT) and the loading of a control register causes the PDPT to be loaded into the processor.
 If an attempt is made to activate IA-32e mode and either the current CS has the L-bit set or the TR references a 16-bit TSS.
- #UD If the LOCK prefix is used.
 If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.

Real-Address Mode Exceptions

- #GP If an attempt is made to write a 1 to any reserved bit in CR4.
 If an attempt is made to write 1 to CR4.PCIDE.
 If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0).
 If an attempt is made to activate IA-32e mode and either the current CS has the L-bit set or the TR references a 16-bit TSS.
- #UD If the LOCK prefix is used.
 If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.

Virtual-8086 Mode Exceptions

- #GP(0) These instructions cannot be executed in virtual-8086 mode.

Compatibility Mode Exceptions

- #GP(0) If the current privilege level is not 0.
 If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
 If an attempt is made to change CR4.PCIDE from 0 to 1 while CR3[11:0] ≠ 000H.
 If an attempt is made to clear CR0.PG[bit 31] while CR4.PCIDE = 1.
 If an attempt is made to leave IA-32e mode by clearing CR4.PAE[bit 5].
- #UD If the LOCK prefix is used.
 If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If an attempt is made to write invalid bit combinations in CR0 (such as setting the PG flag to 1 when the PE flag is set to 0, or setting the CD flag to 0 when the NW flag is set to 1).
	If an attempt is made to change CR4.PCIDE from 0 to 1 while CR3[11:0] ≠ 000H.
	If an attempt is made to clear CR0.PG[bit 31].
	If an attempt is made to write a 1 to any reserved bit in CR4.
	If an attempt is made to write a 1 to any reserved bit in CR8.
	If an attempt is made to write a 1 to any reserved bit in CR3[63:MAXPHYADDR].
	If an attempt is made to leave IA-32e mode by clearing CR4.PAE[bit 5].
#UD	If the LOCK prefix is used.
	If an attempt is made to access CR1, CR5, CR6, CR7, or CR9–CR15.
	If the REX.R prefix is used to specify a register other than CR8.

MOVSX/MOVSXD—Move With Sign-Extension

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
OF BE /r	MOVSX r16, r/m8 ¹	RM	Valid	Valid	Move byte to word with sign-extension.
OF BE /r	MOVSX r32, r/m8 ¹	RM	Valid	Valid	Move byte to doubleword with sign-extension.
REX.W + OF BE /r	MOVSX r64, r/m8 ¹	RM	Valid	N.E.	Move byte to quadword with sign-extension.
OF BF /r	MOVSX r32, r/m16	RM	Valid	Valid	Move word to doubleword, with sign-extension.
REX.W + OF BF /r	MOVSX r64, r/m16	RM	Valid	N.E.	Move word to quadword with sign-extension.
63 /r ²	MOVSXD r16, r/m16	RM	Valid	N.E.	Move word to word with sign-extension.
63 /r ²	MOVSXD r32, r/m32	RM	Valid	N.E.	Move doubleword to doubleword with sign-extension.
REX.W + 63 /r	MOVSXD r64, r/m32	RM	Valid	N.E.	Move doubleword to quadword with sign-extension.

NOTES:

1. With a REX prefix in 64-bit mode, attempts to access AH, BH, CH, or DH will instead access SPL, DIL, BPL, or SIL, respectively.
2. The use of MOVSXD without REX.W in 64-bit mode is discouraged. Regular MOV should be used instead of using MOVSXD without REX.W.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Copies the contents of the source operand (register or memory location) to the destination operand (register) and sign extends the value to 16 or 32 bits (see Figure 7-6 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). The size of the converted value depends on the operand-size attribute.

In 64-bit mode, the instruction's default operation size is 32 bits. Use of the REX.R prefix permits access to additional registers (R8-R15). Use of the REX.W prefix promotes operation to 64 bits. See the summary chart at the beginning of this section for encoding data and limits.

Operation

DEST := SignExtend(SRC);

Flags Affected

None.

Protected Mode Exceptions

- #GP(0) If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
If the DS, ES, FS, or GS register contains a NULL segment selector.
- #SS(0) If a memory operand effective address is outside the SS segment limit.
- #PF(fault-code) If a page fault occurs.
- #AC(0) If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
- #UD If the LOCK prefix is used.

Real-Address Mode Exceptions

#GP	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS	If a memory operand effective address is outside the SS segment limit.
#UD	If the LOCK prefix is used.

Virtual-8086 Mode Exceptions

#GP(0)	If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
#SS(0)	If a memory operand effective address is outside the SS segment limit.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#GP(0)	If the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#AC(0)	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
#UD	If the LOCK prefix is used.

PCONFIG—Platform Configuration

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F 01 C5 PCONFIG	A	V/V	PCONFIG	This instruction is used to execute functions for configuring platform features.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	N/A	N/A	N/A	N/A

Description

The PCONFIG instruction allows software to configure certain platform features. It supports these features with multiple leaf functions, selecting a leaf function using the value in EAX.

Depending on the leaf function, the registers RBX, RCX, and RDX may be used to provide input information or for the instruction to report output information. Addresses and operands are 32 bits outside 64-bit mode and are 64 bits in 64-bit mode. The value of CS.D does not affect operand size or address size.

Executions of PCONFIG may fail for platform-specific reasons. An execution reports failure by setting the ZF flag and loading EAX with a non-zero failure reason; a successful execution clears ZF and EAX.

Each PCONFIG leaf function applies to a specific hardware block called a PCONFIG target. The leaf function is supported only if the processor supports that target. Each target is associated with a numerical target identifier, and CPUID.1BH (PCONFIG information) enumerates the identifiers of the supported targets. An attempt to execute an undefined leaf function, or a leaf function that applies to an unsupported target identifier, results in a general-protection exception (#GP).

Leaf Function MKTME_KEY_PROGRAM

As of this writing, the only defined PCONFIG leaf function is used for key programming for total memory encryption-multi-key (TME-MK).¹ This leaf function is called MKTME_KEY_PROGRAM and it pertains to the TME-MK target, which has target identifier 1. The leaf function is selected by loading EAX with value 0. The MKTME_KEY_PROGRAM leaf function uses the EBX (or RBX) register for additional input information.

Software uses the MKTME_KEY_PROGRAM leaf function to manage the encryption key associated with a particular key identifier (KeyID). The leaf function uses a data structure called the **TME-MK key programming structure** (MKTME_KEY_PROGRAM_STRUCT). Software provides the address of the structure (as an offset in the DS segment) in EBX (or RBX). The format of the structure is given in Table 4-15.

Table 4-15. MKTME_KEY_PROGRAM_STRUCT Format

Field	Offset (bytes)	Size (bytes)	Comments
KEYID	0	2	Key Identifier.
KEYID_CTRL	2	4	KeyID control: ▪ Bits 7:0: key-programming command (COMMAND) ▪ Bits 23:8: encryption algorithm (ENC_ALG) ▪ Bits 31:24: Reserved, must be zero (RSVD)
Ignored	6	58	Not used.
KEY_FIELD_1	64	64	Software supplied data key or entropy for data key.
KEY_FIELD_2	128	64	Software supplied tweak key or entropy for tweak key.

1. Further details on TME-MK can be found here:

<https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf>

A description of each of the fields in MKTME_KEY_PROGRAM_STRUCT is provided below:

- **KEYID:** The key identifier (KeyID) being programmed to the TME-MK engine. PCONFIG causes a general-protection exception (#GP) if the KeyID is zero. KeyID zero always uses the current behavior configured for TME (total memory encryption), either to encrypt with platform TME key or to bypass TME encryption. PCONFIG also causes a #GP if the KeyID exceeds the maximum enumerated in IA32_TME_CAPABILITY.MK_TME_MAX_KEYS[50:36] or configured by the setting of IA32_TME_ACTIVATE[39:32]:
 - If IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32] = k , KEYID[15: k] must be zero.
 - If the logical processor is outside secure arbitration mode (SEAM; see Chapter 34) and IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS[39:36] = $p > 0$, KEYID[$k-1:k-p$] must also be zero. (Note that p will never exceed k .)

If either is not true, PCONFIG causes #GP.

- **KEYID_CTRL:** The KEYID_CTRL field comprises two sub-fields used by software to control the encryption performed for the selected KeyID:
 - Key-programming command (COMMAND; bits 7:0). This 8-bit field should contain one of the following values:
 - KEYID_SET_KEY_DIRECT (value 0). With this command, software programs directly the encryption key to be used for the selected KeyID.
 - KEYID_SET_KEY_RANDOM (value 1). With this command, software has the CPU generate and assign an encryption key to be used for the selected KeyID using a hardware random-number generator.
If this command is used and there is insufficient entropy for the random-number generator, PCONFIG will fail and report the failure by loading EAX with value 2 (ENTROPY_ERROR).
Because the keys programed by PCONFIG are discarded on reset and software cannot read the programmed keys, the keys programmed with this command are ephemeral.
 - KEYID_CLEAR_KEY (value 2). With this command, software indicates that the selected KeyID should use the current behavior configured for TME (see above).
 - KEYID_NO_ENCRYPT (value 3). With this command, software indicates that no encryption should be used for the selected KeyID.

If any other value is used, PCONFIG causes a #GP.

- Encryption algorithm (ENC_ALG, bits 23:8). Bits 63:48 of the IA32_TME_ACTIVATE MSR (MSR index 982H) indicate which encryption algorithms are supported by the platform. The 16-bit ENC_ALG field should specify one of the algorithms indicated in IA32_TME_ACTIVATE. PCONFIG causes a #GP if ENC_ALG does not set exactly one bit or if it sets a bit whose corresponding bit is not set in IA32_TME_ACTIVATE[63:48].
- **KEY_FIELD_1:** Use of this field depends upon selected key-programming command:
 - If the direct key-programming command is used (KEYID_SET_KEY_DIRECT), this field carries the software supplied data key to be used for the KeyID.
 - If the random key-programming command is used (KEYID_SET_KEY_RANDOM), this field carries the software supplied entropy to be mixed in the CPU generated random data key.
 - This field is ignored when one of the other key-programming commands is used.It is software's responsibility to ensure that the key supplied for the direct key-programming option or the entropy supplied for the random key-programming option does not result in weak keys. There are no explicit checks in the instruction to detect or prevent weak keys.
- **KEY_FIELD_2:** Use of this field depends upon selected key-programming command:
 - If the direct key-programming command is used (KEYID_SET_KEY_DIRECT), this field carries the software supplied tweak key to be used for the KeyID.
 - If the random key-programming command is used (KEYID_SET_KEY_RANDOM), this field carries the software supplied entropy to be mixed in the CPU generated random tweak key.
 - This field is ignored when one of the other key-programming commands is used.

It is software's responsibility to ensure that the key supplied for the direct key-programming option or the entropy supplied for the random key-programming option does not result in weak keys. There are no explicit checks in the instruction to detect or prevent weak keys.

All KeyIDs default to TME behavior (encrypt with TME key or bypass encryption) on activation of TME-MK. Software can at any point decide to change the key for a KeyID using the MKTME_KEY_PROGRAM leaf function of the PCONFIG instruction. Changing the key for a KeyID does **not** change the state of the TLB caches or memory pipeline. Software is responsible for taking appropriate actions to ensure correct behavior.

The key table used by TME-MK is shared by all logical processors in a platform. For this reason, execution of the MKTME_KEY_PROGRAM leaf function must gain exclusive access to the key table before updating it. The leaf function does this by acquiring lock (implemented in the platform) and retaining that lock until the execution completes. An execution of the leaf function may fail to acquire the lock if it is already in use. In this situation, the leaf function will load EAX with failure reason 5 (DEVICE_BUSY) indicating that software must retry. When this happens, the key table is not updated, and software should retry execution of PCONFIG.

NOTES

Earlier versions of this manual specified that bytes 63:6 of MKTME_KEY_PROGRAM_STRUCT were reserved and that PCONFIG would cause a #GP if they were not all zero. This is not the case. As indicated in Table 4-15, PCONFIG ignores those bytes.

They also specified that PCONFIG would cause a #GP if the upper 48 bytes of each of the 64-byte key fields were not all 0. This is not the case. From each of these fields, PCONFIG uses the number of bytes required by the selected encryption algorithm (e.g., 32 bytes for AES-XTS 256) and ignores the upper bytes.

They also specified that PCONFIG would complete and report a failure reason in EAX if the structure specified an incorrect KeyID, and unsupported key-programming command, or an incorrect selection of an encryption algorithm. This is not the case. As indicated above (and in the Operation section), those conditions cause #GP.

Operation

(* #UD if PCONFIG is not enumerated or CPL > 0 *)

IF CPUID.07H.00H:EDX[18] = 0 OR CPL > 0
THEN #UD; FI;

(* #GP(0) for an unsupported leaf function *)

IF EAX != 0
THEN #GP(0); FI;

CASE (EAX) (* operation based on selected leaf function *)

0 (MKTME_KEY_PROGRAM):

(* Confirm that TME-MK is properly enabled by the IA32_TME_ACTIVATE MSR *)

(* The MSR must be locked, encryption enabled, and a non-zero number of KeyID bits specified *)

IF IA32_TME_ACTIVATE[0] = 0 OR IA32_TME_ACTIVATE[1] = 0 OR IA32_TME_ACTIVATE[35:32] = 0
THEN #GP(0); FI;

IF DS:RBX is not 256-byte aligned
THEN #GP(0); FI;

Load TMP_KEY_PROGRAM_STRUCT from 192 bytes at linear address DS:RBX;

IF TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL sets any reserved bits
THEN #GP(0); FI;

(* Check for a valid command *)

IF TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.COMMAND > 3
THEN #GP(0); FI;

```

(* Check that the KEYID being operated upon is a valid KEYID *)
IF TMP_KEY_PROGRAM_STRUCT.KEYID = 0 OR
   TMP_KEY_PROGRAM_STRUCT.KEYID > IA32_TME_CAPABILITY.MK_TME_MAX_KEYS
   THEN #GP(0); FI;
k := IA32_TME_ACTIVATE.MK_TME_KEYID_BITS;
IF TMP_KEY_PROGRAM_STRUCT.KEYID[15:k] != 0
   THEN #GP(0); FI;
IF not in SEAM AND IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS > 0
   THEN
      p := IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS;
      IF TMP_KEY_PROGRAM_STRUCT.KEYID[k-1:k-p] != 0
         THEN #GP(0); FI;
FI;

(* Check that only one encryption algorithm is requested for the KeyID and it is one of the activated algorithms *)
IF TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG does not set exactly one bit OR
   (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG & IA32_TME_ACTIVATE[63:48]) = 0
   THEN #GP(0); FI;

```

Attempt to acquire lock to gain exclusive access to platform key table;

```

IF attempt is unsuccessful
   THEN (* PCONFIG failure *)
      RFLAGS.ZF := 1;
      RAX := DEVICE_BUSY; (* failure reason 5 *)
      GOTO EXIT;
FI;

```

```

CASE (TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.COMMAND) OF
  0 (KEYID_SET_KEY_DIRECT):
    Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
      Encrypt with the selected key
      Use the encryption algorithm selected by TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG
      (* The number of bytes used by the next two lines depends on selected encryption algorithm *)
      DATA_KEY is TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1
      TWEEK_KEY is TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2
    BREAK;

  1 (KEYID_SET_KEY_RANDOM):
    Load TMP_RND_DATA_KEY with a random key using hardware RNG; (* key size depends on selected encryption algorithm *)
    IF there was insufficient entropy
       THEN (* PCONFIG failure *)
          RFLAGS.ZF := 1;
          RAX := ENTROPY_ERROR; (* failure reason 2 *)
          Release lock on platform key table;
          GOTO EXIT;
    FI;
    Load TMP_RND_TWEEK_KEY with a random key using hardware RNG; (* key size depends on selected encryption algorithm *)
    IF there was insufficient entropy
       THEN (* PCONFIG failure *)
          RFLAGS.ZF := 1;
          RAX := ENTROPY_ERROR; (* failure reason 2 *)
          Release lock on platform key table;
          GOTO EXIT;

```

```

FI;
(* Combine software-supplied entropy to the data key and tweak key *)
(* The number of bytes used by the next two lines depends on selected encryption algorithm *)
TMP_RND_DATA_KEY := TMP_RND_KEY XOR TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_1;
TMP_RND_TWEAK_KEY := TMP_RND_TWEAK_KEY XOR TMP_KEY_PROGRAM_STRUCT.KEY_FIELD_2;

```

```

Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
    Encrypt with the selected key
    Use the encryption algorithm selected by TMP_KEY_PROGRAM_STRUCT.KEYID_CTRL.ENC_ALG
    (* The number of bytes used by the next two lines depends on selected encryption algorithm *)
    DATA_KEY is TMP_RND_DATA_KEY
    TWEAK_KEY is TMP_RND_TWEAK_KEY
BREAK;

```

```

2 (KEYID_CLEAR_KEY):
Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
    Encrypt (or not) using the current configuration for TME
    The specified encryption algorithm and key values are not used.
BREAK;

```

```

3 (KEYID_NO_ENCRYPT):
Update TME-MK table for TMP_KEY_PROGRAM_STRUCT.KEYID as follows:
    Do not encrypt
    The specified encryption algorithm and key values are not used.
BREAK;

```

```

ESAC;
Release lock on platform key table;
ESAC;

```

```

RAX := 0;
RFLAGS.ZF := 0;

```

```

EXIT:
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;

```

Protected Mode Exceptions

#GP(0)	If input value in EAX encodes an unsupported leaf function. If a memory operand effective address is outside the relevant segment limit. MKTME_KEY_PROGRAM leaf function: If IA32_TME_ACTIVATE MSR is not locked. If hardware encryption and TME-MK capability are not enabled in IA32_TME_ACTIVATE MSR. If the memory operand is not 256B aligned. If any of the reserved bits in the KEYID_CTRL field of the MKTME_KEY_PROGRAM_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm.
#PF(fault-code)	If a page fault occurs in accessing memory operands.
#UD	If any of the LOCK/REP/Operand Size/VEX prefixes are used. If current privilege level is not 0.

If CPUID.07H.00H:EDX.PCONFIG[18] = 0

Real-Address Mode Exceptions

- #GP If input value in EAX encodes an unsupported leaf function.
 MKTME_KEY_PROGRAM leaf function:
 If IA32_TME_ACTIVATE MSR is not locked.
 If hardware encryption and TME-MK capability are not enabled in IA32_TME_ACTIVATE MSR.
 If a memory operand is not 256B aligned.
 If any of the reserved bits in the KEYID_CTRL field of the MKTME_KEY_PROGRAM_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm.
- #UD If any of the LOCK/REP/Operand Size/VEX prefixes are used.
 If current privilege level is not 0.
 If CPUID.07H.00H:EDX.PCONFIG[18] = 0

Virtual-8086 Mode Exceptions

- #UD PCONFIG instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

- #GP(0) If input value in EAX encodes an unsupported leaf function.
 If a memory operand is non-canonical form.
 MKTME_KEY_PROGRAM leaf function:
 If IA32_TME_ACTIVATE MSR is not locked.
 If hardware encryption and TME-MK capability are not enabled in IA32_TME_ACTIVATE MSR.
 If a memory operand is not 256B aligned.
 If any of the reserved bits in the KEYID_CTRL field of the MKTME_KEY_PROGRAM_STRUCT are set or that field indicates an unsupported KeyID, key-programming command, or encryption algorithm.
- #PF(fault-code) If a page fault occurs in accessing memory operands.
- #UD If any of the LOCK/REP/Operand Size/VEX prefixes are used.
 If the current privilege level is not 0.
 If CPUID.07H.00H:EDX.PCONFIG[18] = 0.

UD—Undefined Instruction

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
0F FF /r	UD0 ¹ r32, r/m32	RM	Valid	Valid	Raise invalid opcode exception.
0F B9 /r	UD1 r32, r/m32	RM	Valid	Valid	Raise invalid opcode exception.
0F 0B	UD2	ZO	Valid	Valid	Raise invalid opcode exception.
D6	UDB	ZO	Valid	N.E.	Raise invalid opcode exception.

NOTES:

1. Some processors decode the UD0 instruction without a ModR/M byte. As a result, those processors would deliver an invalid-opcode exception instead of a fault on instruction fetch when the instruction with a ModR/M byte (and any implied bytes) would cross a page or segment boundary.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	N/A	N/A	N/A	N/A
RM	ModRM:reg (r)	ModRM:r/m (r)	N/A	N/A

Description

Generates an invalid opcode exception. This instruction is provided for software testing to explicitly generate an invalid opcode exception. The opcodes for this instruction are reserved for this purpose.

Other than raising the invalid opcode exception, this instruction has no effect on processor state or memory.

Even though it is the execution of the UD instruction that causes the invalid opcode exception, the instruction pointer saved by delivery of the exception references the UD instruction (and not the following instruction).

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

Operation

#UD (* Generates invalid opcode exception *);

Flags Affected

None.

Exceptions (All Operating Modes)

#UD Raises an invalid opcode exception in all operating modes. (UDB does so only in 64-bit mode.)

6. Updates to Chapter 5, Volume 2C

Change bars and violet text show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C: Instruction Set Reference, V.*

Changes to this chapter:

- For VCVTUQQ2PH—Convert Packed Unsigned Quadword Integers to Packed FP16 Values, corrected the size in the description to quadword.

5.1 TERNARY BIT VECTOR LOGIC TABLE

VPTERNLOGD/VPTERNLOGQ instructions operate on dword/qword elements and take three bit vectors of the respective input data elements to form a set of 32/64 indices, where each 3-bit value provides an index into an 8-bit lookup table represented by the imm8 byte of the instruction. The 256 possible values of the imm8 byte is constructed as a 16x16 boolean logic table. The 16 rows of the table uses the lower 4 bits of imm8 as row index. The 16 columns are referenced by imm8[7:4]. The 16 columns of the table are present in two halves, with 8 columns shown in Table 5-1 for the column index value between 0:7, followed by Table 5-2 showing the 8 columns corresponding to column index 8:15. This section presents the two-halves of the 256-entry table using a short-hand notation representing simple or compound boolean logic expressions with three input bit source data.

The three input bit source data will be denoted with the capital letters: A, B, C; where A represents a bit from the first source operand (also the destination operand), B and C represent a bit from the 2nd and 3rd source operands.

Each map entry takes the form of a logic expression consisting of one or more component expressions. Each component expression consists of either a unary or binary boolean operator and associated operands. Each binary boolean operator is expressed in lowercase letters, and operands concatenated after the logic operator. The unary operator 'not' is expressed using '!'. Additionally, the conditional expression "A?B:C" expresses a result returning B if A is set, returning C otherwise.

A binary boolean operator is followed by two operands, e.g., andAB. For a compound binary expression that contains commutative components and comprising the same logic operator, the 2nd logic operator is omitted and three operands can be concatenated in sequence, e.g., andABC. When the 2nd operand of the first binary boolean expression comes from the result of another boolean expression, the 2nd boolean expression is concatenated after the uppercase operand of the first logic expression, e.g., norBnandAC. When the result is independent of an operand, that operand is omitted in the logic expression, e.g., zeros or norCB.

The 3-input expression "majorABC" returns 0 if two or more input bits are 0, returns 1 if two or more input bits are 1. The 3-input expression "minorABC" returns 1 if two or more input bits are 0, returns 0 if two or more input bits are 1.

The building-block bit logic functions used in Table 5-1 and Table 5-2 include:

- Constants: TRUE (1), FALSE (0);
- Unary function: Not (!);
- Binary functions: and, nand, or, nor, xor, xnor;
- Conditional function: Select (?:);
- Tertiary functions: major, minor.

Table 5-1. Lower 8 columns of the 16x16 Map of VPTERNLOG Boolean Logic Operations

Imm	[7:4]							
[3:0]	0H	1H	2H	3H	4H	5H	6H	7H
00H	FALSE	andAnorBC	norBnandAC	andA!B	norCnandBA	andA!C	andAxorBC	andAnandBC
01H	norABC	norCB	norBxorAC	A?!B:norBC	norCxorBA	A?!C:norBC	A?xorBC:norBC	A?nandBC:norBC
02H	andCnorBA	norBxnorAC	andC!B	norBnorAC	C?norBA:andBA	C?norBA:A	C?!B:andBA	C?!B:A
03H	norBA	norBandAC	C?!B:norBA	!B	C?norBA:xnorBA	A?!C:!B	A?xorBC:!B	A?nandBC:!B
04H	andBnorAC	norCxnorBA	B?norAC:andAC	B?norAC:A	andB!C	norCnorBA	B?!C:andAC	B?!C:A
05H	norCA	norCandBA	B?norAC:xnorAC	A?!B:!C	B?!C:norAC	!C	A?xorBC:!C	A?nandBC:!C
06H	norAxnorBC	A?norBC:xorBC	B?norAC:C	xorBorAC	C?norBA:B	xorCorBA	xorCB	B?!C:orAC
07H	norAandBC	minorABC	C?!B:!A	nandBorAC	B?!C:!A	nandCorBA	A?xorBC:nandBC	nandCB
08H	norAnandBC	A?norBC:andBC	andCxorBA	A?!B:andBC	andBxorAC	A?!C:andBC	A?xorBC:andBC	xorAandBC
09H	norAxorBC	A?norBC:xnorBC	C?xorBA:norBA	A?!B:xnorBC	B?xorAC:norAC	A?!C:xnorBC	xnorABC	A?nandBC:xnorBC
0AH	andC!A	A?norBC:C	andCnandBA	A?!B:C	C?!A:andBA	xorCA	xorCandBA	A?nandBC:C
0BH	C?!A:norBA	C?!A:!B	C?nandBA:norBA	C?nandBA:!B	B?xorAC:!A	B?xorAC:nandAC	C?nandBA:xnorBA	nandBxnorAC
0CH	andB!A	A?norBC:B	B?!A:andAC	xorBA	andBnandAC	A?!C:B	xorBandAC	A?nandBC:B
0DH	B?!A:norAC	B?!A:!C	B?!A:xnorAC	C?xorBA:nandBA	B?nandAC:norAC	B?nandAC:!C	B?nandAC:xnorAC	nandCxnorBA
0EH	norAnorBC	xorAorBC	B?!A:C	A?!B:orBC	C?!A:B	A?!C:orBC	B?nandAC:C	A?nandBC:orBC
0FH	!A	nandAorBC	C?nandBA:!A	nandBA	B?nandAC:!A	nandCA	nandAxnorBC	nandABC

Table 5-2 shows the half of 256-entry map corresponding to column index values 8:15.

Table 5-2. Upper 8 columns of the 16x16 Map of VPTERNLOG Boolean Logic Operations

Imm	[7:4]							
[3:0]	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
00H	andABC	andAxnorBC	andCA	B?andAC:A	andBA	C?andBA:A	andAorBC	A
01H	A?andBC:norBC	B?andAC:!C	A?C:norBC	C?A:!B	A?B:norBC	B?A:!C	xnorAorBC	orAnorBC
02H	andCxnorBA	B?andAC:xorAC	B?andAC:C	B?andAC:orAC	C?xnorBA:andBA	B?A:xorAC	B?A:C	B?A:orAC
03H	A?andBC:!B	xnorBandAC	A?C:!B	nandBnandAC	xnorBA	B?A:nandAC	A?orBC:!B	orA!B
04H	andBxnorAC	C?andBA:xorBA	B?xnorAC:andAC	B?xnorAC:A	C?andBA:B	C?andBA:orBA	C?A:B	C?A:orBA
05H	A?andBC:!C	xnorCandBA	xnorCA	C?A:nandBA	A?B:!C	nandCnandBA	A?orBC:!C	orA!C
06H	A?andBC:xorBC	xorABC	A?C:xorBC	B?xnorAC:orAC	A?B:xorBC	C?xnorBA:orBA	A?orBC:xorBC	orAxorBC
07H	xnorAandBC	A?xnorBC:nandBC	A?C:nandBC	nandBxorAC	A?B:nandBC	nandCxorBA	A?orBCnandBC	orAnandBC
08H	andCB	A?xnorBC:andBC	andCorAB	B?C:A	andBorAC	C?B:A	majorABC	orAandBC
09H	B?C:norAC	xnorCB	xnorCorBA	C?orBA:!B	xnorBorAC	B?orAC:!C	A?orBC:xnorBC	orAxnorBC
0AH	A?andBC:C	A?xnorBC:C	C	B?C:orAC	A?B:C	B?orAC:xorAC	orCandBA	orCA
0BH	B?C:!A	B?C:nandAC	orCnorBA	orC!B	B?orAC:!A	B?orAC:nandAC	orCxnorBA	nandBnorAC
0CH	A?andBC:B	A?xnorBC:B	A?C:B	C?orBA:xorBA	B	C?B:orBA	orBandAC	orBA
0DH	C?B!A	C?B:nandBA	C?orBA:!A	C?orBA:nandBA	orBnorAC	orB!C	orBxnorAC	nandCnorBA
0EH	A?andBC:orBC	A?xnorBC:orBC	A?C:orBC	orCxorBA	A?B:orBC	orBxorAC	orCB	orABC
0FH	nandAnandBC	nandAxorBC	orCIA	orCnandBA	orB!A	orBnandAC	nandAnorBC	TRUE

Table 5-1 and Table 5-2 translate each of the possible value of the imm8 byte to a Boolean expression. These tables can also be used by software to translate Boolean expressions to numerical constants to form the imm8 value needed to construct the VPTERNLOG syntax. There is a unique set of three byte constants (F0H, CCH, AAH) that can be used for this purpose as input operands in conjunction with the Boolean expressions defined in those tables. The reverse mapping can be expressed as:

Result_imm8 = Table_Lookup_Entry(0F0H, 0CCH, 0AAH)

Table_Lookup_Entry is the Boolean expression defined in Table 5-1 and Table 5-2.

5.2 INSTRUCTIONS (V)

Chapter 5 continues an alphabetical discussion of Intel® 64 and IA-32 instructions (V). See also: Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume

2A; Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B; and Chapter 5, “Instruction Set Reference, V,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

VCVTUQQ2PH—Convert Packed Unsigned Quadword Integers to Packed FP16 Values

Opcode/ Instruction	Op/ En	64/32 Bit Mode Support	CPUID Feature Flag	Description
EVEX.128.F2.MAP5.W1 7A /r VCVTUQQ2PH xmm1{k1}{z}, xmm2/m128/m64bcst	A	V/V	(AVX512_FP16 AND AVX512VL) OR AVX10.1	Convert two packed unsigned quadword integers from xmm2/m128/m64bcst to packed FP16 values, and store the result in xmm1 subject to writemask k1.
EVEX.256.F2.MAP5.W1 7A /r VCVTUQQ2PH xmm1{k1}{z}, ymm2/m256/m64bcst	A	V/V	(AVX512_FP16 AND AVX512VL) OR AVX10.1	Convert four packed unsigned quadword integers from ymm2/m256/m64bcst to packed FP16 values, and store the result in xmm1 subject to writemask k1.
EVEX.512.F2.MAP5.W1 7A /r VCVTUQQ2PH xmm1{k1}{z}, zmm2/m512/m64bcst {er}	A	V/V	AVX512_FP16 OR AVX10.1	Convert eight packed unsigned quadword integers from zmm2/m512/m64bcst to packed FP16 values, and store the result in xmm1 subject to writemask k1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	Full	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

This instruction converts packed unsigned quadword integers in the source operand to packed FP16 values in the destination operand. The destination elements are updated according to the writemask.

EVEX.vvvv is reserved and must be 1111b otherwise instructions will #UD.

If the result of the convert operation is overflow and MXCSR.OM=0 then a SIMD exception will be raised with OE=1, PE=1.

Operation

VCVTUQQ2PH dest, src

VL = 128, 256 or 512

KL := VL / 64

IF *SRC is a register* and (VL = 512) AND (EVEX.b = 1):

SET_RM(EVEX.RC)

ELSE:

SET_RM(MXCSR.RC)

FOR j := 0 TO KL-1:

IF k1[j] OR *no writemask*:

IF *SRC is memory* and EVEX.b = 1:

tsrc := SRC.qword[0]

ELSE

tsrc := SRC.qword[j]

DEST.fp16[j] := Convert_unsigned_integer64_to_fp16(tsrc)

ELSE IF *zeroing*:

DEST.fp16[j] := 0

// else dest.fp16[j] remains unchanged

DEST[MAXVL-1:VL/4] := 0

Intel C/C++ Compiler Intrinsic Equivalent

```
VCVTUQQ2PH __m128h __mm512_cvt_roundedu64_ph (__m512i a, int rounding);
VCVTUQQ2PH __m128h __mm512_mask_cvt_roundedu64_ph (__m128h src, __mmask8 k, __m512i a, int rounding);
VCVTUQQ2PH __m128h __mm512_maskz_cvt_roundedu64_ph (__mmask8 k, __m512i a, int rounding);
VCVTUQQ2PH __m128h __mm_cvtepu64_ph (__m128i a);
VCVTUQQ2PH __m128h __mm_mask_cvtepu64_ph (__m128h src, __mmask8 k, __m128i a);
VCVTUQQ2PH __m128h __mm_maskz_cvtepu64_ph (__mmask8 k, __m128i a);
VCVTUQQ2PH __m128h __mm256_cvtepu64_ph (__m256i a);
VCVTUQQ2PH __m128h __mm256_mask_cvtepu64_ph (__m128h src, __mmask8 k, __m256i a);
VCVTUQQ2PH __m128h __mm256_maskz_cvtepu64_ph (__mmask8 k, __m256i a);
VCVTUQQ2PH __m128h __mm512_cvtepu64_ph (__m512i a);
VCVTUQQ2PH __m128h __mm512_mask_cvtepu64_ph (__m128h src, __mmask8 k, __m512i a);
VCVTUQQ2PH __m128h __mm512_maskz_cvtepu64_ph (__mmask8 k, __m512i a);
```

SIMD Floating-Point Exceptions

Overflow, Precision.

Other Exceptions

EVEX-encoded instructions, see Table 2-48, “Type E2 Class Exception Conditions.”

7. Updates to Appendix A, Volume 2D

Change bars and violet text show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D: Instruction Set Reference, W-Z*.

Changes to this chapter:

- In Section A.2.5, "Superscripts Utilized in Opcode Tables", updated the meaning of symbol 1B in Table A-1 for the UDB instruction.
- In Section A.3, "One, Two, and THREE-Byte Opcode Maps", updated Table A-2 for UDB.
- In Section A.3, "One, Two, and THREE-Byte Opcode Maps", updated Table A-3 for UD0.
- In Section A.4, "Opcode Extensions For One-Byte And Two-byte Opcodes", updated Table A-6 for UD1.

Use the opcode tables in this chapter to interpret IA-32 and Intel 64 architecture object code. Instructions are divided into encoding groups:

- 1-byte, 2-byte and 3-byte opcode encodings are used to encode integer, system, MMX technology, SSE/SSE2/SSE3/SSSE3/SSE4, and VMX instructions. Maps for these instructions are given in Table A-2 through Table A-6.
- Escape opcodes (in the format: ESC character, opcode, ModR/M byte) are used for floating-point instructions. The maps for these instructions are provided in Table A-7 through Table A-22.

NOTE

All blanks in opcode maps are reserved and must not be used. Do not depend on the operation of undefined or blank opcodes.

A.1 USING OPCODE TABLES

Tables in this appendix list opcodes of instructions (including required instruction prefixes, opcode extensions in associated ModR/M byte). Blank cells in the tables indicate opcodes that are reserved or undefined. Cells marked "Reserved-NOP" are also reserved but may behave as NOP on certain processors. Software should not use opcodes corresponding blank cells or cells marked "Reserved-NOP" nor depend on the current behavior of those opcodes.

The opcode map tables are organized by hex values of the upper and lower 4 bits of an opcode byte. For 1-byte encodings (Table A-2), use the four high-order bits of an opcode to index a row of the opcode table; use the four low-order bits to index a column of the table. For 2-byte opcodes beginning with 0FH (Table A-3), skip any instruction prefixes, the 0FH byte (0FH may be preceded by 66H, F2H, or F3H) and use the upper and lower 4-bit values of the next opcode byte to index table rows and columns. Similarly, for 3-byte opcodes beginning with 0F38H or 0F3AH (Table A-4), skip any instruction prefixes, 0F38H or 0F3AH and use the upper and lower 4-bit values of the third opcode byte to index table rows and columns. See Section A.2.4, "Opcode Look-up Examples for One, Two, and Three-Byte Opcodes."

When a ModR/M byte provides opcode extensions, this information qualifies opcode execution. For information on how an opcode extension in the ModR/M byte modifies the opcode map in Table A-2 and Table A-3, see Section A.4.

The escape (ESC) opcode tables for floating-point instructions identify the eight high order bits of opcodes at the top of each page. See Section A.5. If the accompanying ModR/M byte is in the range of 00H-BFH, bits 3-5 (the top row of the third table on each page) along with the reg bits of ModR/M determine the opcode. ModR/M bytes outside the range of 00H-BFH are mapped by the bottom two tables on each page of the section.

A.2 KEY TO ABBREVIATIONS

Operands are identified by a two-character code of the form Zz. The first character, an uppercase letter, specifies the addressing method; the second character, a lowercase letter, specifies the type of operand.

A.2.1 Codes for Addressing Method

The following abbreviations are used to document addressing methods:

- A Direct address: the instruction has no ModR/M byte; the address of the operand is encoded in the instruction. No base register, index register, or scaling factor can be applied (for example, far JMP (EA)).
- B The VEX.vvvv field of the VEX prefix selects a general purpose register.
- C The reg field of the ModR/M byte selects a control register (for example, MOV (0F20, 0F22)).

D	The reg field of the ModR/M byte selects a debug register (for example, MOV (0F21,0F23)).
E	A ModR/M byte follows the opcode and specifies the operand. The operand is either a general-purpose register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, a displacement.
F	EFLAGS/RFLAGS Register.
G	The reg field of the ModR/M byte selects a general register (for example, AX (000)).
H	The VEX.vvvv field of the VEX prefix selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type. For legacy SSE encodings this operand does not exist, changing the instruction to destructive form.
I	Immediate data: the operand value is encoded in subsequent bytes of the instruction.
J	The instruction contains a relative offset to be added to the instruction pointer register (for example, JMP (0E9), LOOP).
L	The upper 4 bits of the 8-bit immediate selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type. (the MSB is ignored in 32-bit mode)
M	The ModR/M byte may refer only to memory (for example, BOUND, LES, LDS, LSS, LFS, LGS, CMPX-CHG8B).
N	The R/M field of the ModR/M byte selects a packed-quadword, MMX technology register.
O	The instruction has no ModR/M byte. The offset of the operand is coded as a word or double word (depending on address size attribute) in the instruction. No base register, index register, or scaling factor can be applied (for example, MOV (A0-A3)).
P	The reg field of the ModR/M byte selects a packed quadword MMX technology register.
Q	A ModR/M byte follows the opcode and specifies the operand. The operand is either an MMX technology register or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.
R	The R/M field of the ModR/M byte may refer only to a general register (for example, MOV (0F20-0F23)).
S	The reg field of the ModR/M byte selects a segment register (for example, MOV (8C,8E)).
U	The R/M field of the ModR/M byte selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type.
V	The reg field of the ModR/M byte selects a 128-bit XMM register or a 256-bit YMM register, determined by operand type.
W	A ModR/M byte follows the opcode and specifies the operand. The operand is either a 128-bit XMM register, a 256-bit YMM register (determined by operand type), or a memory address. If it is a memory address, the address is computed from a segment register and any of the following values: a base register, an index register, a scaling factor, and a displacement.
X	Memory addressed by the DS:rSI register pair (for example, MOVS, CMPS, OUTS, or LODS).
Y	Memory addressed by the ES:rDI register pair (for example, MOVS, CMPS, INS, STOS, or SCAS).

A.2.2 Codes for Operand Type

The following abbreviations are used to document operand types:

- a Two one-word operands in memory or two double-word operands in memory, depending on operand-size attribute (used only by the BOUND instruction).
- b Byte, regardless of operand-size attribute.
- c Byte or word, depending on operand-size attribute.
- d Doubleword, regardless of operand-size attribute.
- dq Double-quadword, regardless of operand-size attribute.

p	32-bit, 48-bit, or 80-bit pointer, depending on operand-size attribute.
pd	128-bit or 256-bit packed double precision floating-point data.
pi	Quadword MMX technology register (for example: mm0).
ps	128-bit or 256-bit packed single precision floating-point data.
q	Quadword, regardless of operand-size attribute.
qq	Quad-Quadword (256-bits), regardless of operand-size attribute.
s	6-byte or 10-byte pseudo-descriptor.
sd	Scalar element of a 128-bit double precision floating data.
ss	Scalar element of a 128-bit single precision floating data.
si	Doubleword integer register (for example: eax).
v	Word, doubleword or quadword (in 64-bit mode), depending on operand-size attribute.
w	Word, regardless of operand-size attribute.
x	dq or qq based on the operand-size attribute.
y	Doubleword or quadword (in 64-bit mode), depending on operand-size attribute.
z	Word for 16-bit operand-size or doubleword for 32 or 64-bit operand-size.

A.2.3 Register Codes

When an opcode requires a specific register as an operand, the register is identified by name (for example, AX, CL, or ESI). The name indicates whether the register is 64, 32, 16, or 8 bits wide.

A register identifier of the form eXX or rXX is used when register width depends on the operand-size attribute. eXX is used when 16 or 32-bit sizes are possible; rXX is used when 16, 32, or 64-bit sizes are possible. For example: eAX indicates that the AX register is used when the operand-size attribute is 16 and the EAX register is used when the operand-size attribute is 32. rAX can indicate AX, EAX or RAX.

When the REX.B bit is used to modify the register specified in the reg field of the opcode, this fact is indicated by adding "/x" to the register name to indicate the additional possibility. For example, rCX/r9 is used to indicate that the register could either be rCX or r9. Note that the size of r9 in this case is determined by the operand size attribute (just as for rCX).

A.2.4 Opcode Look-up Examples for One, Two, and Three-Byte Opcodes

This section provides examples that demonstrate how opcode maps are used.

A.2.4.1 One-Byte Opcode Instructions

The opcode map for 1-byte opcodes is shown in Table A-2. The opcode map for 1-byte opcodes is arranged by row (the least-significant 4 bits of the hexadecimal value) and column (the most-significant 4 bits of the hexadecimal value). Each entry in the table lists one of the following types of opcodes:

- Instruction mnemonics and operand types using the notations listed in Section A.2
- Opcodes used as an instruction prefix

For each entry in the opcode map that corresponds to an instruction, the rules for interpreting the byte following the primary opcode fall into one of the following cases:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in Section A.1 and Chapter 2, "Instruction Format," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Operand types are listed according to notations listed in Section A.2.
- A ModR/M byte is required and includes an opcode extension in the reg field in the ModR/M byte. Use Table A-6 when interpreting the ModR/M byte.

- Use of the ModR/M byte is reserved or undefined. This applies to entries that represent an instruction prefix or entries for instructions without operands that use ModR/M (for example: 60H, PUSH A; 06H, PUSH ES).

Example A-1. Look-up Example for 1-Byte Opcodes

Opcode 030500000000H for an ADD instruction is interpreted using the 1-byte opcode map (Table A-2) as follows:

- The first digit (0) of the opcode indicates the table row and the second digit (3) indicates the table column. This locates an opcode for ADD with two operands.
- The first operand (type Gv) indicates a general register that is a word or doubleword depending on the operand-size attribute. The second operand (type Ev) indicates a ModR/M byte follows that specifies whether the operand is a word or doubleword general-purpose register or a memory address.
- The ModR/M byte for this instruction is 05H, indicating that a 32-bit displacement follows (00000000H). The reg/opcode portion of the ModR/M byte (bits 3-5) is 000, indicating the EAX register.

The instruction for this opcode is ADD EAX, mem_op, and the offset of mem_op is 00000000H.

Some 1- and 2-byte opcodes point to group numbers (shaded entries in the opcode map table). Group numbers indicate that the instruction uses the reg/opcode bits in the ModR/M byte as an opcode extension (refer to Section A.4).

A.2.4.2 Two-Byte Opcode Instructions

The two-byte opcode map shown in Table A-3 includes primary opcodes that are either two bytes or three bytes in length. Primary opcodes that are 2 bytes in length begin with an escape opcode 0FH. The upper and lower four bits of the second opcode byte are used to index a particular row and column in Table A-3.

Two-byte opcodes that are 3 bytes in length begin with a mandatory prefix (66H, F2H, or F3H) and the escape opcode (0FH). The upper and lower four bits of the third byte are used to index a particular row and column in Table A-3 (except when the second opcode byte is the 3-byte escape opcodes 38H or 3AH; in this situation refer to Section A.2.4.3).

For each entry in the opcode map, the rules for interpreting the byte following the primary opcode fall into one of the following cases:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in Section A.1 and Chapter 2, "Instruction Format," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. The operand types are listed according to notations listed in Section A.2.
- A ModR/M byte is required and includes an opcode extension in the reg field in the ModR/M byte. Use Table A-6 when interpreting the ModR/M byte.
- Use of the ModR/M byte is reserved or undefined. This applies to entries that represent an instruction without operands that are encoded using ModR/M (for example: 0F77H, EMMS).

Example A-2. Look-up Example for 2-Byte Opcodes

Look-up opcode 0FA405000000003H for a SHLD instruction using Table A-3.

- The opcode is located in row A, column 4. The location indicates a SHLD instruction with operands Ev, Gv, and Ib. Interpret the operands as follows:
 - Ev: The ModR/M byte follows the opcode to specify a word or doubleword operand.
 - Gv: The reg field of the ModR/M byte selects a general-purpose register.
 - Ib: Immediate data is encoded in the subsequent byte of the instruction.
- The third byte is the ModR/M byte (05H). The mod and opcode/reg fields of ModR/M indicate that a 32-bit displacement is used to locate the first operand in memory and EAX as the second operand.
- The next part of the opcode is the 32-bit displacement for the destination memory operand (00000000H). The last byte stores immediate byte that provides the count of the shift (03H).
- By this breakdown, it has been shown that this opcode represents the instruction: SHLD DS:00000000H, EAX, 3.

A.2.4.3 Three-Byte Opcode Instructions

The three-byte opcode maps shown in Table A-4 and Table A-5 includes primary opcodes that are either 3 or 4 bytes in length. Primary opcodes that are 3 bytes in length begin with two escape bytes 0F38H or 0F3AH. The upper and lower four bits of the third opcode byte are used to index a particular row and column in Table A-4 or Table A-5.

Three-byte opcodes that are 4 bytes in length begin with a mandatory prefix (66H, F2H, or F3H) and two escape bytes (0F38H or 0F3AH). The upper and lower four bits of the fourth byte are used to index a particular row and column in Table A-4 or Table A-5.

For each entry in the opcode map, the rules for interpreting the byte following the primary opcode fall into the following case:

- A ModR/M byte is required and is interpreted according to the abbreviations listed in A.1 and Chapter 2, “Instruction Format,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A. The operand types are listed according to notations listed in Section A.2.

Example A-3. Look-up Example for 3-Byte Opcodes

Look-up opcode 660F3A0FC108H for a PALIGNR instruction using Table A-5.

- 66H is a prefix and 0F3AH indicate to use Table A-5. The opcode is located in row 0, column F indicating a PALIGNR instruction with operands Vdq, Wdq, and Ib. Interpret the operands as follows:
 - Vdq: The reg field of the ModR/M byte selects a 128-bit XMM register.
 - Wdq: The R/M field of the ModR/M byte selects either a 128-bit XMM register or memory location.
 - Ib: Immediate data is encoded in the subsequent byte of the instruction.
- The next byte is the ModR/M byte (C1H). The reg field indicates that the first operand is XMM0. The mod shows that the R/M field specifies a register and the R/M indicates that the second operand is XMM1.
- The last byte is the immediate byte (08H).
- By this breakdown, it has been shown that this opcode represents the instruction: PALIGNR XMM0, XMM1, 8.

A.2.4.4 VEX Prefix Instructions

Instructions that include a VEX prefix are organized relative to the 2-byte and 3-byte opcode maps, based on the VEX.mmmmm field encoding of implied 0F, 0F38H, 0F3AH, respectively. Each entry in the opcode map of a VEX-encoded instruction is based on the value of the opcode byte, similar to non-VEX-encoded instructions.

A VEX prefix includes several bit fields that encode implied 66H, F2H, F3H prefix functionality (VEX.pp) and operand size/opcode information (VEX.L). See chapter 4 for details.

Opcode tables A2-A6 include both instructions with a VEX prefix and instructions without a VEX prefix. Many entries are only made once, but represent both the VEX and non-VEX forms of the instruction. If the VEX prefix is present all the operands are valid and the mnemonic is usually prefixed with a “v”. If the VEX prefix is not present the VEX.vvvv operand is not available and the prefix “v” is dropped from the mnemonic.

A few instructions exist only in VEX form and these are marked with a superscript “v”.

Operand size of VEX prefix instructions can be determined by the operand type code. 128-bit vectors are indicated by 'dq', 256-bit vectors are indicated by 'qq', and instructions with operands supporting either 128 or 256-bit, determined by VEX.L, are indicated by 'x'. For example, the entry "VMOVUPD Vx,Wx" indicates both VEX.L=0 and VEX.L=1 are supported.

A.2.5 Superscripts Utilized in Opcode Tables

Table A-1 contains notes on particular encodings. These notes are indicated in the following opcode maps by superscripts. Gray cells indicate instruction groupings.

Table A-1. Superscripts Utilized in Opcode Tables

Superscript Symbol	Meaning of Symbol
1A	Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (refer to Section A.4, "Opcode Extensions For One-Byte And Two-byte Opcodes").
1B	Use the 0F0B opcode (UD2 instruction), the 0FB9H opcode (UD1 instruction), the 0FFFH opcode (UD0 instruction), or the D6 opcode (UDB instruction) when deliberately trying to generate an invalid opcode exception (#UD).
1C	Some instructions use the same two-byte opcode. If the instruction has variations, or the opcode represents different instructions, the ModR/M byte will be used to differentiate the instruction. For the value of the ModR/M byte needed to decode the instruction, see Table A-6.
i64	The instruction is invalid or not encodable in 64-bit mode. 40 through 4F (single-byte INC and DEC) are REX prefix combinations when in 64-bit mode (use FE/FF Grp 4 and 5 for INC and DEC).
o64	Instruction is only available when in 64-bit mode.
d64	When in 64-bit mode, instruction defaults to 64-bit operand size and cannot encode 32-bit operand size.
f64	The operand size is forced to a 64-bit operand size when in 64-bit mode (prefixes that change operand size are ignored for this instruction in 64-bit mode).
v	VEX form only exists. There is no legacy SSE form of the instruction. For Integer GPR instructions it means VEX prefix required.
v1	VEX128 & SSE forms only exist (no VEX256), when can't be inferred from the data size.

A.3 ONE, TWO, AND THREE-BYTE OPCODE MAPS

See Table A-2 through Table A-5 below. The tables are multiple page presentations. Rows and columns with sequential relationships are placed on facing pages to make look-up tasks easier. Note that table footnotes are not presented on each page. Table footnotes for each table are presented on the last page of the table.

Table A-2. One-byte Opcode Map: (00H — F7H) *

	0	1	2	3	4	5	6	7
0	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	PUSH ES ⁶⁴	POP ES ⁶⁴
1	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	PUSH SS ⁶⁴	POP SS ⁶⁴
2	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	SEG=ES (Prefix)	DAA ⁶⁴
3	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, lb	rAX, lz	SEG=SS (Prefix)	AAA ⁶⁴
4	eAX REX	eCX REX.B	eDX REX.X	eBX REX.XB	eSP REX.R	eBP REX.RB	eSI REX.RX	eDI REX.RXB
5	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSHA ⁶⁴ / PUSHAD ⁶⁴	POPA ⁶⁴ / POPAD ⁶⁴	BOUND ⁶⁴ Gv, Ma	ARPL ⁶⁴ Ew, Gw MOVSD ⁶⁴ Gv, Ev	SEG=FS (Prefix)	SEG=GS (Prefix)	Operand Size (Prefix)	Address Size (Prefix)
7	O	NO	B/NAE/C	NB/AE/NC	Z/E	NZ/NE	BE/NA	NBE/A
8	Eb, lb	Ev, lz	Eb, lb ⁶⁴	Ev, lb	Eb, Gb	Ev, Gv	Eb, Gb	Ev, Gv
9	NOP PAUSE(F3) XCHG r8, rAX	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
A	AL, Ob	rAX, Ov	Ob, AL	Ov, rAX	MOVS/B Yb, Xb	MOVS/W/D/Q Yv, Xv	CMPS/B Xb, Yb	CMPS/W/D/Q Xv, Yv
B	AL/R8B, lb	CL/R9B, lb	DL/R10B, lb	BL/R11B, lb	AH/R12B, lb	CH/R13B, lb	DH/R14B, lb	BH/R15B, lb
C	Eb, lb	Ev, lb	near RET ⁶⁴ lw	near RET ⁶⁴	LES ⁶⁴ Gz, Mp VEX+2byte	LDS ⁶⁴ Gz, Mp VEX+1byte	Grp 11 ^{1A} - MOV Eb, lb Ev, lz	
D	Eb, 1	Ev, 1	Eb, CL	Ev, CL	AAM ⁶⁴ lb	AAD ⁶⁴ lb	UDB ^{1B}	XLAT/ XLATB
E	LOOPNE ⁶⁴ / LOOPNZ ⁶⁴ Jb	LOOPE ⁶⁴ / LOOPZ ⁶⁴ Jb	LOOP ⁶⁴ Jb	Jrcxz ⁶⁴ / Jb	AL, lb	IN eAX, lb	OUT lb, AL	lb, eAX
F	LOCK (Prefix)	INT1	REPNE XACQUIRE (Prefix)	REP/REPE XRELEASE (Prefix)	HLT	CMC	Unary Grp 3 ^{1A} Eb Ev	

Table A-2. One-byte Opcode Map: (08H — FFH) *

	8	9	A	B	C	D	E	F
0	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH CS ⁱ⁶⁴	2-byte escape (Table A-3)
1	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	PUSH DS ⁱ⁶⁴	POP DS ⁱ⁶⁴
2	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=CS (Prefix)	DAS ⁱ⁶⁴
3	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	AL, Ib	rAX, Iz	SEG=DS (Prefix)	AAS ⁱ⁶⁴
4	eAX REX.W	eCX REX.WB	eDX REX.WX	eBX REX.WXB	eSP REX.WR	eBP REX.WRB	eSI REX.WRX	eDI REX.WRXB
5	rAX/r8	rCX/r9	rDX/r10	rBX/r11	rSP/r12	rBP/r13	rSI/r14	rDI/r15
6	PUSH ^{d64} Iz	IMUL Gv, Ev, Iz	PUSH ^{d64} Ib	IMUL Gv, Ev, Ib	INS/ INSB Yb, DX	INS/ INSW/ INSD Yz, DX	OUTS/ OUTSB DX, Xb	OUTS/ OUTSW/ OUTSD DX, Xz
7	S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
8	Eb, Gb	Ev, Gv	Gb, Eb	Gv, Ev	MOV Ev, Sw	LEA Gv, M	MOV Sw, Ew	Grp 1A ^{1A} POP ^{d64} Ev
9	CBW/ CWDE/ CDQE	CWD/ CDQ/ CQO	far CALL ⁱ⁶⁴ Ap	FWAIT/ WAIT	PUSHF/D/Q ^{d64} / Fv	POPF/D/Q ^{d64} / Fv	SAHF	LAHF
A	TEST AL, Ib	rAX, Iz	STOS/B Yb, AL	STOS/W/D/Q Yv, rAX	LODS/B AL, Xb	LODS/W/D/Q rAX, Xv	SCAS/B AL, Yb	SCAS/W/D/Q rAX, Yv
B	rAX/r8, Iv	rCX/r9, Iv	rDX/r10, Iv	rBX/r11, Iv	rSP/r12, Iv	rBP/r13, Iv	rSI/r14, Iv	rDI/r15, Iv
C	ENTER Iw, Ib	LEAVE ^{d64}	far RET Iw	far RET	INT3	INT Ib	INTO ⁱ⁶⁴	IRET/D/Q
D	ESC (Escape to coprocessor instruction set)							
E	near CALL ^{f64} Jz	near ^{f64} Jz	JMP far ⁱ⁶⁴ Ap	short ^{f64} Jb	AL, DX	eAX, DX	DX, AL	DX, eAX
F	CLC	STC	CLI	STI	CLD	STD	INC/DEC Grp 4 ^{1A}	INC/DEC Grp 5 ^{1A}

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-3. Two-byte Opcode Map: 00H – 77H (First Byte is 0FH) *

	pxf	0	1	2	3	4	5	6	7
0		Grp 6 ^{1A}	Grp 7 ^{1A}	LAR Gv, Ew	LSL Gv, Ew		SYSCALL ^{o64}	CLTS	SYSRET ^{o64}
1		vmovups Vps, Wps	vmovups Wps, Vps	vmovlps Vq, Hq, Mq vmovhlps Vq, Hq, Uq	vmovlps Mq, Vq	vunpcklps Vx, Hx, Wx	vunpckhps Vx, Hx, Wx	vmovhps ^{v1} Vdq, Hq, Mq vmovhlps Vdq, Hq, Uq	vmovhps ^{v1} Mq, Vq
	66	vmovupd Vpd, Wpd	vmovupd Wpd, Vpd	vmovlpd Vq, Hq, Mq	vmovlpd Mq, Vq	vunpcklpd Vx, Hx, Wx	vunpckhpd Vx, Hx, Wx	vmovhpd ^{v1} Vdq, Hq, Mq	vmovhpd ^{v1} Mq, Vq
	F3	vmovss Vx, Hx, Wss	vmovss Wss, Hx, Vss	vmovsldup Vx, Wx				vmovshdup Vx, Wx	
	F2	vmovsd Vx, Hx, Wsd	vmovsd Wsd, Hx, Vsd	vmovddup Vx, Wx					
2		MOV Rd, Cd	MOV Rd, Dd	MOV Cd, Rd	MOV Dd, Rd				
3		WRMSR	RDTSC	RDMSR	RDPMC	SYSENTER	SYSEXIT		GETSEC
4		CMOVcc, (Gv, Ev) - Conditional Move							
		O	NO	B/C/NAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
5		vmovmskps Gy, Ups	vsqrtps Vps, Wps	vrsqrtps Vps, Wps	vrcpps Vps, Wps	vandps Vps, Hps, Wps	vandnps Vps, Hps, Wps	vorps Vps, Hps, Wps	vxorps Vps, Hps, Wps
	66	vmovmskpd Gy, Upd	vsqrtpd Vpd, Wpd			vandpd Vpd, Hpd, Wpd	vandnpd Vpd, Hpd, Wpd	vorpd Vpd, Hpd, Wpd	vxorpd Vpd, Hpd, Wpd
	F3		vsqrtss Vss, Hss, Wss	vrsqrtss Vss, Hss, Wss	vrcpss Vss, Hss, Wss				
	F2		vsqrtsd Vsd, Hsd, Wsd						
6		punpcklbw Pq, Qd	punpcklwd Pq, Qd	punpckldq Pq, Qd	packsswb Pq, Qq	pcmpgtb Pq, Qq	pcmpgtw Pq, Qq	pcmpgtd Pq, Qq	packuswb Pq, Qq
	66	vpunpcklbw Vx, Hx, Wx	vpunpcklwd Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpacksswb Vx, Hx, Wx	vpcmpgtb Vx, Hx, Wx	vpcmpgtw Vx, Hx, Wx	vpcmpgtd Vx, Hx, Wx	vpackuswb Vx, Hx, Wx
	F3								
7		pshufw Pq, Qq, Ib	(Grp 12 ^{1A})	(Grp 13 ^{1A})	(Grp 14 ^{1A})	pcmpeqb Pq, Qq	pcmpeqw Pq, Qq	pcmpeqd Pq, Qq	emms vzeroupper ^v vzeroall ^v
	66	vpshufd Vx, Wx, Ib				vpcmpeqb Vx, Hx, Wx	vpcmpeqw Vx, Hx, Wx	vpcmpeqd Vx, Hx, Wx	
	F3	vpshufhw Vx, Wx, Ib							
	F2	vpshufw Vx, Wx, Ib							

Table A-3. Two-byte Opcode Map: 08H — 7FH (First Byte is 0FH) *

	pxf	8	9	A	B	C	D	E	F
0		INVD	WBINVD		2-byte Illegal Opcodes UD2 ^{1B}		prefetchw(/1) Ev		
1		Prefetch ^{1C} (Grp 16 ^{1A})	Reserved-NOP	bndldx	bndstx	Reserved-NOP			NOP /0 Ev
	66			bndmov	bndmov				
	F3			bndcl	bndmk				
	F2			bndcu	bndcn				
2		vmovaps Vps, Wps	vmovaps Wps, Vps	cvtps2ps Vps, Qpi	vmovntps Mps, Vps	cvtps2pi Ppi, Wps	cvtps2pi Ppi, Wps	vucomiss Vss, Wss	vcomiss Vss, Wss
	66	vmovapd Vpd, Wpd	vmovapd Wpd, Vpd	cvtps2pd Vpd, Qpi	vmovntpd Mpd, Vpd	cvtps2pi Ppi, Wpd	cvtps2pi Qpi, Wpd	vucomisd Vsd, Wsd	vcomisd Vsd, Wsd
	F3			vcvts2ss Vss, Hss, Ey		vcvts2si Gy, Wss	vcvts2si Gy, Wss		
	F2			vcvts2sd Vsd, Hsd, Ey		vcvts2si Gy, Wsd	vcvts2si Gy, Wsd		
3		3-byte escape (Table A-4)		3-byte escape (Table A-5)					
4		CMOVcc(Gv, Ev) - Conditional Move							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
5		vaddps Vps, Hps, Wps	vmulps Vps, Hps, Wps	vcvtps2pd Vpd, Wps	vcvtdq2ps Vps, Wdq	vsubps Vps, Hps, Wps	vminps Vps, Hps, Wps	vdivps Vps, Hps, Wps	vmaxps Vps, Hps, Wps
	66	vaddpd Vpd, Hpd, Wpd	vmulpd Vpd, Hpd, Wpd	vcvtps2ps Vps, Wpd	vcvtps2dq Vdq, Wps	vsubpd Vpd, Hpd, Wpd	vminpd Vpd, Hpd, Wpd	vdivpd Vpd, Hpd, Wpd	vmaxpd Vpd, Hpd, Wpd
	F3	vaddss Vss, Hss, Wss	vmulss Vss, Hss, Wss	vcvts2sd Vsd, Hx, Wss	vcvtps2dq Vdq, Wps	vsubss Vss, Hss, Wss	vminss Vss, Hss, Wss	vdivss Vss, Hss, Wss	vmaxss Vss, Hss, Wss
	F2	vaddsd Vsd, Hsd, Wsd	vmulsd Vsd, Hsd, Wsd	vcvts2sd Vss, Hx, Wsd		vsubsd Vsd, Hsd, Wsd	vminsd Vsd, Hsd, Wsd	vdivsd Vsd, Hsd, Wsd	vmaxsd Vsd, Hsd, Wsd
6		punpckhbw Pq, Qd	punpckhwd Pq, Qd	punpckhdq Pq, Qd	packssdw Pq, Qd			movd/q Pd, Ey	movq Pq, Qq
	66	vpunpckhbw Vx, Hx, Wx	vpunpckhwd Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vpackssdw Vx, Hx, Wx	vpunpckldq Vx, Hx, Wx	vpunpckhdq Vx, Hx, Wx	vmovd/q Vy, Ey	vmovdqa Vx, Wx
	F3								vmovdqu Vx, Wx
7		VMREAD Ey, Gy	VMWRITE Gy, Ey					movd/q Ey, Pd	movq Qq, Pq
	66					vhaddpd Vpd, Hpd, Wpd	vhsbpd Vpd, Hpd, Wpd	vmovd/q Ey, Vy	vmovdqa Wx, Vx
	F3							vmovq Vq, Wq	vmovdqu Wx, Vx
	F2					vhaddps Vps, Hps, Wps	vhsbps Vps, Hps, Wps		

Table A-3. Two-byte Opcode Map: 80H — F7H (First Byte is 0FH) *

	pxf	0	1	2	3	4	5	6	7
8		Jcc ^{f64} , Jz - Long-displacement jump on condition							
		O	NO	B/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
9		SETcc, Eb - Byte Set on condition							
		O	NO	B/CNAE	AE/NB/NC	E/Z	NE/NZ	BE/NA	A/NBE
A		PUSH ^{d64} FS	POP ^{d64} FS	CPUID	BT Ev, Gv	SHLD Ev, Gv, Ib	SHLD Ev, Gv, CL		
B		CMPXCHG Eb, Gb		LSS Gv, Mp	BTR Ev, Gv	LFS Gv, Mp	LGS Gv, Mp	MOVZX Gv, Eb Gv, Ew	
C		XADD Eb, Gb	XADD Ev, Gv	vcmpps Vps,Hps,Wps,Ib	movnti My, Gy	pinsrw Pq,Ry/Mw,Ib	pextrw Gd, Nq, Ib	vshufps Vps,Hps,Wps,Ib	Grp 9 ^{1A}
	66			vcmppd Vpd,Hpd,Wpd,Ib		vpinsrw Vdq,Hdq,Ry/Mw,Ib	vpextrw Gd, Udq, Ib	vshufpd Vpd,Hpd,Wpd,Ib	
	F3			vcmpss Vss,Hss,Wss,Ib					
	F2			vcmpsdb Vsd,Hsd,Wsd,Ib					
D			psrlw Pq, Qq	psrld Pq, Qq	psrlq Pq, Qq	paddq Pq, Qq	pmullw Pq, Qq		pmovmskb Gd, Nq
	66	vaddsubpd Vpd, Hpd, Wpd	vpsrlw Vx, Hx, Wx	vpsrld Vx, Hx, Wx	vpsrlq Vx, Hx, Wx	vpaddq Vx, Hx, Wx	vpmullw Vx, Hx, Wx	vmovq Wq, Vq	vpmovmskb Gd, Ux
	F3							movq2dq Vdq, Nq	
	F2	vaddsubps Vps, Hps, Wps						movdq2q Pq, Uq	
E		pavgb Pq, Qq	psraw Pq, Qq	psrad Pq, Qq	pavgw Pq, Qq	pmulhuw Pq, Qq	pmulhw Pq, Qq		movntq Mq, Pq
	66	vpavgb Vx, Hx, Wx	vpsraw Vx, Hx, Wx	vpsrad Vx, Hx, Wx	vpavgw Vx, Hx, Wx	vpmulhuw Vx, Hx, Wx	vpmulhw Vx, Hx, Wx	vcvttd2dq Vx, Wpd	vmovntdq Mx, Vx
	F3							vcvtdq2pd Vx, Wpd	
	F2							vcvtpd2dq Vx, Wpd	
F			psllw Pq, Qq	pslld Pq, Qq	psllq Pq, Qq	pmuludq Pq, Qq	pmaddwd Pq, Qq	psadbw Pq, Qq	maskmovq Pq, Nq
	66		vpsllw Vx, Hx, Wx	vpslld Vx, Hx, Wx	vpsllq Vx, Hx, Wx	vpmuludq Vx, Hx, Wx	vpmaddwd Vx, Hx, Wx	vpsadbw Vx, Hx, Wx	vmaskmovdqu Vdq, Udq
	F2	vlddqu Vx, Mx							

Table A-3. Two-byte Opcode Map: 88H — FFH (First Byte is 0FH) *

	pxf	8	9	A	B	C	D	E	F
8		Jcc ⁶⁴ , Jz - Long-displacement jump on condition							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
9		SETcc, Eb - Byte Set on condition							
		S	NS	P/PE	NP/PO	L/NGE	NL/GE	LE/NG	NLE/G
A		PUSH ^{d64} GS	POP ^{d64} GS	RSM	BTS Ev, Gv	SHRD Ev, Gv, Ib	SHRD Ev, Gv, CL	(Grp 15 ^{1A}) ^{1C}	IMUL Gv, Ev
B		JMPE (reserved for emulator on IPF)	Grp 10 ^{1A} Invalid Opcode ^{1B}	Grp 8 ^{1A} Ev, Ib	BTC Ev, Gv	BSF Gv, Ev	BSR Gv, Ev	MOVsx Gv, Eb Gv, Ew	
	F3	POPCNT Gv, Ev				TZCNT Gv, Ev	LZCNT Gv, Ev		
C		BSWAP							
		RAX/EAX/ R8/R8D	RCX/ECX/ R9/R9D	RDX/EDX/ R10/R10D	RBX/EBX/ R11/R11D	RSP/ESP/ R12/R12D	RBP/EBP/ R13/R13D	RSI/ESI/ R14/R14D	RDI/EDI/ R15/R15D
D		psubusb Pq, Qq	psubusw Pq, Qq	pminub Pq, Qq	pand Pq, Qq	paddusb Pq, Qq	paddusw Pq, Qq	pmaxub Pq, Qq	pandn Pq, Qq
	66	vpsubusb Vx, Hx, Wx	vpsubusw Vx, Hx, Wx	vpminub Vx, Hx, Wx	vpand Vx, Hx, Wx	vpaddusb Vx, Hx, Wx	vpaddusw Vx, Hx, Wx	vpmaxub Vx, Hx, Wx	vpandn Vx, Hx, Wx
	F3								
	F2								
E		psubsb Pq, Qq	psubsw Pq, Qq	pminsw Pq, Qq	por Pq, Qq	paddsb Pq, Qq	paddsw Pq, Qq	pmaxsw Pq, Qq	pxor Pq, Qq
	66	vpsubsb Vx, Hx, Wx	vpsubsw Vx, Hx, Wx	vpminsw Vx, Hx, Wx	vpior Vx, Hx, Wx	vpaddsb Vx, Hx, Wx	vpaddsw Vx, Hx, Wx	vpmaxsw Vx, Hx, Wx	vpxor Vx, Hx, Wx
	F3								
	F2								
F		psubb Pq, Qq	psubw Pq, Qq	psubd Pq, Qq	psubq Pq, Qq	paddb Pq, Qq	paddw Pq, Qq	paddd Pq, Qq	UD0 ^{1B}
	66	vpsubb Vx, Hx, Wx	vpsubw Vx, Hx, Wx	vpsubd Vx, Hx, Wx	vpsubq Vx, Hx, Wx	vpaddb Vx, Hx, Wx	vpaddw Vx, Hx, Wx	vpaddd Vx, Hx, Wx	
	F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-4. Three-byte Opcode Map: 00H — F7H (First Two Bytes are 0F 38H) *

	pxf	0	1	2	3	4	5	6	7
0		pshufb Pq, Qq	phaddw Pq, Qq	phadd Pq, Qq	phaddsw Pq, Qq	pmaddubsw Pq, Qq	phsubw Pq, Qq	phsubd Pq, Qq	phsubsw Pq, Qq
	66	vpshufb Vx, Hx, Wx	vphaddw Vx, Hx, Wx	vphadd Vx, Hx, Wx	vphaddsw Vx, Hx, Wx	vpmaddubsw Vx, Hx, Wx	vphsubw Vx, Hx, Wx	vphsubd Vx, Hx, Wx	vphsubsw Vx, Hx, Wx
1		pblendvb Vdq, Wdq			vcvtph2ps ^V Vx, Wx, lb	blendvps Vdq, Wdq	blendvpd Vdq, Wdq	vpermps ^V Vqq, Hqq, Wqq	vptest Vx, Wx
	66								
2	66	vpmovsxbw Vx, Ux/Mq	vpmovsxbd Vx, Ux/Md	vpmovsxbq Vx, Ux/Mw	vpmovsxd Vx, Ux/Mq	vpmovsxwq Vx, Ux/Md	vpmovsxdq Vx, Ux/Mq		
3	66	vpmovzxbw Vx, Ux/Mq	vpmovzxbd Vx, Ux/Md	vpmovzxbq Vx, Ux/Mw	vpmovzxd Vx, Ux/Mq	vpmovzxwq Vx, Ux/Md	vpmovzxdq Vx, Ux/Mq	vpermd ^V Vqq, Hqq, Wqq	vpcmpgtq Vx, Hx, Wx
4	66	vpmulld Vx, Hx, Wx	vphminposuw Vdq, Wdq				vpsrlvd/q ^V Vx, Hx, Wx	vpsravd ^V Vx, Hx, Wx	vpslvd/q ^V Vx, Hx, Wx
5									
6									
7									
8	66	INVEPT Gy, Mdq	INVVPID Gy, Mdq	INVPCID Gy, Mdq					
9	66	vgatherdd/q ^V Vx,Hx,Wx	vgatherqd/q ^V Vx,Hx,Wx	vgatherdps/d ^V Vx,Hx,Wx	vgatherqps/d ^V Vx,Hx,Wx			vfmaddsub132ps/d ^V Vx,Hx,Wx	vfmsubadd132ps/d ^V Vx,Hx,Wx
A	66							vfmaddsub213ps/d ^V Vx,Hx,Wx	vfmsubadd213ps/d ^V Vx,Hx,Wx
B	66							vfmaddsub231ps/d ^V Vx,Hx,Wx	vfmsubadd231ps/d ^V Vx,Hx,Wx
C									
D									
E									
F		MOVBE Gy, My	MOVBE My, Gy	ANDN ^V Gy, By, Ey	Grp 17 ^{1A}		BZHI ^V Gy, Ey, By		BEXTR ^V Gy, Ey, By
	66	MOVBE Gw, Mw	MOVBE Mw, Gw					ADCX Gy, Ey	SHLX ^V Gy, Ey, By
	F3						PEXT ^V Gy, By, Ey	ADOX Gy, Ey	SARX ^V Gy, Ey, By
	F2	CRC32 Gd, Eb	CRC32 Gd, Ey				PDEP ^V Gy, By, Ey	MULX ^V By,Gy,rDX,Ey	SHRX ^V Gy, Ey, By
	66 & F2	CRC32 Gd, Eb	CRC32 Gd, Ew						

Table A-4. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 38H) *

	pfx	8	9	A	B	C	D	E	F
0		psignb Pq, Qq	psignw Pq, Qq	psignd Pq, Qq	pmulhrsw Pq, Qq				
	66	vpsignb Vx, Hx, Wx	vpsignw Vx, Hx, Wx	vpsignd Vx, Hx, Wx	vpmulhrsw Vx, Hx, Wx	vpermilps ^V Vx,Hx,Wx	vpermilpd ^V Vx,Hx,Wx	vtestps ^V Vx, Wx	vtestpd ^V Vx, Wx
1						pabsb Pq, Qq	pabsw Pq, Qq	pabsd Pq, Qq	
	66	vbroadcastss ^V Vx, Wd	vbroadcastsd ^V Vqq, Wq	vbroadcastf128 ^V Vqq, Mdq		vpabsb Vx, Wx	vpabsw Vx, Wx	vpabsd Vx, Wx	
2	66	vpmuldq Vx, Hx, Wx	vpcmpqq Vx, Hx, Wx	vmovntdqa Vx, Mx	vpackusdw Vx, Hx, Wx	vmaskmovps ^V Vx,Hx,Mx	vmaskmovpd ^V Vx,Hx,Mx	vmaskmovps ^V Mx,Hx,Vx	vmaskmovpd ^V Mx,Hx,Vx
3	66	vpminsb Vx, Hx, Wx	vpminsd Vx, Hx, Wx	vpminuw Vx, Hx, Wx	vpminud Vx, Hx, Wx	vpmaxsb Vx, Hx, Wx	vpmaxsd Vx, Hx, Wx	vpmaxuw Vx, Hx, Wx	vpmaxud Vx, Hx, Wx
4									
5	66	vpbroadcast ^V Vx, Wx	vpbroadcastq ^V Vx, Wx	vpbroadcasti128 ^V Vqq, Mdq					
6									
7	66	vpbroadcastb ^V Vx, Wx	vpbroadcastw ^V Vx, Wx						
8	66					vpmaskmovd/q ^V Vx,Hx,Mx		vpmaskmovd/q ^V Mx,Vx,Hx	
9	66	vfmadd132ps/d ^V Vx, Hx, Wx	vfmadd132ss/d ^V Vx, Hx, Wx	vfmsub132ps/d ^V Vx, Hx, Wx	vfmsub132ss/d ^V Vx, Hx, Wx	vfnmadd132ps/d ^V Vx, Hx, Wx	vfnmadd132ss/d ^V Vx, Hx, Wx	vfmsub132ps/d ^V Vx, Hx, Wx	vfmsub132ss/d ^V Vx, Hx, Wx
A	66	vfmadd213ps/d ^V Vx, Hx, Wx	vfmadd213ss/d ^V Vx, Hx, Wx	vfmsub213ps/d ^V Vx, Hx, Wx	vfmsub213ss/d ^V Vx, Hx, Wx	vfnmadd213ps/d ^V Vx, Hx, Wx	vfnmadd213ss/d ^V Vx, Hx, Wx	vfmsub213ps/d ^V Vx, Hx, Wx	vfmsub213ss/d ^V Vx, Hx, Wx
B	66	vfmadd231ps/d ^V Vx, Hx, Wx	vfmadd231ss/d ^V Vx, Hx, Wx	vfmsub231ps/d ^V Vx, Hx, Wx	vfmsub231ss/d ^V Vx, Hx, Wx	vfnmadd231ps/d ^V Vx, Hx, Wx	vfnmadd231ss/d ^V Vx, Hx, Wx	vfmsub231ps/d ^V Vx, Hx, Wx	vfmsub231ss/d ^V Vx, Hx, Wx
C		sha1nexte Vdq,Wdq	sha1msg1 Vdq,Wdq	sha1msg2 Vdq,Wdq	sha256rds2 Vdq,Wdq	sha256msg1 Vdq,Wdq	sha256msg2 Vdq,Wdq		
	66								
D	66				VAESIMC Vdq, Wdq	VAESEC Vdq,Hdq,Wdq	VAESENCLAST Vdq,Hdq,Wdq	VAESDEC Vdq,Hdq,Wdq	VAESDECLAST Vdq,Hdq,Wdq
E									
F									
	66								
	F3								
	F2								
	66 & F2								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-5. Three-byte Opcode Map: 00H — F7H (First two bytes are 0F 3AH) *

	px	0	1	2	3	4	5	6	7
0	66	vpermq ^v Vqq, Wqq, lb	vpermpd ^v Vqq, Wqq, lb	vpblendd ^v Vx,Hx,Wx,lb		vpermilps ^v Vx, Wx, lb	vpermilpd ^v Vx, Wx, lb	vperm2f128 ^v Vqq,Hqq,Wqq,lb	
1	66					vpextrb Rd/Mb, Vdq, lb	vpextrw Rd/Mw, Vdq, lb	vpextrd/q Ey, Vdq, lb	vextractps Ed, Vdq, lb
2	66	vpinsrb Vdq,Hdq,Ry/Mb,lb	vinsertps Vdq,Hdq,Udq/Md,lb	vpinsrd/q Vdq,Hdq,Ey,lb					
3									
4	66	vdpps Vx,Hx,Wx,lb	vdppd Vdq,Hdq,Wdq,lb	vmepsadbw Vx,Hx,Wx,lb		vpcimulqdd Vdq,Hdq,Wdq,lb		vperm2i128 ^v Vqq,Hqq,Wqq,lb	
5									
6	66	vpcmpestrm Vdq, Wdq, lb	vpcmpestri Vdq, Wdq, lb	vpcmpistrm Vdq, Wdq, lb	vpcmpistri Vdq, Wdq, lb				
7									
8									
9									
A									
B									
C									
D									
E									
F	F2	RORX ^v Gy, Ey, lb							

Table A-5. Three-byte Opcode Map: 08H — FFH (First Two Bytes are 0F 3AH) *

	px	8	9	A	B	C	D	E	F
0									palignr Pq, Qq, Ib
	66	vroundps Vx, Wx, Ib	vroundpd Vx, Wx, Ib	vroundss Vss, Wss, Ib	vroundsd Vsd, Wsd, Ib	vblendps Vx, Hx, Wx, Ib	vblendpd Vx, Hx, Wx, Ib	vpblendw Vx, Hx, Wx, Ib	vpalignr Vx, Hx, Wx, Ib
1	66	vinser128 ^v Vqq, Hqq, Wqq, Ib	vextractf128 ^v Wdq, Vqq, Ib				vcvtps2ph ^v Wx, Vx, Ib		
2									
3	66	vinser128 ^v Vqq, Hqq, Wqq, Ib	vextractf128 ^v Wdq, Vqq, Ib						
4	66			vblendvps ^v Vx, Hx, Wx, Lx	vblendvpd ^v Vx, Hx, Wx, Lx	vpblendvb ^v Vx, Hx, Wx, Lx			
5									
6									
7									
8									
9									
A									
B									
C						sha1m4 Vdq, Wdq, Ib			
D	66								VAESKEYGEN Vdq, Wdq, Ib
E									
F									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.4 OPCODE EXTENSIONS FOR ONE-BYTE AND TWO-BYTE OPCODES

Some 1-byte and 2-byte opcodes use bits 3-5 of the ModR/M byte (the nnn field in Figure A-1) as an extension of the opcode.

mod	nnn	R/M

Figure A-1. ModR/M Byte nnn Field (Bits 5, 4, and 3)

Opcodes that have opcode extensions are indicated in Table A-6 and organized by group number. Group numbers (from 1 to 16, second column) provide a table entry point. The encoding for the r/m field for each instruction can be established using the third column of the table.

A.4.1 Opcode Look-up Examples Using Opcode Extensions

An Example is provided below.

Example A-4. Interpreting an ADD Instruction

An ADD instruction with a 1-byte opcode of 80H is a Group 1 instruction:

- Table A-6 indicates that the opcode extension field encoded in the ModR/M byte for this instruction is 000B.
- The r/m field can be encoded to access a register (11B) or a memory address using a specified addressing mode (for example: mem = 00B, 01B, 10B).

Example A-5. Looking Up 0F01C3H

Look up opcode 0F01C3 for a VMRESUME instruction by using Table A-2, Table A-3, and Table A-6:

- 0F indicates that this instruction is in the 2-byte opcode map.
- 01 (row 0, column 1 in Table A-3) reveals that this opcode is in Group 7 of Table A-6.
- C3 is the ModR/M byte. The first two bits of C3 are 11B. This tells us to look at the second of the Group 7 rows in Table A-6.
- The Op/Reg bits [5,4,3] are 000B. This tells us to look in the 000 column for Group 7.
- Finally, the R/M bits [2,1,0] are 011B. This identifies the opcode as the VMRESUME instruction.

A.4.2 Opcode Extension Tables

See Table A-6 below.

Table A-6. Opcode Extensions for One- and Two-byte Opcodes by Group Number *

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
80-83	1	mem, 11B		ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
8F	1A	mem, 11B		POP							
C0,C1 reg, imm D0, D1 reg, 1 D2, D3 reg, CL	2	mem, 11B		ROL	ROR	RCL	RCR	SHL/SAL	SHR		SAR
F6, F7	3	mem, 11B		TEST lb/lz		NOT	NEG	MUL AL/rAX	IMUL AL/rAX	DIV AL/rAX	IDIV AL/rAX
FE	4	mem, 11B		INC Eb	DEC Eb						
FF	5	mem, 11B		INC Ev	DEC Ev	near CALL ^{f64} Ev	far CALL Ep	near JMP ^{f64} Ev	far JMP Mp	PUSH ^{d64} Ev	
0F 00	6	mem, 11B		SLDT Rv/Mw	STR Rv/Mw	LLDT Ew	LTR Ew	VERR Ew	VERW Ew		
0F 01	7	mem		SGDT Ms	SIDT Ms	LGDT Ms	LIDT Ms	SMSW Mw/Rv		LMSW Ew	INVLPG Mb
		11B		VMCALL (001) VMLAUNCH (010) VMRESUME (011) VMXOFF (100)	MONITOR (000) MWAIT (001) CLAC (010) STAC (011) ENCLS (111)	XGETBV (000) XSETBV (001) VMFUNC (100) XEND (101) XTEST (110) ENCLU(111)				SWAPGS ^{o64} (000) RDTSCP (001)	
0F BA	8	mem, 11B						BT	BTS	BTR	BTC
0F C7	9	mem			CMPXCH8B Mq CMPXCHG16B Mdq					VMPTRLD Mq	VMPTRST Mq
			66							VMCLEAR Mq	
			F3							VMXON Mq	
		11B							RDRAND Rv	RDSEED Rv	
		F3								RDPID Rd/q	
0F B9	10	mem		UD1 ^{1B}							
		11B									
C6	11	mem		MOV Eb, lb							
11B										XABORT (000) lb	
C7		mem		MOV Ev, lz							
		11B								XBEGIN (000) Jz	
0F 71	12	mem									
		11B			psrlw Nq, lb		psraw Nq, lb		psllw Nq, lb		
		66			vpsrlw Hx,Ux,lb		vpsraw Hx,Ux,lb		vpsllw Hx,Ux,lb		
0F 72	13	mem									
		11B			psrld Nq, lb		psrad Nq, lb		pslld Nq, lb		
			66		vpsrld Hx,Ux,lb		vpsrad Hx,Ux,lb		vpslld Hx,Ux,lb		
0F 73	14	mem									
		11B			psrlq Nq, lb				psllq Nq, lb		
			66		vpsrlq Hx,Ux,lb		vpsrldq Hx,Ux,lb		vpsllq Hx,Ux,lb	vpslldq Hx,Ux,lb	

Table A-6. Opcode Extensions for One- and Two-byte Opcodes by Group Number * (Contd.)

Opcode	Group	Mod 7,6	pfx	Encoding of Bits 5,4,3 of the ModR/M Byte (bits 2,1,0 in parenthesis)							
				000	001	010	011	100	101	110	111
0F AE	15	mem		fxsave	fxrstor	ldmxcsr	stmxcsr	XSAVE	XRSTOR	XSAVEOPT	clflush
		11B mem							lfence	mfence	sfence
			F3	RDFSBASE Ry	RDGSBASE Ry	WRFSBASE Ry	WRGSBASE Ry				
0F 18	16			prefetch NTA	prefetch T0	prefetch T1	prefetch T2	Reserved NOP			
		11B		Reserved NOP							
VEX.0F38 F3	17	mem			BLSR ^v By, Ey	BLSMSK ^v By, Ey	BLSI ^v By, Ey				
		11B									

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5 ESCAPE OPCODE INSTRUCTIONS

Opcode maps for coprocessor escape instruction opcodes (x87 floating-point instruction opcodes) are in Table A-7 through Table A-22. These maps are grouped by the first byte of the opcode, from D8-DF. Each of these opcodes has a ModR/M byte. If the ModR/M byte is within the range of 00H-BFH, bits 3-5 of the ModR/M byte are used as an opcode extension, similar to the technique used for 1- and 2-byte opcodes (see A.4). If the ModR/M byte is outside the range of 00H through BFH, the entire ModR/M byte is used as an opcode extension.

A.5.1 Opcode Look-up Examples for Escape Instruction Opcodes

Examples are provided below.

Example A-6. Opcode with ModR/M Byte in the 00H through BFH Range

DD0504000000H can be interpreted as follows:

- The instruction encoded with this opcode can be located in Section . Since the ModR/M byte (05H) is within the 00H through BFH range, bits 3 through 5 (000) of this byte indicate the opcode for an FLD double-real instruction (see Table A-9).
- The double-real value to be loaded is at 00000004H (the 32-bit displacement that follows and belongs to this opcode).

Example A-7. Opcode with ModR/M Byte outside the 00H through BFH Range

D8C1H can be interpreted as follows:

- This example illustrates an opcode with a ModR/M byte outside the range of 00H through BFH. The instruction can be located in Section A.4.
- In Table A-8, the ModR/M byte C1H indicates row C, column 1 (the FADD instruction using ST(0), ST(1) as operands).

A.5.2 Escape Opcode Instruction Tables

Tables are listed below.

A.5.2.1 Escape Opcodes with D8 as First Byte

Table A-7 and A-8 contain maps for the escape instruction opcodes that begin with D8H. Table A-7 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-7. D8 Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte (refer to Figure A.4)							
000B	001B	010B	011B	100B	101B	110B	111B
FADD single-real	FMUL single-real	FCOM single-real	FCOMP single-real	FSUB single-real	FSUBR single-real	FDIV single-real	FDIVR single-real

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-8 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-8. D8 Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FADD							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCOM							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),T(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FSUB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F	FDIV							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C	FMUL							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCOMP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),T(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FSUBR							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F	FDIVR							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.2 Escape Opcodes with D9 as First Byte

Table A-9 and A-10 contain maps for escape instruction opcodes that begin with D9H. Table A-9 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-9. D9 Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FLD single-real		FST single-real	FSTP single-real	FLDENV 14/28 bytes	FLDCW 2 bytes	FSTENV 14/28 bytes	FSTCW 2 bytes

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-10 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-10. D9 Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FLD							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FNOP							
E	FCHS	FABS			FTST	FXAM		
F	F2XM1	FYL2X	FPTAN	FPATAN	EXTRACT	FPREM1	FDECSTP	FINCSTP

	8	9	A	B	C	D	E	F
C	FXCH							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D								
E	FLD1	FLDL2T	FLDL2E	FLDPI	FLDLG2	FLDLN2	FLDZ	
F	FPREM	FYL2XP1	FSQRT	FSINCOS	FRNDINT	FSCALE	FSIN	FCOS

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.3 Escape Opcodes with DA as First Byte

Table A-11 and A-12 contain maps for escape instruction opcodes that begin with DAH. Table A-11 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-11. DA Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FIADD dword-integer	FIMUL dword-integer	FICOM dword-integer	FICOMP dword-integer	FISUB dword-integer	FISUBR dword-integer	FIDIV dword-integer	FIDIVR dword-integer

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-12 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-12. DA Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FCMOVB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVBE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E								
F								

	8	9	A	B	C	D	E	F
C	FCMOVE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVU							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E		FUCOMPP						
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.4 Escape Opcodes with DB as First Byte

Table A-13 and A-14 contain maps for escape instruction opcodes that begin with DBH. Table A-13 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-13. DB Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FILD dword-integer	FISTTP dword-integer	FIST dword-integer	FISTP dword-integer		FLD extended-real		FSTP extended-real

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-14 shows the map if the ModR/M byte is outside the range of 00H-BFH. Here, the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-14. DB Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FCMOVNB							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVNBE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E			FCLEX	FINIT				
F	FCOMI							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C	FCMOVNE							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
D	FCMOVNU							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
E	FUCOMI							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.5 Escape Opcodes with DC as First Byte

Table A-15 and A-16 contain maps for escape instruction opcodes that begin with DCH. Table A-15 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-15. DC Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte (refer to Figure A-1)							
000B	001B	010B	011B	100B	101B	110B	111B
FADD double-real	FMUL double-real	FCOM double-real	FCOMP double-real	FSUB double-real	FSUBR double-real	FDIV double-real	FDIVR double-real

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-16 shows the map if the ModR/M byte is outside the range of 00H-BFH. In this case the first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-16. DC Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FADD							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUBR							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVR							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

	8	9	A	B	C	D	E	F
C	FMUL							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUB							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIV							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.6 Escape Opcodes with DD as First Byte

Table A-17 and A-18 contain maps for escape instruction opcodes that begin with DDH. Table A-17 shows the map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-17. DD Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FLD double-real	FISTTP integer64	FST double-real	FSTP double-real	FRSTOR 98/108bytes		FSAVE 98/108bytes	FSTSW 2 bytes

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-18 shows the map if the ModR/M byte is outside the range of 00H-BFH. The first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-18. DD Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FFREE							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
D	FST							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
E	FUCOM							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F								

	8	9	A	B	C	D	E	F
C								
D	FSTP							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
E	FUCOMP							
	ST(0)	ST(1)	ST(2)	ST(3)	ST(4)	ST(5)	ST(6)	ST(7)
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.7 Escape Opcodes with DE as First Byte

Table A-19 and A-20 contain opcode maps for escape instruction opcodes that begin with DEH. Table A-19 shows the opcode map if the ModR/M byte is in the range of 00H-BFH. In this case, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-19. DE Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FIADD word-integer	FIMUL word-integer	FICOM word-integer	FICOMP word-integer	FISUB word-integer	FISUBR word-integer	FIDIV word-integer	FIDIVR word-integer

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-20 shows the opcode map if the ModR/M byte is outside the range of 00H-BFH. The first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-20. DE Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C	FADDP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D								
E	FSUBRP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVRP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

	8	9	A	B	C	D	E	F
C	FMULP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
D		FCOMPP						
E	FSUBP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)
F	FDIVP							
	ST(0),ST(0)	ST(1),ST(0)	ST(2),ST(0)	ST(3),ST(0)	ST(4),ST(0)	ST(5),ST(0)	ST(6),ST(0)	ST(7),ST(0)

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

A.5.2.8 Escape Opcodes with DF As First Byte

Table A-21 and A-22 contain the opcode maps for escape instruction opcodes that begin with DFH. Table A-21 shows the opcode map if the ModR/M byte is in the range of 00H-BFH. Here, the value of bits 3-5 (the nnn field in Figure A-1) selects the instruction.

Table A-21. DF Opcode Map When ModR/M Byte is Within 00H to BFH *

nnn Field of ModR/M Byte							
000B	001B	010B	011B	100B	101B	110B	111B
FILD word-integer	FISTTP word-integer	FIST word-integer	FISTP word-integer	FBLD packed-BCD	FILD qword-integer	FBSTP packed-BCD	FISTP qword-integer

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

Table A-22 shows the opcode map if the ModR/M byte is outside the range of 00H-BFH. The first digit of the ModR/M byte selects the table row and the second digit selects the column.

Table A-22. DF Opcode Map When ModR/M Byte is Outside 00H to BFH *

	0	1	2	3	4	5	6	7
C								
D								
E	FSTSW AX							
F	FCOMIP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)

	8	9	A	B	C	D	E	F
C								
D								
E	FUCOMIP							
	ST(0),ST(0)	ST(0),ST(1)	ST(0),ST(2)	ST(0),ST(3)	ST(0),ST(4)	ST(0),ST(5)	ST(0),ST(6)	ST(0),ST(7)
F								

NOTES:

* All blanks in all opcode maps are reserved and must not be used. Do not depend on the operation of undefined or reserved locations.

8. Updates to Chapter 2, Volume 3A

Change bars and violet text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Added footnote in Section 2.5, "Control Registers" for the MAXPHYADDR regarding SEAM for Bits in CR3 in the range 63:MAXPHYADDR.

IA-32 architecture (beginning with the Intel386 processor family) provides extensive support for operating-system and system-development software. This support offers multiple modes of operation, which include:

- Real mode, protected mode, virtual 8086 mode, and system management mode. These are sometimes referred to as legacy modes.

Intel 64 architecture supports almost all the system programming facilities available in IA-32 architecture and extends them to a new operating mode (IA-32e mode) that supports a 64-bit programming environment. IA-32e mode allows software to operate in one of two sub-modes:

- 64-bit mode supports 64-bit OS and 64-bit applications
- Compatibility mode allows most legacy software to run; it co-exists with 64-bit applications under a 64-bit OS.

The IA-32 system-level architecture includes features to assist in the following operations:

- Memory management.
- Protection of software modules.
- Multitasking.
- Exception and interrupt handling.
- Multiprocessing.
- Cache management.
- Hardware resource and power management.
- Debugging and performance monitoring.

This chapter provides a description of each part of this architecture. It also describes the system registers that are used to set up and control the processor at the system level and gives a brief overview of the processor's system-level (operating system) instructions.

Many features of the system-level architecture are used only by system programmers. However, application programmers may need to read this chapter and the following chapters in order to create a reliable and secure environment for application programs.

This overview and most subsequent chapters of this book focus on protected-mode operation of the IA-32 architecture. IA-32e mode operation of the Intel 64 architecture, as it differs from protected mode operation, is also described.

All Intel 64 and IA-32 processors enter real-address mode following a power-up or reset (see Chapter 11, "Processor Management and Initialization"). Software then initiates the switch from real-address mode to protected mode. If IA-32e mode operation is desired, software also initiates a switch from protected mode to IA-32e mode.

2.1 OVERVIEW OF THE SYSTEM-LEVEL ARCHITECTURE

System-level architecture consists of a set of registers, data structures, and instructions designed to support basic system-level operations such as memory management, interrupt and exception handling, task management, and control of multiple processors.

Figure 2-1 provides a summary of system registers and data structures that applies to 32-bit modes. System registers and data structures that apply to IA-32e mode are shown in Figure 2-2.

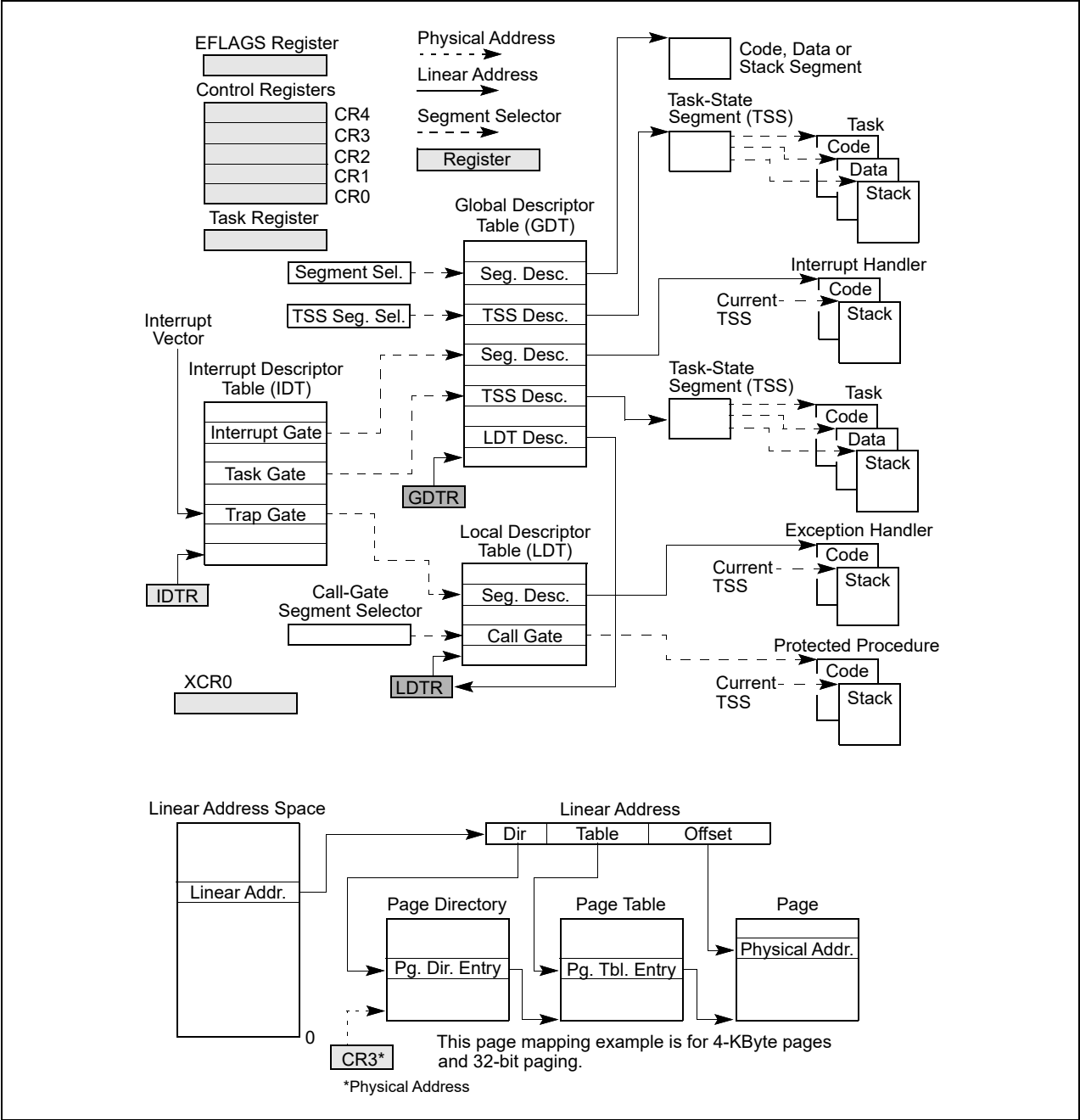


Figure 2-1. IA-32 System-Level Registers and Data Structures

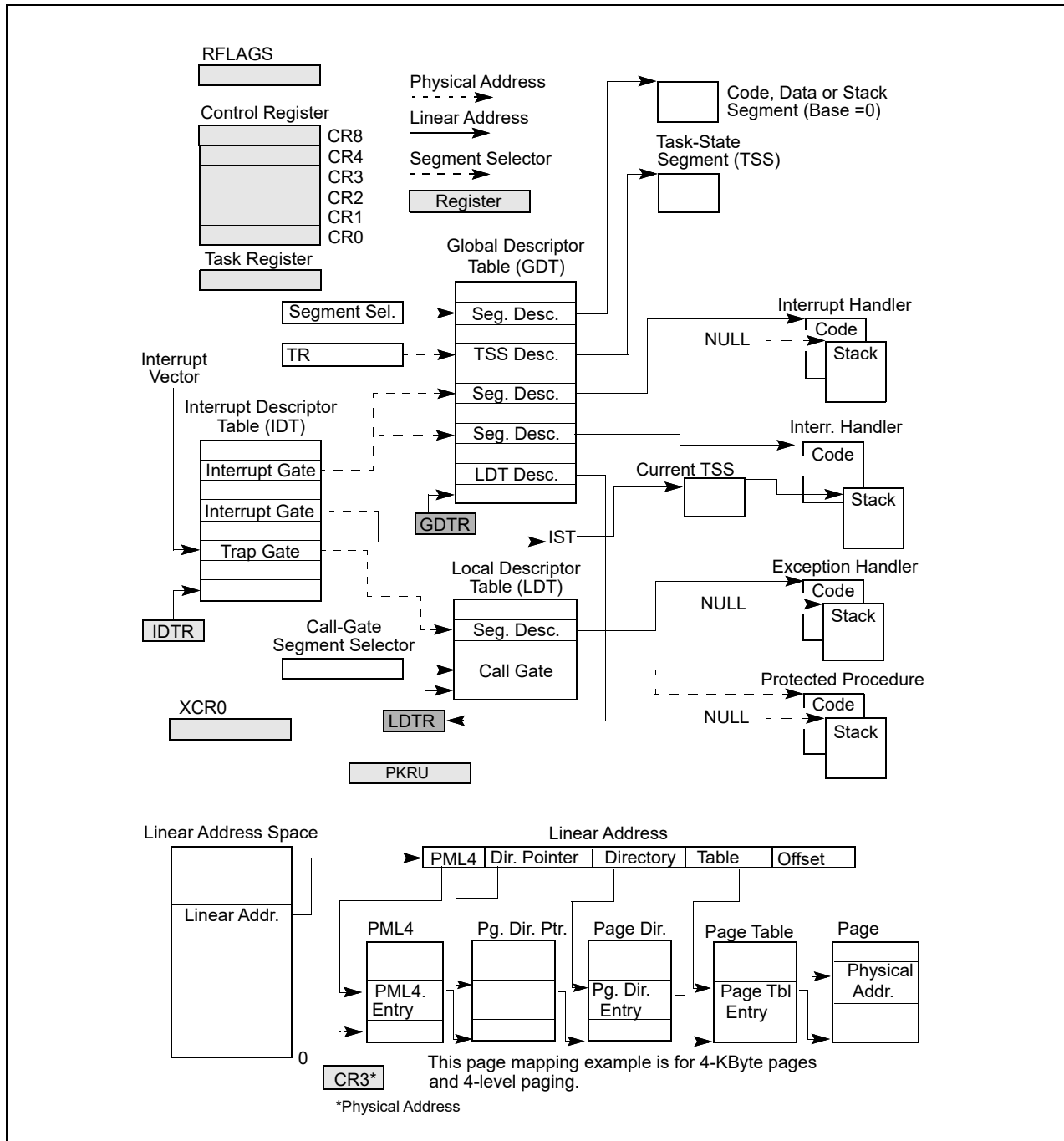


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode and 4-Level Paging

2.1.1 Global and Local Descriptor Tables

When operating in protected mode, all memory accesses pass through either the global descriptor table (GDT) or an optional local descriptor table (LDT) as shown in Figure 2-1. These tables contain entries called segment descriptors. Segment descriptors provide the base address of segments well as access rights, type, and usage information.

Each segment descriptor has an associated segment selector. A segment selector provides the software that uses it with an index into the GDT or LDT (the offset of its associated segment descriptor), a global/local flag (determines whether the selector points to the GDT or the LDT), and access rights information.

To access a byte in a segment, a segment selector and an offset must be supplied. The segment selector provides access to the segment descriptor for the segment (in the GDT or LDT). From the segment descriptor, the processor obtains the base address of the segment in the linear address space. The offset then provides the location of the byte relative to the base address. This mechanism can be used to access any valid code, data, or stack segment, provided the segment is accessible from the current privilege level (CPL) at which the processor is operating. The CPL is defined as the protection level of the currently executing code segment.

See Figure 2-1. The solid arrows in the figure indicate a linear address, dashed lines indicate a segment selector, and the dotted arrows indicate a physical address. For simplicity, many of the segment selectors are shown as direct pointers to a segment. However, the actual path from a segment selector to its associated segment is always through a GDT or LDT.

The linear address of the base of the GDT is contained in the GDT register (GDTR); the linear address of the LDT is contained in the LDT register (LDTR).

2.1.1.1 Global and Local Descriptor Tables in IA-32e Mode

GDTR and LDTR registers are expanded to 64-bits wide in both IA-32e sub-modes (64-bit mode and compatibility mode). For more information: see Section 3.5.2, "Segment Descriptor Tables in IA-32e Mode."

Global and local descriptor tables are expanded in 64-bit mode to support 64-bit base addresses, (16-byte LDT descriptors hold a 64-bit base address and various attributes). In compatibility mode, descriptors are not expanded.

2.1.2 System Segments, Segment Descriptors, and Gates

Besides code, data, and stack segments that make up the execution environment of a program or procedure, the architecture defines two system segments: the task-state segment (TSS) and the LDT. The GDT is not considered a segment because it is not accessed by means of a segment selector and segment descriptor. TSSs and LDTs have segment descriptors defined for them.

The architecture also defines a set of special descriptors called gates (call gates, interrupt gates, trap gates, and task gates). These provide protected gateways to system procedures and handlers that may operate at a different privilege level than application programs and most procedures. For example, a CALL to a call gate can provide access to a procedure in a code segment that is at the same or a numerically lower privilege level (more privileged) than the current code segment. To access a procedure through a call gate, the calling procedure¹ supplies the selector for the call gate. The processor then performs an access rights check on the call gate, comparing the CPL with the privilege level of the call gate and the destination code segment pointed to by the call gate.

If access to the destination code segment is allowed, the processor gets the segment selector for the destination code segment and an offset into that code segment from the call gate. If the call requires a change in privilege level, the processor also switches to the stack for the targeted privilege level. The segment selector for the new stack is obtained from the TSS for the currently running task. Gates also facilitate transitions between 16-bit and 32-bit code segments, and vice versa.

2.1.2.1 Gates in IA-32e Mode

In IA-32e mode, the following descriptors are 16-byte descriptors (expanded to allow a 64-bit base): LDT descriptors, 64-bit TSSs, call gates, interrupt gates, and trap gates.

Call gates facilitate transitions between 64-bit mode and compatibility mode. Task gates are not supported in IA-32e mode. On privilege level changes, stack segment selectors are not read from the TSS. Instead, they are set to NULL.

1. The word "procedure" is commonly used in this document as a general term for a logical unit or block of code (such as a program, procedure, function, or routine).

2.1.3 Task-State Segments and Task Gates

The TSS (see Figure 2-1) defines the state of the execution environment for a task. It includes the state of general-purpose registers, segment registers, the EFLAGS register, the EIP register, and segment selectors with stack pointers for three stack segments (one stack for each privilege level). The TSS also includes the segment selector for the LDT associated with the task and the base address of the paging-structure hierarchy.

All program execution in protected mode happens within the context of a task (called the current task). The segment selector for the TSS for the current task is stored in the task register. The simplest method for switching to a task is to make a call or jump to the new task. Here, the segment selector for the TSS of the new task is given in the CALL or JMP instruction. In switching tasks, the processor performs the following actions:

1. Stores the state of the current task in the current TSS.
2. Loads the task register with the segment selector for the new task.
3. Accesses the new TSS through a segment descriptor in the GDT.
4. Loads the state of the new task from the new TSS into the general-purpose registers, the segment registers, the LDTR, control register CR3 (base address of the paging-structure hierarchy), the EFLAGS register, and the EIP register.
5. Begins execution of the new task.

A task can also be accessed through a task gate. A task gate is similar to a call gate, except that it provides access (through a segment selector) to a TSS rather than a code segment.

2.1.3.1 Task-State Segments in IA-32e Mode

Hardware task switches are not supported in IA-32e mode. However, TSSs continue to exist. The base address of a TSS is specified by its descriptor.

A 64-bit TSS holds the following information that is important to 64-bit operation:

- Stack pointer addresses for each privilege level.
- Pointer addresses for the interrupt stack table.
- Offset address of the IO-permission bitmap (from the TSS base).

The task register is expanded to hold 64-bit base addresses in IA-32e mode. See also: Section 9.7, “Task Management in 64-bit Mode.”

2.1.4 Interrupt and Exception Handling

External interrupts, software interrupts and exceptions are handled through the interrupt descriptor table (IDT). The IDT stores a collection of gate descriptors that provide access to interrupt and exception handlers. Like the GDT, the IDT is not a segment. The linear address for the base of the IDT is contained in the IDT register (IDTR).

Gate descriptors in the IDT can be interrupt, trap, or task gate descriptors. To access an interrupt or exception handler, the processor first receives an interrupt vector from internal hardware, an external interrupt controller, or from software by means of an INT *n*, INTO, INT3, INT1, or BOUND instruction. The interrupt vector provides an index into the IDT. If the selected gate descriptor is an interrupt gate or a trap gate, the associated handler procedure is accessed in a manner similar to calling a procedure through a call gate. If the descriptor is a task gate, the handler is accessed through a task switch.

2.1.4.1 Interrupt and Exception Handling IA-32e Mode

In IA-32e mode, interrupt gate descriptors are expanded to 16 bytes to support 64-bit base addresses. This is true for 64-bit mode and compatibility mode.

The IDTR register is expanded to hold a 64-bit base address. Task gates are not supported.

2.1.5 Memory Management

System architecture supports either direct physical addressing of memory or virtual memory (through paging). When physical addressing is used, a linear address is treated as a physical address. When paging is used: all code, data, stack, and system segments (including the GDT and IDT) can be paged with only the most recently accessed pages being held in physical memory.

The location of pages (sometimes called page frames) in physical memory is contained in the paging structures. These structures reside in physical memory (see Figure 2-1 for the case of 32-bit paging).

The base physical address of the paging-structure hierarchy is contained in control register CR3. The entries in the paging structures determine the physical address of the base of a page frame, access rights and memory management information.

To use this paging mechanism, a linear address is broken into parts. The parts provide separate offsets into the paging structures and the page frame. A system can have a single hierarchy of paging structures or several. For example, each task can have its own hierarchy.

2.1.5.1 Memory Management in IA-32e Mode

In IA-32e mode, physical memory pages are managed by a set of system data structures. In both compatibility mode and 64-bit mode, four or five levels of system data structures are used (see Chapter 5, “Paging”). These include the following:

- **The page map level 5 (PML5)** — An entry in the PML5 table contains the physical address of the base of a PML4 table, access rights, and memory management information. The base physical address of the PML5 table is stored in CR3. The PML5 table is used only with 5-level paging.
- **A page map level 4 (PML4)** — An entry in a PML4 table contains the physical address of the base of a page directory pointer table, access rights, and memory management information. With 4-level paging, there is only one PML4 table and its base physical address is stored in CR3.
- **A set of page directory pointer tables** — An entry in a page directory pointer table contains the physical address of the base of a page directory table, access rights, and memory management information.
- **Sets of page directories** — An entry in a page directory table contains the physical address of the base of a page table, access rights, and memory management information.
- **Sets of page tables** — An entry in a page table contains the physical address of a page frame, access rights, and memory management information.

2.1.6 System Registers

To assist in initializing the processor and controlling system operations, the system architecture provides system flags in the EFLAGS register and several system registers:

- The system flags and IOPL field in the EFLAGS register control task and mode switching, interrupt handling, instruction tracing, and access rights. See also: Section 2.3, “System Flags and Fields in the EFLAGS Register.”
- The control registers (CR0, CR2, CR3, and CR4) contain a variety of flags and data fields for controlling system-level operations. Other flags in these registers are used to indicate support for specific processor capabilities within the operating system or executive. See also: Chapter 2, “Control Registers,” and Section 2.6, “Extended Control Registers (Including XCR0).”
- The debug registers (not shown in Figure 2-1) allow the setting of breakpoints for use in debugging programs and systems software. See also: Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”
- The GDTR, LDTR, and IDTR registers contain the linear addresses and sizes (limits) of their respective tables. See also: Section 2.4, “Memory-Management Registers.”
- The task register contains the linear address and size of the TSS for the current task. See also: Section 2.4, “Memory-Management Registers.”
- Model-specific registers (not shown in Figure 2-1).

The model-specific registers (MSRs) are a group of registers available primarily to operating-system or executive procedures (that is, code running at privilege level 0). These registers control items such as the debug extensions, the performance-monitoring counters, the machine-check architecture, and the memory type ranges (MTRRs).

The number and function of these registers varies among different members of the Intel 64 and IA-32 processor families. See also: Section 11.4, “Model-Specific Registers (MSRs),” and Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

Most systems restrict access to system registers (other than the EFLAGS register) by application programs. Systems can be designed, however, where all programs and procedures run at the most privileged level (privilege level 0). In such a case, application programs would be allowed to modify the system registers.

2.1.6.1 System Registers in IA-32e Mode

In IA-32e mode, the four system-descriptor-table registers (GDTR, IDTR, LDTR, and TR) are expanded in hardware to hold 64-bit base addresses. EFLAGS becomes the 64-bit RFLAGS register. CR0–CR4 are expanded to 64 bits. CR8 becomes available. CR8 provides read-write access to the task priority register (TPR) so that the operating system can control the priority classes of external interrupts.

In 64-bit mode, debug registers DR0–DR7 are 64 bits. In compatibility mode, address-matching in DR0–DR3 is also done at 64-bit granularity.

On systems that support IA-32e mode, the extended feature enable register (IA32_EFER) is available. This model-specific register controls activation of IA-32e mode and other IA-32e mode operations. In addition, there are several model-specific registers that govern IA-32e mode instructions:

- **IA32_KERNEL_GS_BASE** — Used by SWAPGS instruction.
- **IA32_LSTAR** — Used by SYSCALL instruction.
- **IA32_FMASK** — Used by SYSCALL instruction.
- **IA32_STAR** — Used by SYSCALL and SYSRET instruction.

2.1.7 Other System Resources

Besides the system registers and data structures described in the previous sections, system architecture provides the following additional resources:

- Operating system instructions (see also: Section 2.8, “System Instruction Summary”).
- Performance-monitoring counters (not shown in Figure 2-1).
- Internal caches and buffers (not shown in Figure 2-1).

Performance-monitoring counters are event counters that can be programmed to count processor events such as the number of instructions decoded, the number of interrupts received, or the number of cache loads.

The processor provides several internal caches and buffers. The caches are used to store both data and instructions. The buffers are used to store things like decoded addresses to system and application segments and write operations waiting to be performed. See also: Chapter 13, “Memory Cache Control.”

2.2 MODES OF OPERATION

The IA-32 architecture supports three operating modes and one quasi-operating mode:

- **Protected mode** — This is the native operating mode of the processor. It provides a rich set of architectural features, flexibility, high performance and backward compatibility to existing software base.
- **Real-address mode** — This operating mode provides the programming environment of the Intel 8086 processor, with a few extensions (such as the ability to switch to protected or system management mode).
- **System management mode (SMM)** — SMM is a standard architectural feature in all IA-32 processors, beginning with the Intel386 SL processor. This mode provides an operating system or executive with a transparent mechanism for implementing power management and OEM differentiation features. SMM is entered through activation of an external system interrupt pin (SMI#), which generates a system management

interrupt (SMI). In SMM, the processor switches to a separate address space while saving the context of the currently running program or task. SMM-specific code may then be executed transparently. Upon returning from SMM, the processor is placed back into its state prior to the SMI.

- **Virtual-8086 mode** — In protected mode, the processor supports a quasi-operating mode known as virtual-8086 mode. This mode allows the processor execute 8086 software in a protected, multitasking environment.

Intel 64 architecture supports all operating modes of IA-32 architecture and IA-32e modes:

- **IA-32e mode** — In IA-32e mode, the processor supports two sub-modes: compatibility mode and 64-bit mode. 64-bit mode provides 64-bit linear addressing and support for physical address space larger than 64 GBytes. Compatibility mode allows most legacy protected-mode applications to run unchanged.

Figure 2-3 shows how the processor moves between operating modes.

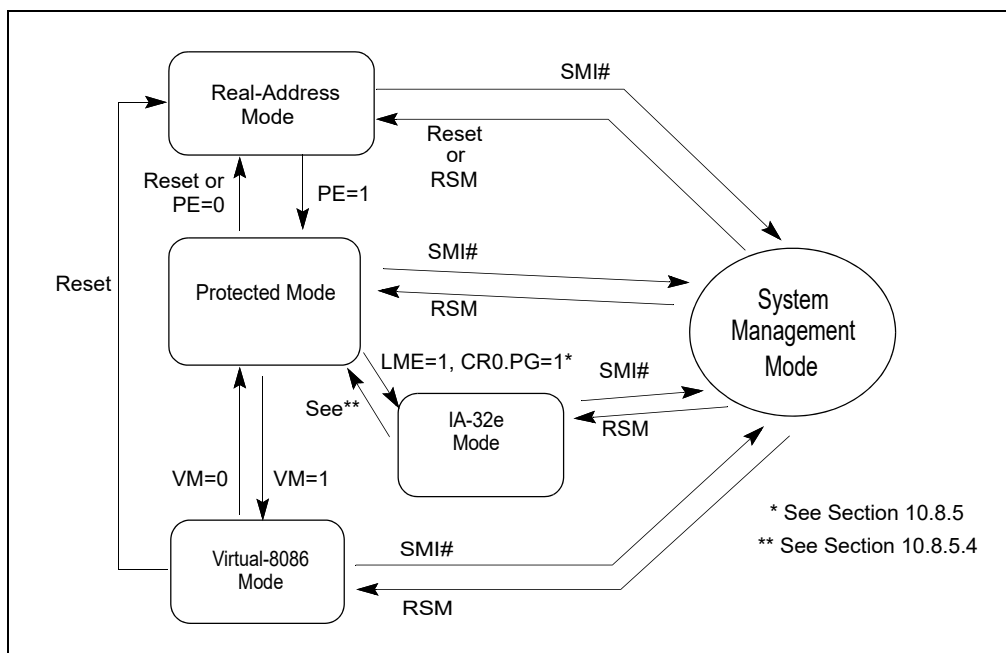


Figure 2-3. Transitions Among the Processor's Operating Modes

The processor is placed in real-address mode following power-up or a reset. The PE flag in control register CR0 then controls whether the processor is operating in real-address or protected mode. See also: Section 11.9, "Mode Switching," and Section 5.1.2, "Paging-Mode Enabling."

The VM flag in the EFLAGS register determines whether the processor is operating in protected mode or virtual-8086 mode. Transitions between protected mode and virtual-8086 mode are generally carried out as part of a task switch or a return from an interrupt or exception handler. See also: Section 22.2.5, "Entering Virtual-8086 Mode."

The LMA bit (IA32_EFER.LMA[bit 10]) determines whether the processor is operating in IA-32e mode. When running in IA-32e mode, 64-bit or compatibility sub-mode operation is determined by CS.L bit of the code segment. The processor enters into IA-32e mode from protected mode by enabling paging and setting the LME bit (IA32_EFER.LME[bit 8]). See also: Chapter 11, "Processor Management and Initialization."

The processor switches to SMM whenever it receives an SMI while the processor is in real-address, protected, virtual-8086, or IA-32e modes. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

2.2.1 Extended Feature Enable Register

The IA32_EFER MSR provides several fields related to IA-32e mode enabling and operation. It also provides one field that relates to page-access right modification (see Section 5.6, "Access Rights"). The layout of the IA32_EFER MSR is shown in Figure 2-4.

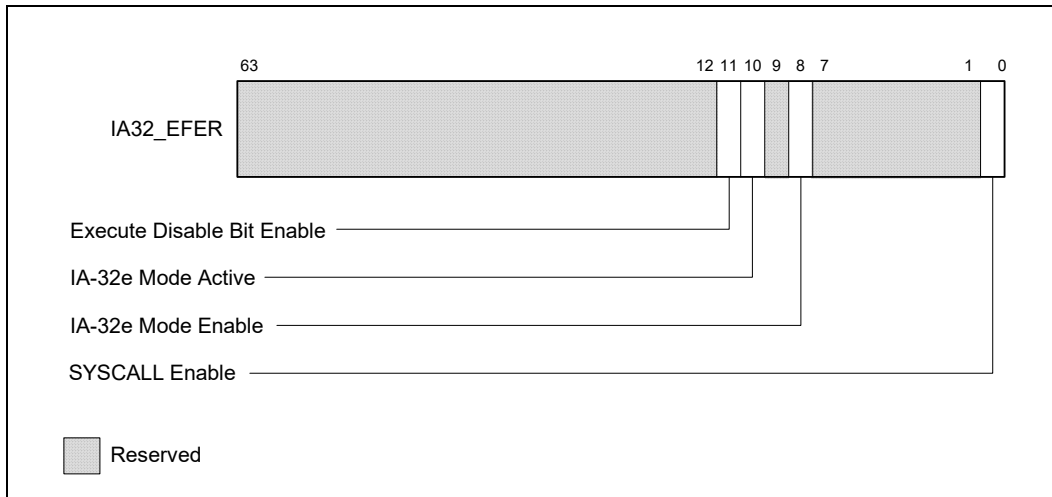


Figure 2-4. IA32_EFER MSR Layout

Table 2-1. IA32_EFER MSR Information

Bit	Description
0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.
7:1	Reserved.
8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.
9	Reserved.
10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.
11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W) Enables page access restriction by preventing instruction fetches from PAE pages with the XD bit set (See Section 5.6).
63:12	Reserved.

2.3 SYSTEM FLAGS AND FIELDS IN THE EFLAGS REGISTER

The system flags and IOPL field of the EFLAGS register control I/O, maskable hardware interrupts, debugging, task switching, and the virtual-8086 mode (see Figure 2-5). Only privileged code (typically operating system or executive code) should be allowed to modify these bits.

The system flags and IOPL are:

- TF **Trap (bit 8)** — Set to enable single-step mode for debugging; clear to disable single-step mode. In single-step mode, the processor generates a debug exception after each instruction. This allows the execution state of a program to be inspected after each instruction. If an application program sets the TF flag using a POPF, POPFD, or IRET instruction, a debug exception is generated after the instruction that follows the POPF, POPFD, or IRET.

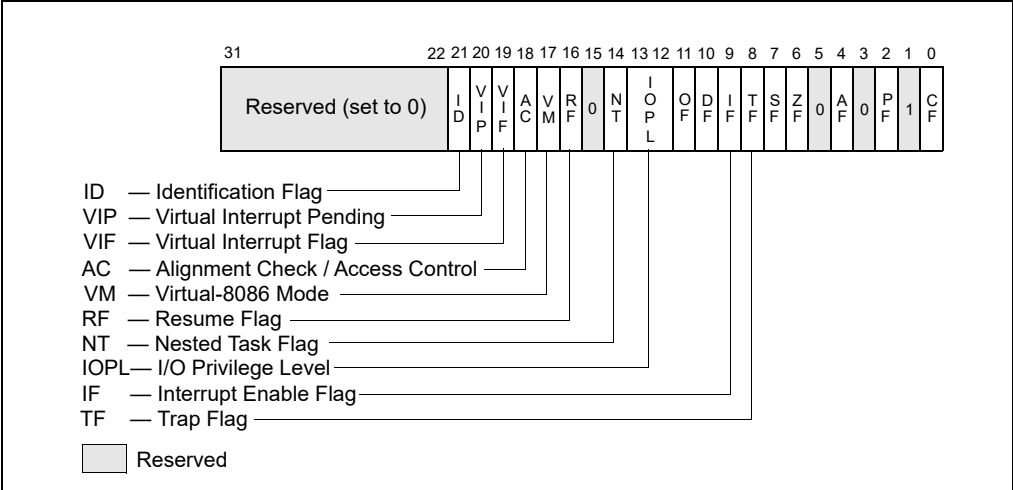


Figure 2-5. System Flags in the EFLAGS Register

- IF

Interrupt enable (bit 9) — Controls the response of the processor to maskable hardware interrupt requests (see also: Section 7.3.2, “Maskable Hardware Interrupts”). The flag is set to respond to maskable hardware interrupts; cleared to inhibit maskable hardware interrupts. The IF flag does not affect the generation of exceptions or nonmaskable interrupts (NMI interrupts). The CPL, IOPL, and the state of the VME flag in control register CR4 determine whether the IF flag can be modified by the CLI, STI, POPF, POPFD, and IRET.
- IOPL

I/O privilege level field (bits 12 and 13) — Indicates the I/O privilege level (IOPL) of the currently running program or task. The CPL of the currently running program or task must be less than or equal to the IOPL to access the I/O address space. The POPF and IRET instructions can modify this field only when operating at a CPL of 0.

The IOPL is also one of the mechanisms that controls the modification of the IF flag and the handling of interrupts in virtual-8086 mode when virtual mode extensions are in effect (when CR4.VME = 1). See also: Chapter 20, “Input/Output,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.
- NT

Nested task (bit 14) — Controls the chaining of interrupted and called tasks. The processor sets this flag on calls to a task initiated with a CALL instruction, an interrupt, or an exception. It examines and modifies this flag on returns from a task initiated with the IRET instruction. The flag can be explicitly set or cleared with the POPF/POPFD instructions; however, changing to the state of this flag can generate unexpected exceptions in application programs.

See also: Section 9.4, “Task Linking.”
- RF

Resume (bit 16) — Controls the processor’s response to instruction-breakpoint conditions. When set, this flag temporarily disables debug exceptions (#DB) from being generated for instruction breakpoints (although other exception conditions can cause an exception to be generated). When clear, instruction breakpoints will generate debug exceptions.

The primary function of the RF flag is to allow the restarting of an instruction following a debug exception that was caused by an instruction breakpoint condition. Here, debug software must set this flag in the EFLAGS image on the stack just prior to returning to the interrupted program with IRETD (to prevent the instruction breakpoint from causing another debug exception). The processor then automatically clears this flag after the instruction returned to has been successfully executed, enabling instruction breakpoint faults again.

See also: Section 19.3.1.1, “Instruction-Breakpoint Exception Condition.”
- VM

Virtual-8086 mode (bit 17) — Set to enable virtual-8086 mode; clear to return to protected mode.

See also: Section 22.2.1, “Enabling Virtual-8086 Mode.”

- AC Alignment check or access control (bit 18)** — If the AM bit is set in the CR0 register, alignment checking of user-mode data accesses is enabled if and only if this flag is 1. An alignment-check exception is generated when reference is made to an unaligned operand, such as a word at an odd byte address or a doubleword at an address which is not an integral multiple of four. Alignment-check exceptions are generated only in user mode (privilege level 3). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate this exception even when caused by instructions executed in user-mode.
- The alignment-check exception can be used to check alignment of data. This is useful when exchanging data with processors which require all data to be aligned. The alignment-check exception can also be used by interpreters to flag some pointers as special by misaligning the pointer. This eliminates overhead of checking each pointer and only handles the special pointer when used.
- If the SMAP bit is set in the CR4 register, explicit supervisor-mode data accesses to user-mode pages are allowed if and only if this bit is 1. See Section 5.6, “Access Rights.”
- VIF Virtual Interrupt (bit 19)** — Contains a virtual image of the IF flag. This flag is used in conjunction with the VIP flag. The processor only recognizes the VIF flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. (The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.)
- See also: Section 22.3.3.5, “Method 6: Software Interrupt Handling,” and Section 22.4, “Protected-Mode Virtual Interrupts.”
- VIP Virtual interrupt pending (bit 20)** — Set by software to indicate that an interrupt is pending; cleared to indicate that no interrupt is pending. This flag is used in conjunction with the VIF flag. The processor reads this flag but never modifies it. The processor only recognizes the VIP flag when either the VME flag or the PVI flag in control register CR4 is set and the IOPL is less than 3. The VME flag enables the virtual-8086 mode extensions; the PVI flag enables the protected-mode virtual interrupts.
- See Section 22.3.3.5, “Method 6: Software Interrupt Handling,” and Section 22.4, “Protected-Mode Virtual Interrupts.”
- ID Identification (bit 21)** — The ability of a program or procedure to set or clear this flag indicates support for the CPUID instruction.

2.3.1 System Flags and Fields in IA-32e Mode

In 64-bit mode, the RFLAGS register expands to 64 bits with the upper 32 bits reserved. System flags in RFLAGS (64-bit mode) or EFLAGS (compatibility mode) are shown in Figure 2-5.

In IA-32e mode, the processor does not allow the VM bit to be set because virtual-8086 mode is not supported (attempts to set the bit are ignored). Also, the processor will not set the NT bit. The processor does, however, allow software to set the NT bit (note that an IRET causes a general protection fault in IA-32e mode if the NT bit is set).

In IA-32e mode, the SYSCALL/SYSRET instructions have a programmable method of specifying which bits are cleared in RFLAGS/EFLAGS. These instructions save/restore EFLAGS/RFLAGS.

2.4 MEMORY-MANAGEMENT REGISTERS

The processor provides four memory-management registers (GDTR, LDTR, IDTR, and TR) that specify the locations of the data structures which control segmented memory management (see Figure 2-6). Special instructions are provided for loading and storing these registers.

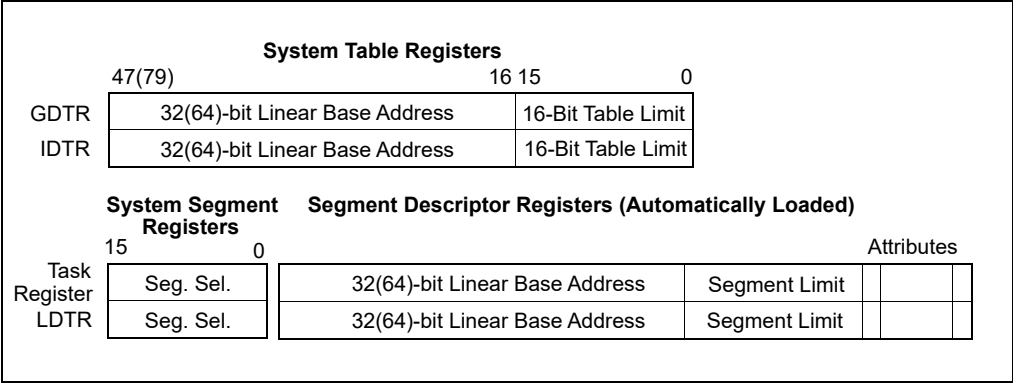


Figure 2-6. Memory Management Registers

2.4.1 Global Descriptor Table Register (GDTR)

The GDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and the 16-bit table limit for the GDT. The base address specifies the linear address of byte 0 of the GDT; the table limit specifies the number of bytes in the table.

The LGDT and SGDT instructions load and store the GDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. A new base address must be loaded into the GDTR as part of the processor initialization process for protected-mode operation.

See also: Section 3.5.1, "Segment Descriptor Tables."

2.4.2 Local Descriptor Table Register (LDTR)

The LDTR register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the LDT. The base address specifies the linear address of byte 0 of the LDT segment; the segment limit specifies the number of bytes in the segment. See also: Section 3.5.1, "Segment Descriptor Tables."

The LLDT and SLDT instructions load and store the segment selector part of the LDTR register, respectively. The segment that contains the LDT must have a segment descriptor in the GDT. When the LLDT instruction loads a segment selector in the LDTR: the base address, limit, and descriptor attributes from the LDT descriptor are automatically loaded in the LDTR.

When a task switch occurs, the LDTR is automatically loaded with the segment selector and descriptor for the LDT for the new task. The contents of the LDTR are not automatically saved prior to writing the new LDT information into the register.

On power up or reset of the processor, the segment selector and base address are set to the default value of 0 and the limit is set to 0FFFFH.

2.4.3 IDTR Interrupt Descriptor Table Register

The IDTR register holds the base address (32 bits in protected mode; 64 bits in IA-32e mode) and 16-bit table limit for the IDT. The base address specifies the linear address of byte 0 of the IDT; the table limit specifies the number of bytes in the table. The LIDT and SIDT instructions load and store the IDTR register, respectively. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH. The base address and limit in the register can then be changed as part of the processor initialization process.

See also: Section 7.10, "Interrupt Descriptor Table (IDT)."

2.4.4 Task Register (TR)

The task register holds the 16-bit segment selector, base address (32 bits in protected mode; 64 bits in IA-32e mode), segment limit, and descriptor attributes for the TSS of the current task. The selector references the TSS descriptor in the GDT. The base address specifies the linear address of byte 0 of the TSS; the segment limit specifies the number of bytes in the TSS. See also: Section 9.2.4, “Task Register.”

The LTR and STR instructions load and store the segment selector part of the task register, respectively. When the LTR instruction loads a segment selector in the task register, the base address, limit, and descriptor attributes from the TSS descriptor are automatically loaded into the task register. On power up or reset of the processor, the base address is set to the default value of 0 and the limit is set to 0FFFFH.

When a task switch occurs, the task register is automatically loaded with the segment selector and descriptor for the TSS for the new task. The contents of the task register are not automatically saved prior to writing the new TSS information into the register.

2.5 CONTROL REGISTERS

Control registers (CR0, CR1, CR2, CR3, and CR4; see Figure 2-7) determine operating mode of the processor and the characteristics of the currently executing task. These registers are 32 bits in all 32-bit modes and compatibility mode.

In 64-bit mode, control registers are expanded to 64 bits. The MOV CRn instructions are used to manipulate the register bits. Operand-size prefixes for these instructions are ignored. The following is also true:

- The control registers can be read and loaded (or modified) using the move-to-or-from-control-registers forms of the MOV instruction. In protected mode, the MOV instructions allow the control registers to be read or loaded (at privilege level 0 only). This restriction means that application programs or operating-system procedures (running at privilege levels 1, 2, or 3) are prevented from reading or loading the control registers.
- Some of the bits in CR0 and CR4 are reserved and must be written with zeros. Attempting to set any reserved bits in CR0[31:0] is ignored. Attempting to set any reserved bits in CR0[63:32] results in a general-protection exception, #GP(0). Attempting to set any reserved bits in CR4 results in a general-protection exception, #GP(0).
- All 64 bits of CR2 are writable by software.
- Bits in CR3 in the range 63:MAXPHYADDR that are reserved (see Figure 2-7) must be zero. Attempting to set any of them results in #GP(0).¹
- The MOV CR2 instruction does not check that address written to CR2 is canonical.
- A 64-bit capable processor will retain the upper 32 bits of each control register when transitioning out of IA-32e mode.
- On a 64-bit capable processor, an execution of MOV to CR outside of 64-bit mode zeros the upper 32 bits of the control register.
- Register CR8 is available in 64-bit mode only.

The control registers are summarized below, and each architecturally defined control field in these control registers is described individually. In Figure 2-7, the width of the register in 64-bit mode is indicated in parenthesis (except for CR0).

- **CR0** — Contains system control flags that control operating mode and states of the processor.
- **CR1** — Reserved.
- **CR2** — Contains the page-fault linear address (the linear address that caused a page fault).
- **CR3** — Contains the physical address of the base of the paging-structure hierarchy and four flags (PWT, PCD, LAM_U57, and LAM_U48). Only the most-significant bits (less the lower 12 bits) of the base address are specified; the lower 12 bits of the address are assumed to be 0. The first paging structure must thus be aligned

1. MAXPHYADDR is derived from the value enumerated by CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is out-side secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.

to a page (4-KByte) boundary. The PCD and PWT flags control caching of that paging structure in the processor's internal data caches (they do not control TLB caching of page-directory information). The LAM_U57 and LAM_U48 flags control linear-address masking for user addresses (see Section 4.4, "Linear-Address Masking").

When using the physical address extension, the CR3 register contains the base address of the page-directory-pointer table. With 4-level paging and 5-level paging, the CR3 register contains the base address of the PML4 table and PML5 table, respectively. If PCIDs are enabled, CR3 has a format different from that illustrated in Figure 2-7. See Section 5.5, "4-Level Paging and 5-Level Paging."

See also: Chapter 5, "Paging."

- **CR4** — Contains a group of flags that enable several architectural extensions, and indicate operating system or executive support for specific processor capabilities. Bits CR4[63:32] can only be used for IA-32e mode only features that are enabled after entering 64-bit mode. Bits CR4[31:0] do not have any effect outside of IA-32e mode.
- **CR8** — Provides (in CR8[3:0]) read and write access to bits 7:4 of the local APIC's Task-Priority Register (TPR). These bits are the **task-priority class**; all interrupts in this and lower priority classes are blocked. A value of 15 blocks interrupts in all priority classes, while a value of 0 blocks none. CR8 can be accessed only in 64-bit mode, but the value of the TPR blocks interrupts regardless of mode.

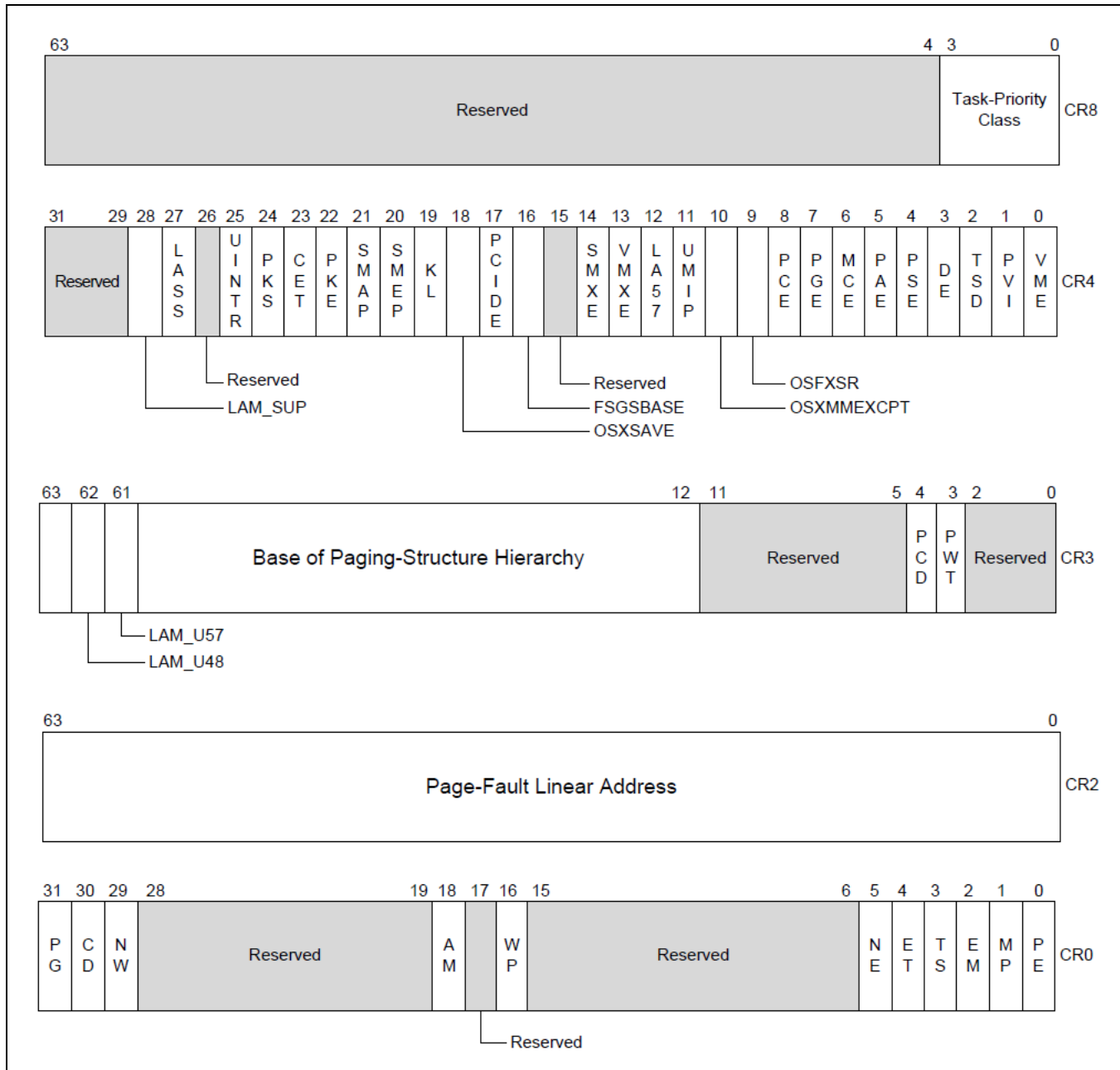


Figure 2-7. Control Registers

The flags in control registers are:

CR0.PG

Paging (bit 31 of CR0) — Enables paging when set; disables paging when clear. When paging is disabled, all linear addresses are treated as physical addresses. The PG flag has no effect if the PE flag (bit 0 of register CR0) is not also set; setting the PG flag when the PE flag is clear causes a general-protection exception (#GP). See also: Chapter 5, “Paging.”

On Intel 64 processors, enabling and disabling IA-32e mode operation also requires modifying CR0.PG.

CR0.CD

Cache Disable (bit 30 of CR0) — When the CD and NW flags are clear, caching of memory locations for the whole of physical memory in the processor's internal (and external) caches is enabled. When the CD flag is set, caching is restricted as described in Table 13-5. To prevent the processor from accessing and updating its caches, the CD flag must be set and the caches must be invalidated so that no cache hits can occur.

See also: Section 13.5.3, "Preventing Caching," and Section 13.5, "Cache Control."

CR0.NW

Not Write-through (bit 29 of CR0) — When the NW and CD flags are clear, write-back (for Pentium 4, Intel Xeon, P6 family, and Pentium processors) or write-through (for Intel486 processors) is enabled for writes that hit the cache and invalidation cycles are enabled. See Table 13-5 for detailed information about the effect of the NW flag on caching for other settings of the CD and NW flags.

CR0.AM

Alignment Mask (bit 18 of CR0) — Enables automatic alignment checking when set; disables alignment checking when clear. Alignment checking is performed only when the AM flag is set, the AC flag in the EFLAGS register is set, CPL is 3, and the processor is operating in either protected or virtual-8086 mode.

CR0.WP

Write Protect (bit 16 of CR0) — When set, inhibits supervisor-level procedures from writing into read-only pages; when clear, allows supervisor-level procedures to write into read-only pages (regardless of the U/S bit setting; see Section 5.1.3 and Section 5.6). This flag facilitates implementation of the copy-on-write method of creating a new process (forking) used by operating systems such as UNIX. This flag must be set before software can set CR4.CET, and it cannot be cleared as long as CR4.CET = 1 (see below).

CR0.NE

Numeric Error (bit 5 of CR0) — Enables the native (internal) mechanism for reporting x87 FPU errors when set; enables the PC-style x87 FPU error reporting mechanism when clear. When the NE flag is clear and the IGNNE# input is asserted, x87 FPU errors are ignored. When the NE flag is clear and the IGNNE# input is deasserted, an unmasked x87 FPU error causes the processor to assert the FERR# pin to generate an external interrupt and to stop instruction execution immediately before executing the next waiting floating-point instruction or WAIT/FWAIT instruction.

The FERR# pin is intended to drive an input to an external interrupt controller (the FERR# pin emulates the ERROR# pin of the Intel 287 and Intel 387 DX math coprocessors). The NE flag, IGNNE# pin, and FERR# pin are used with external logic to implement PC-style error reporting. Using FERR# and IGNNE# to handle floating-point exceptions is deprecated by modern operating systems; this non-native approach also limits newer processors to operate with one logical processor active.

See also: Section 8.7, "Handling x87 FPU Exceptions in Software," in Chapter 8, "Programming with the x87 FPU," and Appendix A, "EFLAGS Cross-Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

CR0.ET

Extension Type (bit 4 of CR0) — Reserved in the Pentium 4, Intel Xeon, P6 family, and Pentium processors. In the Pentium 4, Intel Xeon, and P6 family processors, this flag is hardcoded to 1. In the Intel386 and Intel486 processors, this flag indicates support of Intel 387 DX math coprocessor instructions when set.

CR0.TS

Task Switched (bit 3 of CR0) — Allows the saving of the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context on a task switch to be delayed until an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction is actually executed by the new task. The processor sets this flag on every task switch and tests it when executing x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

- If the TS flag is set and the EM flag (bit 2 of CR0) is clear, a device-not-available exception (#NM) is raised prior to the execution of any x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction; with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. See the paragraph below for the special case of the WAIT/FWAIT instructions.
- If the TS flag is set and the MP flag (bit 1 of CR0) and EM flag are clear, an #NM exception is not raised prior to the execution of an x87 FPU WAIT/FWAIT instruction.
- If the EM flag is set, the setting of the TS flag has no effect on the execution of x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

Table 2-2 shows the actions taken when the processor encounters an x87 FPU instruction based on the settings of the TS, EM, and MP flags. Table 14-1 and 15-1 show the actions taken when the processor encounters an MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction.

The processor does not automatically save the context of the x87 FPU, XMM, and MXCSR registers on a task switch. Instead, it sets the TS flag, which causes the processor to raise an #NM exception whenever it encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction in the instruction stream for the new task (with the exception of the instructions listed above).

The fault handler for the #NM exception can then be used to clear the TS flag (with the CLTS instruction) and save the context of the x87 FPU, XMM, and MXCSR registers. If the task never encounters an x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instruction, the x87 FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4 context is never saved.

Table 2-2. Action Taken By x87 FPU Instructions for Different Combinations of EM, MP, and TS

CR0 Flags			x87 FPU Instruction Type	
EM	MP	TS	Floating-Point	WAIT/FWAIT
0	0	0	Execute	Execute.
0	0	1	#NM Exception	Execute.
0	1	0	Execute	Execute.
0	1	1	#NM Exception	#NM exception.
1	0	0	#NM Exception	Execute.
1	0	1	#NM Exception	Execute.
1	1	0	#NM Exception	Execute.
1	1	1	#NM Exception	#NM exception.

CR0.EM

Emulation (bit 2 of CR0) — Indicates that the processor does not have an internal or external x87 FPU when set; indicates an x87 FPU is present when clear. This flag also affects the execution of MMX/SSE/SSE2/SSE3/SSSE3/SSE4 instructions.

When the EM flag is set, execution of an x87 FPU instruction generates a device-not-available exception (#NM). This flag must be set when the processor does not have an internal x87 FPU or is not connected to an external math coprocessor. Setting this flag forces all floating-point instructions to be handled by software emulation. Table 11-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the EM, MP, and TS flags.

Also, when the EM flag is set, execution of an MMX instruction causes an invalid-opcode exception (#UD) to be generated (see Table 14-1). Thus, if an IA-32 or Intel 64 processor incorporates MMX technology, the EM flag must be set to 0 to enable execution of MMX instructions.

Similarly for SSE/SSE2/SSE3/SSSE3/SSE4 extensions, when the EM flag is set, execution of most SSE/SSE2/SSE3/SSSE3/SSE4 instructions causes an invalid opcode exception (#UD) to be generated (see Table 15-1). If an IA-32 or Intel 64 processor incorporates the SSE/SSE2/SSE3/SSSE3/SSE4 extensions, the EM flag must be set to 0 to enable execution of these extensions. SSE/SSE2/SSE3/SSSE3/SSE4 instructions not affected by the EM flag include: PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

CR0.MP

Monitor Coprocessor (bit 1 of CR0) — Controls the interaction of the WAIT (or FWAIT) instruction with the TS flag (bit 3 of CR0). If the MP flag is set, a WAIT instruction generates a device-not-available exception (#NM) if the TS flag is also set. If the MP flag is clear, the WAIT instruction ignores the setting of the TS flag. Table 11-3 shows the recommended setting of this flag, depending on the IA-32 processor and x87 FPU or math coprocessor present in the system. Table 2-2 shows the interaction of the MP, EM, and TS flags.

CR0.PE

Protection Enable (bit 0 of CR0) — Enables protected mode when set; enables real-address mode when clear. This flag does not enable paging directly. It only enables segment-level protection. To enable paging, both the PE and PG flags must be set.

See also: Section 11.9, “Mode Switching.”

CR3.LAM_U48

User LAM48 enable (bit 62 of CR3) — When set and CR3.LAM_U57 is clear, enables LAM48 (masking of linear-address bits 62:48) for user pointers. See Section 4.4, “Linear-Address Masking.”

CR3.LAM_U57

User LAM57 enable (bit 61 of CR3) — When set, enables LAM57 (masking of linear-address bits 62:57) for user pointers and overrides CR3.LAM_U48. See Section 4.4, “Linear-Address Masking.”

CR3.PCD

Page-level Cache Disable (bit 4 of CR3) — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 5.9, “Paging and Memory Typing.” This bit is not used if paging is disabled, with PAE paging, or with 4-level paging¹ or 5-level paging if CR4.PCIDE=1.

CR3.PWT

Page-level Write-Through (bit 3 of CR3) — Controls the memory type used to access the first paging structure of the current paging-structure hierarchy. See Section 5.9, “Paging and Memory Typing.” This bit is not used if paging is disabled, with PAE paging, or with 4-level paging or 5-level paging if CR4.PCIDE=1.

CR4.VME

Virtual-8086 Mode Extensions (bit 0 of CR4) — Enables interrupt- and exception-handling extensions in virtual-8086 mode when set; disables the extensions when clear. Use of the virtual mode extensions can improve the performance of virtual-8086 applications by eliminating the overhead of calling the virtual-8086 monitor to handle interrupts and exceptions that occur while executing an 8086 program and, instead, redirecting the interrupts and exceptions back to the 8086 program’s handlers. It also provides hardware support for a virtual interrupt flag (VIF) to improve reliability of running 8086 programs in multi-tasking and multiple-processor environments.

See also: Section 22.3, “Interrupt and Exception Handling in Virtual-8086 Mode.”

CR4.LAM_SUP

Supervisor LAM enable (bit 28 of CR4) — When set, enables LAM (linear-address masking) for supervisor pointers. See Section 4.4, “Linear-Address Masking.”

CR4.LASS

Linear-address-space Separation (bit 27 of CR4) — When set, enables LASS (linear-address-space separation). See Section 4.3, “Linear-Address-Space Separation (LASS).”

CR4.UINTR

User Interrupts Enable Bit (bit 25 of CR4) — Enables user interrupts when set, including user-interrupt delivery, user-interrupt notification identification, and the user-interrupt instructions.

CR4.PKS

Enable protection keys for supervisor-mode pages (bit 24 of CR4) — 4-level paging and 5-level paging associate each supervisor-mode linear address with a protection key. When set, this flag allows use of the IA32_PKRS MSR to specify, for each protection key, whether supervisor-mode linear addresses with that protection key can be read or written.

CR4.CET

Control-flow Enforcement Technology (bit 23 of CR4) — Enables control-flow enforcement technology when set. See Chapter 18, “Control-flow Enforcement Technology (CET),” of the *IA-32 Intel® Architecture Software Developer’s Manual, Volume 1*. This flag can be set only if CR0.WP is set, and it must be clear before CR0.WP can be cleared (see below).

CR4.PKE

Enable protection keys for user-mode pages (bit 22 of CR4) — 4-level paging and 5-level paging associate each user-mode linear address with a protection key. When set, this flag indicates (via CPUID.07H.00H:ECX.OSPKE[4]) that the operating system supports use of the PKRU register to specify, for

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

each protection key, whether user-mode linear addresses with that protection key can be read or written. This bit also enables access to the PKRU register using the RDPKRU and WRPKRU instructions.

CR4.SMAP

SMAP-Enable Bit (bit 21 of CR4) — Enables supervisor-mode access prevention (SMAP) when set. See Section 5.6, “Access Rights.”

CR4.SMEP

SMEP-Enable Bit (bit 20 of CR4) — Enables supervisor-mode execution prevention (SMEP) when set. See Section 5.6, “Access Rights.”

CR4.KL

Key-Locker-Enable Bit (bit 19 of CR4) — When set, the `LOADIWKEY` instruction is enabled; in addition, if support for the AES Key Locker instructions has been activated by system firmware, `CPUID.19H:EBX.AESKLE[0]` is enumerated as 1 and the AES Key Locker instructions are enabled.¹ When clear, `CPUID.19H:EBX.AESKLE[0]` is enumerated as 0 and execution of any Key Locker instruction causes an invalid-opcode exception (`#UD`).

CR4.OSXSAVE

XSAVE and Processor Extended States-Enable Bit (bit 18 of CR4) — When set, this flag: (1) indicates (via `CPUID.01H:ECX.OSXSAVE[27]`) that the operating system supports the use of the `XGETBV`, `XSAVE`, and `XRSTOR` instructions by general software; (2) enables the `XSAVE` and `XRSTOR` instructions to save and restore the x87 FPU state (including MMX registers), the SSE state (XMM registers and MXCSR), along with other processor extended states enabled in `XCR0`; (3) enables the processor to execute `XGETBV` and `XSETBV` instructions in order to read and write `XCR0`. See Section 2.6 and Chapter 15, “System Programming for Instruction Set Extensions and Processor Extended States.”

CR4.PCIDE

PCID-Enable Bit (bit 17 of CR4) — Enables process-context identifiers (PCIDs) when set. See Section 5.10.1, “Process-Context Identifiers (PCIDs).” Applies only in IA-32e mode (if `IA32_EFER.LMA = 1`).

CR4.FSGSBASE

FSGSBASE-Enable Bit (bit 16 of CR4) — Enables the instructions `RDFSBASE`, `RDGSBASE`, `WRFSBASE`, and `WRGSBASE`.

CR4.SMXE

SMX-Enable Bit (bit 14 of CR4) — Enables SMX operation when set. See Chapter 7, “Safer Mode Extensions Reference,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

See also: Chapter 5, “Paging.”

CR4.VMXE

VMX-Enable Bit (bit 13 of CR4) — Enables VMX operation when set. See Chapter 25, “Introduction to Virtual Machine Extensions.”

CR4.LA57

57-bit linear addresses (bit 12 of CR4) — When set in IA-32e mode, the processor uses 5-level paging to translate 57-bit linear addresses. When clear in IA-32e mode, the processor uses 4-level paging to translate 48-bit linear addresses. This bit cannot be modified in IA-32e mode.

CR4.UIMP

User-Mode Instruction Prevention (bit 11 of CR4) — When set, the following instructions cannot be executed if `CPL > 0`: `SGDT`, `SIDT`, `SLDT`, `SMSW`, and `STR`. An attempt at such execution causes a general-protection exception (`#GP`).

CR4.OSXMMEXCPT

Operating System Support for Unmasked SIMD Floating-Point Exceptions (bit 10 of CR4) — When set, indicates that the operating system supports the handling of unmasked SIMD floating-point exceptions through an exception handler that is invoked when a SIMD floating-point exception (`#XM`) is

1. Software can check `CPUID.19H:EBX.AESKLE[0]` after setting `CR4.KL` to determine whether the AES Key Locker instructions have been enabled. Note that some processors may allow enabling of those instructions without activation by system firmware. Some processors may not support use of the AES Key Locker instructions in system-management mode (SMM). Those processors enumerate `CPUID.19H:EBX.AESKLE[0]` as 0 in SMM regardless of the setting of `CR4.KL`.

generated. SIMD floating-point exceptions are only generated by SSE/SSE2/SSE3/SSE4.1 SIMD floating-point instructions.

The operating system or executive must explicitly set this flag. If this flag is not set, the processor will generate an invalid opcode exception (#UD) whenever it detects an unmasked SIMD floating-point exception.

CR4.OSFXSR

Operating System Support for FXSAVE and FXRSTOR instructions (bit 9 of CR4) — When set, this flag: (1) indicates to software that the operating system supports the use of the FXSAVE and FXRSTOR instructions, (2) enables the FXSAVE and FXRSTOR instructions to save and restore the contents of the XMM and MXCSR registers along with the contents of the x87 FPU and MMX registers, and (3) enables the processor to execute SSE/SSE2/SSE3/SSSE3/SSE4 instructions, with the exception of the PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT.

If this flag is clear, the FXSAVE and FXRSTOR instructions will save and restore the contents of the x87 FPU and MMX registers, but they may not save and restore the contents of the XMM and MXCSR registers. Also, the processor will generate an invalid opcode exception (#UD) if it attempts to execute any SSE/SSE2/SSE3 instruction, with the exception of PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, MOVNTI, CLFLUSH, CRC32, and POPCNT. The operating system or executive must explicitly set this flag.

NOTE

CPUID feature flag FXSR indicates availability of the FXSAVE/FXRSTOR instructions. The OSFXSR bit provides operating system software with a means of enabling FXSAVE/FXRSTOR to save/restore the contents of the X87 FPU, XMM, and MXCSR registers. Consequently OSFXSR bit indicates that the operating system provides context switch support for SSE/SSE2/SSE3/SSSE3/SSE4.

CR4.PCE

Performance-Monitoring Counter Enable (bit 8 of CR4) — Enables execution of the RDPMC instruction for programs or procedures running at any protection level when set; RDPMC instruction can be executed only at protection level 0 when clear.

CR4.PGE

Page Global Enable (bit 7 of CR4) — (Introduced in the P6 family processors.) Enables the global page feature when set; disables the global page feature when clear. The global page feature allows frequently used or shared pages to be marked as global to all users (done with the global flag, bit 8, in a page-directory-pointer-table entry, a page-directory entry, or a page-table entry). Global pages are not flushed from the translation-lookaside buffer (TLB) on a task switch or a write to register CR3.

When enabling the global page feature, paging must be enabled (by setting the PG flag in control register CR0) before the PGE flag is set. Reversing this sequence may affect program correctness, and processor performance will be impacted.

See also: Section 5.10, "Caching Translation Information."

CR4.MCE

Machine-Check Enable (bit 6 of CR4) — Enables the machine-check exception when set; disables the machine-check exception when clear.

See also: Chapter 17, "Machine-Check Architecture."

CR4.PAE

Physical Address Extension (bit 5 of CR4) — When set, enables paging to produce physical addresses with more than 32 bits. When clear, restricts physical addresses to 32 bits. PAE must be set before entering IA-32e mode.

See also: Chapter 5, "Paging."

CR4.PSE

Page Size Extensions (bit 4 of CR4) — Enables 4-MByte pages with 32-bit paging when set; restricts 32-bit paging to pages of 4 KBytes when clear.

See also: Section 5.3, "32-Bit Paging."

CR4.DE

Debugging Extensions (bit 3 of CR4) — References to debug registers DR4 and DR5 cause an undefined opcode (#UD) exception to be generated when set; when clear, processor aliases references to registers DR4 and DR5 for compatibility with software written to run on earlier IA-32 processors.

See also: Section 19.2.2, “Debug Registers DR4 and DR5.”

CR4.TSDTime Stamp Disable (bit 2 of CR4) — Restricts the execution of the RDTSC instruction to procedures running at privilege level 0 when set; allows RDTSC instruction to be executed at any privilege level when clear. This bit also applies to the RDTSCP instruction if supported (if CPUID.80000001H:EDX[27] = 1).

CR4.PVI

Protected-Mode Virtual Interrupts (bit 1 of CR4) — Enables hardware support for a virtual interrupt flag (VIF) in protected mode when set; disables the VIF flag in protected mode when clear.

See also: Section 22.4, “Protected-Mode Virtual Interrupts.”

2.5.1 CPUID Qualification of Control Register Flags

Not all flags in control register CR4 are implemented on all processors. With the exception of the PCE flag, they can be qualified with the CPUID instruction to determine if they are implemented on the processor before they are used.

The CR8 register is available on processors that support Intel 64 architecture.

2.6 EXTENDED CONTROL REGISTERS (INCLUDING XCRO)

If CPUID.01H:ECX.XSAVE[26] is 1, the processor supports one or more **extended control registers** (XCRs). Currently, the only such register defined is XCR0. This register specifies the set of processor state components for which the operating system provides context management, e.g., x87 FPU state, SSE state, AVX state. The OS programs XCR0 to reflect the features for which it provides context management.

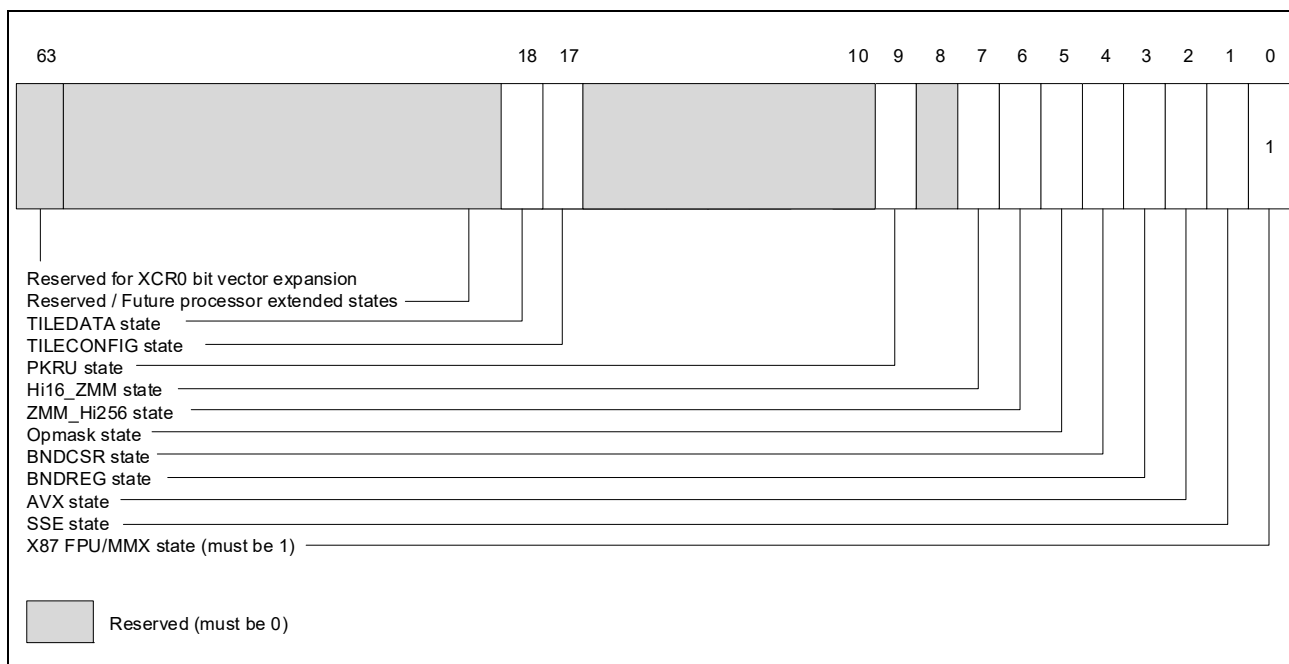


Figure 2-8. XCR0

Software can access XCR0 only if CR4.OSXSAVE[bit 18] = 1. (This bit is also readable as CPUID.01H:ECX.OSXSAVE[27].) Software can use CPUID.0DH to enumerate the bits in XCR0 that the processor supports (see CPUID instruction in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A).

Each supported state component is represented by a bit in XCR0. System software enables state components by loading an appropriate bit mask value into XCR0 using the XSETBV instruction.

As each bit in XCR0 (except bit 63) corresponds to a processor state component, XCR0 thus provides support for up to 63 sets of processor state components. Bit 63 of XCR0 is reserved for future expansion and will not represent a processor state component.

Currently, XCR0 defines support for the following state components:

- XCR0.X87 (bit 0): This bit 0 must be 1. An attempt to write 0 to this bit causes a #GP exception.
- XCR0.SSE (bit 1): If 1, the XSAVE feature set can be used to manage MXCSR and the XMM registers (XMM0-XMM15 in 64-bit mode; otherwise XMM0-XMM7).
- XCR0.AVX (bit 2): If 1, Intel AVX instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the YMM registers (YMM0-YMM15 in 64-bit mode; otherwise YMM0-YMM7).
- XCR0.BNDREG (bit 3): If 1, Intel MPX instructions can be executed and the XSAVE feature set can be used to manage the bounds registers BND0-BND3.
- XCR0.BNDCSR (bit 4): If 1, Intel MPX instructions can be executed and the XSAVE feature set can be used to manage the BNDCFGU and BNDSTATUS registers.
- XCR0.opmask (bit 5): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the opmask registers k0-k7.
- XCR0.ZMM_Hi256 (bit 6): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper halves of the lower ZMM registers (ZMM0-ZMM15 in 64-bit mode; otherwise ZMM0-ZMM7).
- XCR0.Hi16_ZMM (bit 7): If 1, Intel AVX-512 instructions can be executed and the XSAVE feature set can be used to manage the upper ZMM registers (ZMM16-ZMM31, only in 64-bit mode).
- XCR0.PKRU (bit 9): If 1, the XSAVE feature set can be used to manage the PKRU register (see Section 2.7).
- XCR0.TILECFG (bit 17): If 1, and if XCR0.TILEDATA is also 1, Intel AMX instructions can be executed and the XSAVE feature set can be used to manage TILECFG.
- XCR0.TILEDATA (bit 18): If 1, and if XCR0.TILECFG is also 1, Intel AMX instructions can be executed and the XSAVE feature set can be used to manage TILEDATA.

An attempt to use XSETBV to write to XCR0 results in general-protection exceptions (#GP) if it would do any of the following:

- Set a bit reserved in XCR0 for a given processor (as determined by the contents of EAX and EDX after executing CPUID.0DH.00H).
- Clear XCR0.x87.
- Clear XCR0.SSE and set XCR0.AVX.
- Clear XCR0.AVX and set any of XCR0.opmask, XCR0.ZMM_Hi256, or XCR0.Hi16_ZMM.
- Set either XCR0.BNDREG or XCR0.BNDCSR while not setting the other.
- Set any of XCR0.opmask, XCR0.ZMM_Hi256, and XCR0.Hi16_ZMM while not setting all of them.
- Set either XCR0.TILECFG or XCR0.TILEDATA while not setting the other.

After reset, all bits (except bit 0) in XCR0 are cleared to zero; XCR0[0] is set to 1.

2.7 PROTECTION-KEY RIGHTS REGISTERS (PKRU AND IA32_PKRS)

Processors may support either or both of two protection-key rights registers: PKRU for user-mode pages and the IA32_PKRS MSR (MSR index 6E1H) for supervisor-mode pages. 4-level paging and 5-level paging associate a 4-bit **protection key** with each page. The protection-key rights registers determine accessibility based on a page's protection key.

If CPUID.07H.00H:ECX.PKU[3] = 1, the processor supports the protection-key feature for user-mode pages. When CR4.PKE = 1, software can use the **protection-key rights register for user pages** (PKRU) to specify the access rights for user-mode pages for each protection key.

If CPUID.07H.00H:ECX.PKS[31] = 1, the processor supports the protection-key feature for supervisor-mode pages. When CR4.PKS = 1, software can use the **protection-key rights register for supervisor pages** (the IA32_PKRS MSR) to specify the access rights for supervisor-mode pages for each protection key.

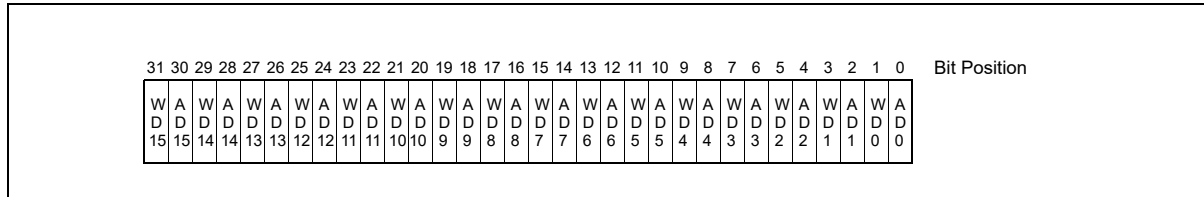


Figure 2-9. Format of Protection-Key Rights Registers

The format of each protection-key rights register is given in Figure 2-9. Each contains 16 pairs of disable controls to prevent data accesses to linear addresses (user-mode or supervisor-mode, depending on the register) based on their protection keys. Each protection key i ($0 \leq i \leq 15$) is associated with two bits in each protection-key rights register:

- Bit $2i$, shown as “AD i ” (access disable): if set, the processor prevents any data accesses to linear addresses (user-mode or supervisor-mode, depending on the register) with protection key i .
- Bit $2i+1$, shown as “WD i ” (write disable): if set, the processor prevents write accesses to linear addresses (user-mode or supervisor-mode, depending on the register) with protection key i .

(Bits 63:32 of the IA32_PKRS MSR are reserved and must be zero.)

See Section 5.6.2, “Protection Keys,” for details of how the processor uses the protection-key rights registers to control accesses to linear addresses.

Software can read and write PKRU using the RDPKRU and WRPKRU instructions. The IA32_PKRS MSR can be read and written with the RDMSR and WRMSR instructions. Writes to the IA32_PKRS MSR using WRMSR are not serializing.

2.8 SYSTEM INSTRUCTION SUMMARY

System instructions handle system-level functions such as loading system registers, managing the cache, managing interrupts, or setting up the debug registers. Many of these instructions can be executed only by operating-system or executive procedures (that is, procedures running at privilege level 0). Others can be executed at any privilege level and are thus available to application programs.

Table 2-3 lists the system instructions and indicates whether they are available and useful for application programs. These instructions are described in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D.

Table 2-3. Summary of System Instructions

Instruction	Description	Useful to Application?	Protected from Application?
LLDT	Load LDT Register	No	Yes
SLDT	Store LDT Register	No	If CR4.UMIP = 1
LGDT	Load GDT Register	No	Yes
SGDT	Store GDT Register	No	If CR4.UMIP = 1
LTR	Load Task Register	No	Yes
STR	Store Task Register	No	If CR4.UMIP = 1
LIDT	Load IDT Register	No	Yes
SIDT	Store IDT Register	No	If CR4.UMIP = 1
MOV CR n	Load and store control registers	No	Yes

Table 2-3. Summary of System Instructions (Contd.)

Instruction	Description	Useful to Application?	Protected from Application?
SMSW	Store MSW	Yes	If CR4.UMIP = 1
LMSW	Load MSW	No	Yes
CLTS	Clear TS flag in CR0	No	Yes
ARPL	Adjust RPL	Yes ^{1, 5}	No
LAR	Load Access Rights	Yes	No
LSL	Load Segment Limit	Yes	No
VERR	Verify for Reading	Yes	No
VERW	Verify for Writing	Yes	No
MOV DR _n	Load and store debug registers	No	Yes
INVD	Invalidate cache, no writeback	No	Yes
WBINVD	Invalidate cache, with writeback	No	Yes
INVLPG	Invalidate TLB entry	No	Yes
HLT	Halt Processor	No	Yes
LOCK (Prefix)	Bus Lock	Yes	No
RSM	Return from system management mode	No	Yes
RDMSR ³	Read Model-Specific Registers	No	Yes
WRMSR ³	Write Model-Specific Registers	No	Yes
RDPMS ⁴	Read Performance-Monitoring Counter	Yes	Yes ²
RDTSC ³	Read Time-Stamp Counter	Yes	Yes ²
RDTSCP ⁷	Read Serialized Time-Stamp Counter	Yes	Yes ²
XGETBV	Return the state of XCR0	Yes	No
XSETBV	Enable one or more processor extended states	No ⁶	Yes

NOTES:

1. Useful to application programs running at a CPL of 1 or 2.
2. The TSD and PCE flags in control register CR4 control access to these instructions by application programs running at a CPL of 3.
3. These instructions were introduced into the IA-32 Architecture with the Pentium processor.
4. This instruction was introduced into the IA-32 Architecture with the Pentium Pro processor and the Pentium processor with MMX technology.
5. This instruction is not supported in 64-bit mode.
6. Application uses XGETBV to query which set of processor extended states are enabled.
7. RDTSCP is introduced in Intel Core i7 processor.

2.8.1 Loading and Storing System Registers

The GDTR, LDTR, IDTR, and TR registers each have a load and store instruction for loading data into and storing data from the register:

- **LGDT (Load GDTR Register)** — Loads the GDT base address and limit from memory into the GDTR register.
- **SGDT (Store GDTR Register)** — Stores the GDT base address and limit from the GDTR register into memory.
- **LIDT (Load IDTR Register)** — Loads the IDT base address and limit from memory into the IDTR register.
- **SIDT (Store IDTR Register)** — Stores the IDT base address and limit from the IDTR register into memory.
- **LLDT (Load LDTR Register)** — Loads the LDT segment selector and segment descriptor from memory into the LDTR. (The segment selector operand can also be located in a general-purpose register.)

- **SLDT (Store LDTR Register)** — Stores the LDT segment selector from the LDTR register into memory or a general-purpose register.
- **LTR (Load Task Register)** — Loads segment selector and segment descriptor for a TSS from memory into the task register. (The segment selector operand can also be located in a general-purpose register.)
- **STR (Store Task Register)** — Stores the segment selector for the current task TSS from the task register into memory or a general-purpose register.

The LMSW (load machine status word) and SMSW (store machine status word) instructions operate on bits 0 through 15 of control register CR0. These instructions are provided for compatibility with the 16-bit Intel 286 processor. Programs written to run on 32-bit IA-32 processors should not use these instructions. Instead, they should access the control register CR0 using the MOV CR instruction.

The CLTS (clear TS flag in CR0) instruction is provided for use in handling a device-not-available exception (#NM) that occurs when the processor attempts to execute a floating-point instruction when the TS flag is set. This instruction allows the TS flag to be cleared after the x87 FPU context has been saved, preventing further #NM exceptions. See Section 2.5, “Control Registers,” for more information on the TS flag.

The control registers (CR0, CR1, CR2, CR3, CR4, and CR8) are loaded using the MOV instruction. The instruction loads a control register from a general-purpose register or stores the content of a control register in a general-purpose register.

2.8.2 Verifying of Access Privileges

The processor provides several instructions for examining segment selectors and segment descriptors to determine if access to their associated segments is allowed. These instructions duplicate some of the automatic access rights and type checking done by the processor, thus allowing operating-system or executive software to prevent exceptions from being generated.

The ARPL (adjust RPL) instruction adjusts the RPL (requestor privilege level) of a segment selector to match that of the program or procedure that supplied the segment selector. See Section 6.10.4, “Checking Caller Access Privileges (ARPL Instruction),” for a detailed explanation of the function and use of this instruction. Note that ARPL is not supported in 64-bit mode.

The LAR (load access rights) instruction verifies the accessibility of a specified segment and loads access rights information from the segment’s segment descriptor into a general-purpose register. Software can then examine the access rights to determine if the segment type is compatible with its intended use. See Section 6.10.1, “Checking Access Rights (LAR Instruction),” for a detailed explanation of the function and use of this instruction.

The LSL (load segment limit) instruction verifies the accessibility of a specified segment and loads the segment limit from the segment’s segment descriptor into a general-purpose register. Software can then compare the segment limit with an offset into the segment to determine whether the offset lies within the segment. See Section 6.10.3, “Checking That the Pointer Offset Is Within Limits (LSL Instruction),” for a detailed explanation of the function and use of this instruction.

The VERR (verify for reading) and VERW (verify for writing) instructions verify if a selected segment is readable or writable, respectively, at a given CPL. See Section 6.10.2, “Checking Read/Write Rights (VERR and VERW Instructions),” for a detailed explanation of the function and use of these instructions.

2.8.3 Loading and Storing Debug Registers

Internal debugging facilities in the processor are controlled by a set of 8 debug registers (DR0-DR7). The MOV instruction allows setup data to be loaded to and stored from these registers.

On processors that support Intel 64 architecture, debug registers DR0-DR7 are 64 bits. In 32-bit modes and compatibility mode, writes to a debug register fill the upper 32 bits with zeros. Reads return the lower 32 bits. In 64-bit mode, the upper 32 bits of DR6-DR7 are reserved and must be written with zeros. Writing one to any of the upper 32 bits causes an exception, #GP(0).

In 64-bit mode, MOV DRn instructions read or write all 64 bits of a debug register (operand-size prefixes are ignored). All 64 bits of DR0-DR3 are writable by software. However, MOV DRn instructions do not check that

addresses written to DR0-DR3 are in the limits of the implementation. Address matching is supported only on valid addresses generated by the processor implementation.

2.8.4 Invalidating Caches and TLBs

The processor provides several instructions for use in explicitly invalidating its caches and TLB entries. The INVD (invalidate cache with no writeback) instruction invalidates all data and instruction entries in the internal caches and sends a signal to the external caches indicating that they should also be invalidated.

The WBINVD (invalidate cache with writeback) instruction performs the same function as the INVD instruction, except that it writes back modified lines in its internal caches to memory before it invalidates the caches. After invalidating the caches local to the executing logical processor or processor core, WBINVD signals caches higher in the cache hierarchy (caches shared with the invalidating logical processor or core) to write back any data they have in modified state at the time of instruction execution and to invalidate their contents.

Note, non-shared caches may not be written back nor invalidated. In Figure 2-10 below, if code executing on either LP0 or LP1 were to execute a WBINVD, the shared L1 and L2 for LP0/LP1 will be written back and invalidated as will the shared L3. However, the L1 and L2 caches not shared with LP0 and LP1 will not be written back nor invalidated.

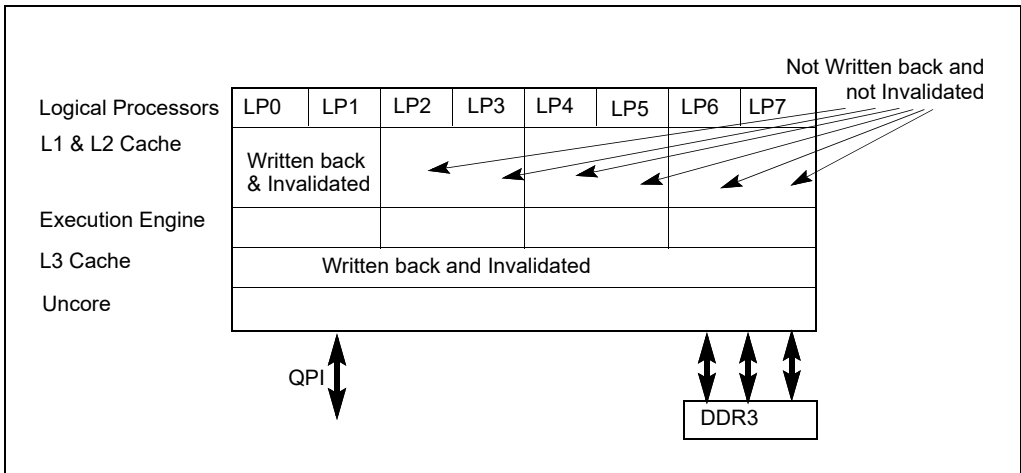


Figure 2-10. WBINVD Invalidation of Shared and Non-Shared Cache Hierarchy

The INVLPG (invalidate TLB entry) instruction invalidates (flushes) the TLB entry for a specified page.

2.8.5 Controlling the Processor

The HLT (halt processor) instruction stops the processor until an enabled interrupt (such as NMI or SMI, which are normally enabled), a debug exception, the BINIT# signal, the INIT# signal, or the RESET# signal is received. The processor generates a special bus cycle to indicate that the halt mode has been entered.

Hardware may respond to this signal in a number of ways. An indicator light on the front panel may be turned on. An NMI interrupt for recording diagnostic information may be generated. Reset initialization may be invoked (note that the BINIT# pin was introduced with the Pentium Pro processor). If any non-wake events are pending during shutdown, they will be handled after the wake event from shutdown is processed (for example, A20M# interrupts).

The LOCK prefix invokes a locked (atomic) read-modify-write operation when modifying a memory operand. This mechanism is used to allow reliable communications between processors in multiprocessor systems, as described below:

- In the Pentium processor and earlier IA-32 processors, the LOCK prefix causes the processor to assert the LOCK# signal during the instruction. This always causes an explicit bus lock to occur.

- In the Pentium 4, Intel Xeon, and P6 family processors, the locking operation is handled with either a cache lock or bus lock. If a memory access is cacheable and affects only a single cache line, a cache lock is invoked and the system bus and the actual memory location in system memory are not locked during the operation. Here, other Pentium 4, Intel Xeon, or P6 family processors on the bus write-back any modified data and invalidate their caches as necessary to maintain system memory coherency. If the memory access is not cacheable and/or it crosses a cache line boundary, the processor's LOCK# signal is asserted and the processor does not respond to requests for bus control during the locked operation.

The RSM (return from SMM) instruction restores the processor (from a context dump) to the state it was in prior to a system management mode (SMM) interrupt.

2.8.6 Reading Performance-Monitoring and Time-Stamp Counters

The RDPMC (read performance-monitoring counter) and RDTSC (read time-stamp counter) instructions allow application programs to read the processor's performance-monitoring and time-stamp counters, respectively. Processors based on Intel NetBurst® microarchitecture have eighteen 40-bit performance-monitoring counters; P6 family processors have two 40-bit counters. Intel Atom® processors and most of the processors based on the Intel Core microarchitecture support two types of performance monitoring counters: programmable performance counters similar to those available in the P6 family, and three fixed-function performance monitoring counters. Details of programmable and fixed-function performance monitoring counters for each processor generation are described in Chapter 20, "Last Branch Records."

The programmable performance counters can support counting either the occurrence or duration of events. Events that can be monitored on programmable counters generally are model specific (except for architectural performance events enumerated by CPUID.0AH); they may include the number of instructions decoded, interrupts received, or the number of cache loads. Individual counters can be set up to monitor different events. Use the system instruction WRMSR to set up values in one of the IA32_PERFEVTSELx MSR, in one of the 45 ESCRs and one of the 18 CCCR MSRs (for Pentium 4 and Intel Xeon processors); or in the PerfEvtSel0 or the PerfEvtSel1 MSR (for the P6 family processors). The RDPMC instruction loads the current count from the selected counter into the EDX:EAX registers.

Fixed-function performance counters record only specific events that are defined at: <https://perfmon-events.intel.com/>, and the width/number of fixed-function counters are enumerated by CPUID.0AH.

The time-stamp counter is a model-specific 64-bit counter that is reset to zero each time the processor is reset. If not reset, the counter will increment $\sim 9.5 \times 10^{16}$ times per year when the processor is operating at a clock rate of 3GHz. At this clock frequency, it would take over 190 years for the counter to wrap around. The RDTSC instruction loads the current count of the time-stamp counter into the EDX:EAX registers.

See Section 21.1, "Performance Monitoring Overview," and Section 19.17, "Time-Stamp Counter," for more information about the performance monitoring and time-stamp counters.

The RDTSC instruction was introduced into the IA-32 architecture with the Pentium processor. The RDPMC instruction was introduced into the IA-32 architecture with the Pentium Pro processor and the Pentium processor with MMX technology. Earlier Pentium processors have two performance-monitoring counters, but they can be read only with the RDMSR instruction, and only at privilege level 0.

2.8.6.1 Reading Counters in 64-Bit Mode

In 64-bit mode, RDTSC operates the same as in protected mode. The count in the time-stamp counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

RDPMC requires an index to specify the offset of the performance-monitoring counter. In 64-bit mode for Pentium 4 or Intel Xeon processor families, the index is specified in ECX[30:0]. The current count of the performance-monitoring counter is stored in EDX:EAX (or RDX[31:0]:RAX[31:0] with RDX[63:32]:RAX[63:32] cleared).

2.8.7 Reading and Writing Model-Specific Registers

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions allow a processor's 64-bit model-specific registers (MSRs) to be read and written, respectively. The MSR to be read or

written is specified by the value in the ECX register.

RDMSR reads the value from the specified MSR to the EDX:EAX registers; WRMSR writes the value in the EDX:EAX registers to the specified MSR. RDMSR and WRMSR were introduced into the IA-32 architecture with the Pentium processor.

See Section 11.4, “Model-Specific Registers (MSRs),” for more information.

2.8.7.1 Reading and Writing Model-Specific Registers in 64-Bit Mode

RDMSR and WRMSR require an index to specify the address of an MSR. In 64-bit mode, the index is 32 bits; it is specified using ECX.

2.8.8 Enabling Processor Extended States

The XSETBV instruction is required to enable OS support of individual processor extended states in XCR0 (see Section 2.6).

9. Updates to Chapter 3, Volume 3A

Change bars and violet text show changes to Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Added content for SEAM in Section 3.3.1, "Intel® 64 Processors and Physical Address Space".

CHAPTER 3

PROTECTED-MODE MEMORY MANAGEMENT

This chapter describes the Intel 64 and IA-32 architecture's protected-mode memory management facilities, including the physical memory requirements, segmentation mechanism, and paging mechanism.

See also: Chapter 6, "Protection," (for a description of the processor's protection mechanism) and Chapter 22, "8086 Emulation," (for a description of memory addressing protection in real-address and virtual-8086 modes).

3.1 MEMORY MANAGEMENT OVERVIEW

The memory management facilities of the IA-32 architecture are divided into two parts: segmentation and paging. Segmentation provides a mechanism of isolating individual code, data, and stack modules so that multiple programs (or tasks) can run on the same processor without interfering with one another. Paging provides a mechanism for implementing a conventional demand-paged, virtual-memory system where sections of a program's execution environment are mapped into physical memory as needed. Paging can also be used to provide isolation between multiple tasks. When operating in protected mode, some form of segmentation must be used. **There is no mode bit to disable segmentation.** The use of paging, however, is optional.

These two mechanisms (segmentation and paging) can be configured to support simple single-program (or single-task) systems, multitasking systems, or multiple-processor systems that used shared memory.

As shown in Figure 3-1, segmentation provides a mechanism for dividing the processor's addressable memory space (called the **linear address space**) into smaller protected address spaces called **segments**. Segments can be used to hold the code, data, and stack for a program or to hold system data structures (such as a TSS or LDT). If more than one program (or task) is running on a processor, each program can be assigned its own set of segments. The processor then enforces the boundaries between these segments and ensures that one program does not interfere with the execution of another program by writing into the other program's segments. The segmentation mechanism also allows typing of segments so that the operations that may be performed on a particular type of segment can be restricted.

All the segments in a system are contained in the processor's linear address space. To locate a byte in a particular segment, a **logical address** (also called a far pointer) must be provided. A logical address consists of a segment selector and an offset. The segment selector is a unique identifier for a segment. Among other things it provides an offset into a descriptor table (such as the global descriptor table, GDT) to a data structure called a segment descriptor. Each segment has a segment descriptor, which specifies the size of the segment, the access rights and privilege level for the segment, the segment type, and the location of the first byte of the segment in the linear address space (called the base address of the segment). The offset part of the logical address is added to the base address for the segment to locate a byte within the segment. The base address plus the offset thus forms a **linear address** in the processor's linear address space.

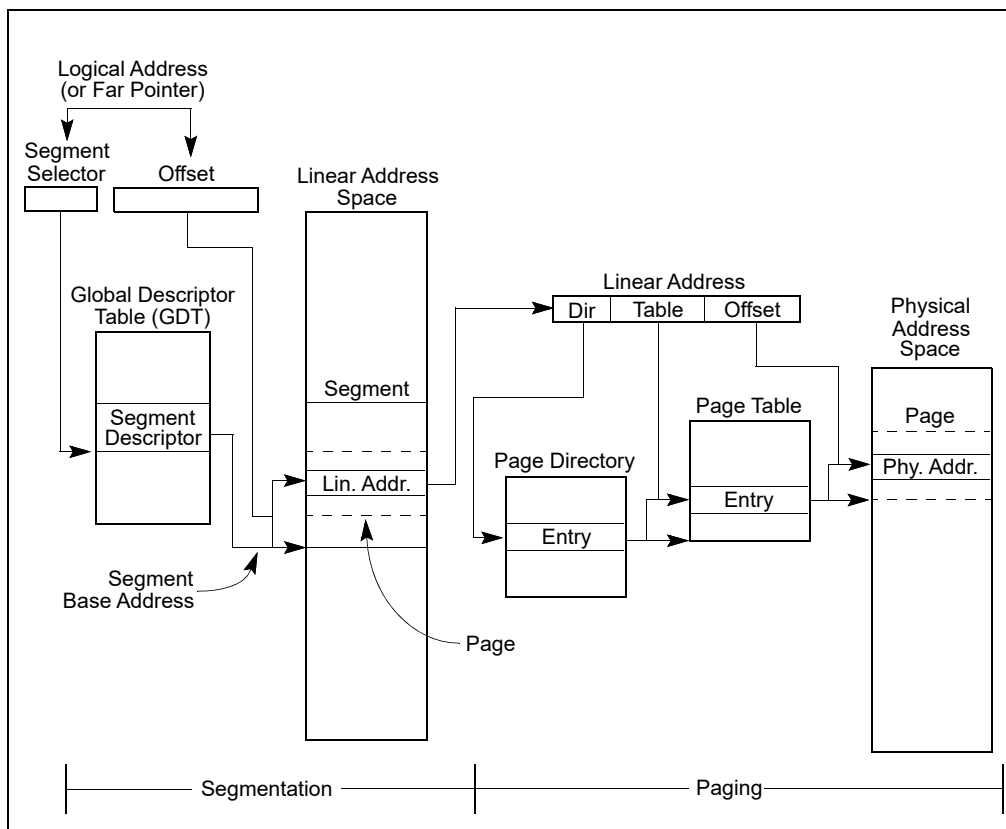


Figure 3-1. Segmentation and Paging

If paging is not used, the linear address space of the processor is mapped directly into the physical address space of processor. The physical address space is defined as the range of addresses that the processor can generate on its address bus.

Because multitasking computing systems commonly define a linear address space much larger than it is economically feasible to contain all at once in physical memory, some method of “virtualizing” the linear address space is needed. This virtualization of the linear address space is handled through the processor’s paging mechanism.

Paging supports a “virtual memory” environment where a large linear address space is simulated with a small amount of physical memory (RAM and ROM) and some disk storage. When using paging, each segment is divided into pages (typically 4 KBytes each in size), which are stored either in physical memory or on the disk. The operating system or executive maintains a page directory and a set of page tables to keep track of the pages. When a program (or task) attempts to access an address location in the linear address space, the processor uses the page directory and page tables to translate the linear address into a physical address and then performs the requested operation (read or write) on the memory location.

If the page being accessed is not currently in physical memory, the processor interrupts execution of the program (by generating a page-fault exception). The operating system or executive then reads the page into physical memory from the disk and continues executing the program.

When paging is implemented properly in the operating-system or executive, the swapping of pages between physical memory and the disk is transparent to the correct execution of a program. Even programs written for 16-bit IA-32 processors can be paged (transparently) when they are run in virtual-8086 mode.

3.2 USING SEGMENTS

The segmentation mechanism supported by the IA-32 architecture can be used to implement a wide variety of system designs. These designs range from flat models that make only minimal use of segmentation to protect

programs to multi-segmented models that employ segmentation to create a robust operating environment in which multiple programs and tasks can be executed reliably.

The following sections give several examples of how segmentation can be employed in a system to improve memory management performance and reliability.

3.2.1 Basic Flat Model

The simplest memory model for a system is the basic “flat model,” in which the operating system and application programs have access to a continuous, unsegmented address space. To the greatest extent possible, this basic flat model hides the segmentation mechanism of the architecture from both the system designer and the application programmer.

To implement a basic flat memory model with the IA-32 architecture, at least two segment descriptors must be created, one for referencing a code segment and one for referencing a data segment (see Figure 3-2). Both of these segments, however, are mapped to the entire linear address space: that is, both segment descriptors have the same base address value of 0 and the same segment limit of 4 GBytes. By setting the segment limit to 4 GBytes, the segmentation mechanism is kept from generating exceptions for out of limit memory references, even if no physical memory resides at a particular address. ROM (EPROM) is generally located at the top of the physical address space, because the processor begins execution at FFFF_FFF0H. RAM (DRAM) is placed at the bottom of the address space because the initial base address for the DS data segment after reset initialization is 0.

3.2.2 Protected Flat Model

The protected flat model is similar to the basic flat model, except the segment limits are set to include only the range of addresses for which physical memory actually exists (see Figure 3-3). A general-protection exception (#GP) is then generated on any attempt to access nonexistent memory. This model provides a minimum level of hardware protection against some kinds of program bugs.

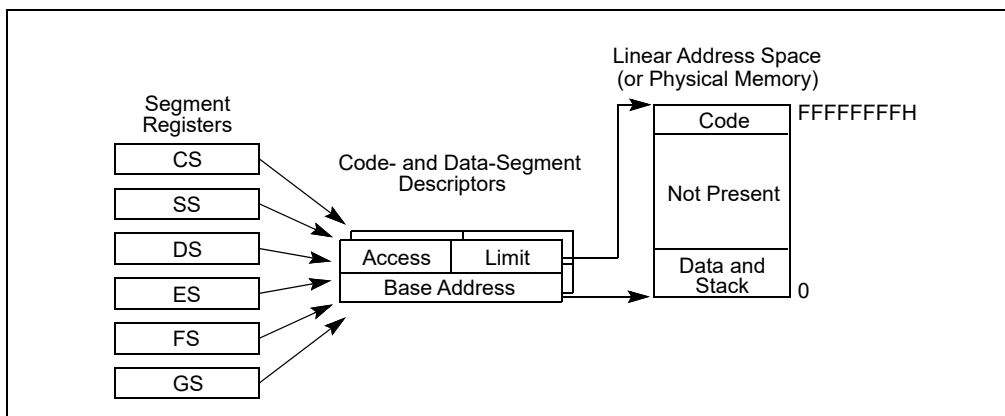


Figure 3-2. Flat Model

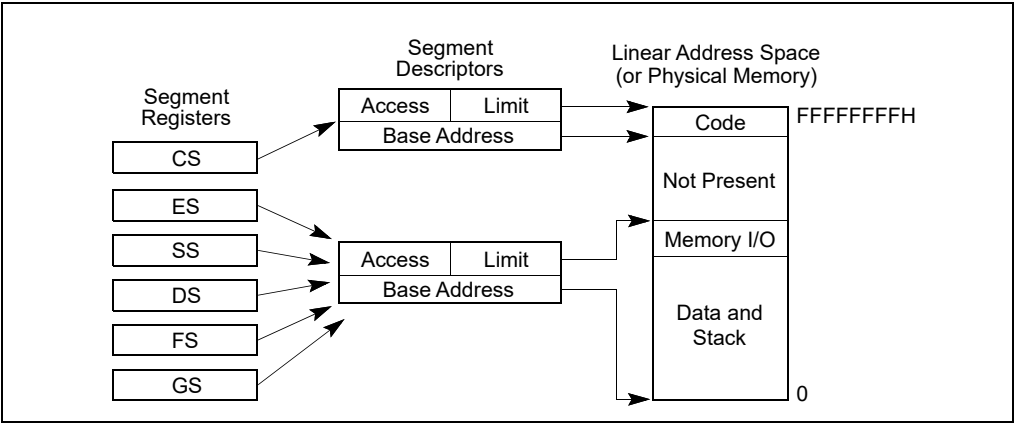


Figure 3-3. Protected Flat Model

More complexity can be added to this protected flat model to provide more protection. For example, for the paging mechanism to provide isolation between user and supervisor code and data, four segments need to be defined: code and data segments at privilege level 3 for the user, and code and data segments at privilege level 0 for the supervisor. Usually these segments all overlay each other and start at address 0 in the linear address space. This flat segmentation model along with a simple paging structure can protect the operating system from applications, and by adding a separate paging structure for each task or process, it can also protect applications from each other. Similar designs are used by several popular multitasking operating systems.

3.2.3 Multi-Segment Model

A multi-segment model (such as the one shown in Figure 3-4) uses the full capabilities of the segmentation mechanism to provide hardware enforced protection of code, data structures, and programs and tasks. Here, each program (or task) is given its own table of segment descriptors and its own segments. The segments can be completely private to their assigned programs or shared among programs. Access to all segments and to the execution environments of individual programs running on the system is controlled by hardware.

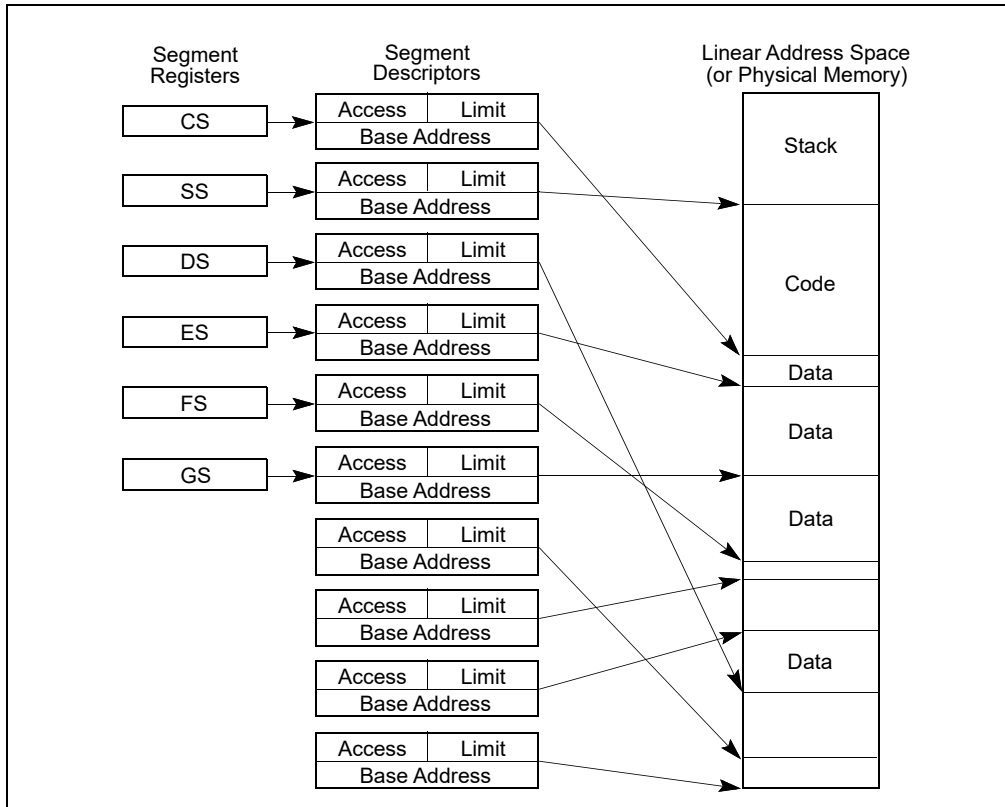


Figure 3-4. Multi-Segment Model

Access checks can be used to protect not only against referencing an address outside the limit of a segment, but also against performing disallowed operations in certain segments. For example, since code segments are designated as read-only segments, hardware can be used to prevent writes into code segments. The access rights information created for segments can also be used to set up protection rings or levels. Protection levels can be used to protect operating-system procedures from unauthorized access by application programs.

3.2.4 Segmentation in IA-32e Mode

In IA-32e mode of Intel 64 architecture, the effects of segmentation depend on whether the processor is running in compatibility mode or 64-bit mode. In compatibility mode, segmentation functions just as it does using legacy 16-bit or 32-bit protected mode semantics.

In 64-bit mode, segmentation is generally (but not completely) disabled, creating a flat 64-bit linear-address space. The processor treats the segment base of CS, DS, ES, SS as zero, creating a linear address that is equal to the effective address. The FS and GS segments are exceptions. These segment registers (which hold the segment base) can be used as additional base registers in linear address calculations. They facilitate addressing local data and certain operating system data structures.

Note that the processor does not perform segment limit checks at runtime in 64-bit mode.

3.2.5 Paging and Segmentation

Paging can be used with any of the segmentation models described in Figures 3-2, 3-3, and 3-4. The processor's paging mechanism divides the linear address space (into which segments are mapped) into pages (as shown in Figure 3-1). These linear-address-space pages are then mapped to pages in the physical address space. The paging mechanism offers several page-level protection facilities that can be used with or instead of the segment-

protection facilities. For example, it lets read-write protection be enforced on a page-by-page basis. The paging mechanism also provides two-level user-supervisor protection that can also be specified on a page-by-page basis.

3.3 PHYSICAL ADDRESS SPACE

In protected mode, the IA-32 architecture provides a normal physical address space of 4 GBytes (2^{32} bytes). This is the address space that the processor can address on its address bus. This address space is flat (unsegmented), with addresses ranging continuously from 0 to FFFFFFFFH. This physical address space can be mapped to read-write memory, read-only memory, and memory mapped I/O. The memory mapping facilities described in this chapter can be used to divide this physical memory up into segments and/or pages.

Starting with the Pentium Pro processor, the IA-32 architecture also supports an extension of the physical address space to 2^{36} bytes (64 GBytes); with a maximum physical address of FFFFFFFFH. This extension is invoked in either of two ways:

- Using the physical address extension (PAE) flag, located in bit 5 of control register CR4.
- Using the 36-bit page size extension (PSE-36) feature (introduced in the Pentium III processors).

Physical address support has since been extended beyond 36 bits. See Chapter 5, “Paging,” for more information about 36-bit physical addressing.

3.3.1 Intel® 64 Processors and Physical Address Space

On processors that support Intel 64 architecture (CPUID.80000001H:EDX[29] = 1), the size of the physical address range is implementation-specific and indicated by CPUID.80000008H:EAX[7:0].

For the format of information returned in EAX, see “CPUID—CPU Identification” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A. See also: Chapter 5, “Paging.”

MAXPHYADDR is derived from the value enumerated by CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.

3.4 LOGICAL AND LINEAR ADDRESSES

At the system-architecture level in protected mode, the processor uses two stages of address translation to arrive at a physical address: logical-address translation and linear address space paging.

Even with the minimum use of segments, every byte in the processor’s address space is accessed with a logical address. A logical address consists of a 16-bit segment selector and a 32-bit offset (see Figure 3-5). The segment selector identifies the segment the byte is located in and the offset specifies the location of the byte in the segment relative to the base address of the segment.

The processor translates every logical address into a linear address. A linear address is a 32-bit address in the processor’s linear address space. Like the physical address space, the linear address space is a flat (unsegmented), 2^{32} -byte address space, with addresses ranging from 0 to FFFFFFFFH. The linear address space contains all the segments and system tables defined for a system.

To translate a logical address into a linear address, the processor does the following:

1. Uses the offset in the segment selector to locate the segment descriptor for the segment in the GDT or LDT and reads it into the processor. (This step is needed only when a new segment selector is loaded into a segment register.)
2. Examines the segment descriptor to check the access rights and range of the segment to ensure that the segment is accessible and that the offset is within the limits of the segment.
3. Adds the base address of the segment from the segment descriptor to the offset to form a linear address.

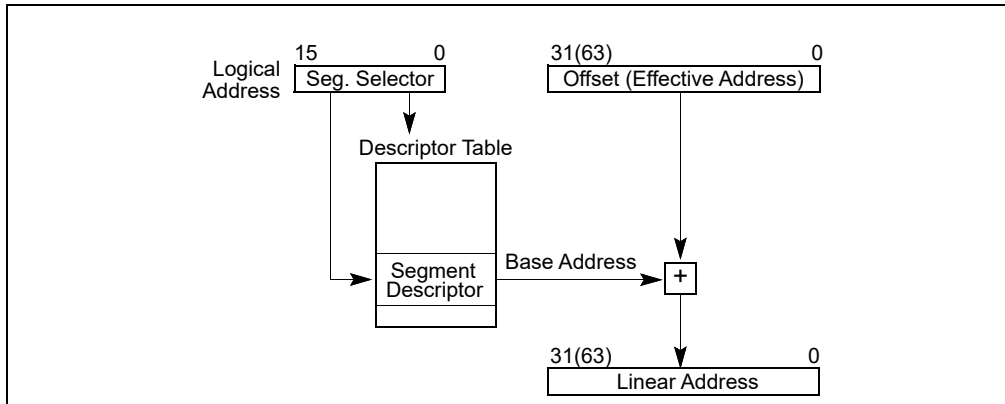


Figure 3-5. Logical Address to Linear Address Translation

If paging is not used, the processor maps the linear address directly to a physical address (that is, the linear address goes out on the processor's address bus). If the linear address space is paged, a second level of address translation is used to translate the linear address into a physical address.

See also: Chapter 5, "Paging."

3.4.1 Logical Address Translation in IA-32e Mode

In IA-32e mode, an Intel 64 processor uses the steps described above to translate a logical address to a linear address. In 64-bit mode, the offset and base address of the segment are 64-bits instead of 32 bits. The linear address format is also 64 bits wide and is subject to linear-address pre-processing (see Chapter 4).

Each code segment descriptor provides an L bit. This bit allows a code segment to execute 64-bit code or legacy 32-bit code by code segment.

3.4.2 Segment Selectors

A segment selector is a 16-bit identifier for a segment (see Figure 3-6). It does not point directly to the segment, but instead points to the segment descriptor that defines the segment. A segment selector contains the following items:

Index (Bits 3 through 15) — Selects one of 8192 descriptors in the GDT or LDT. The processor multiplies the index value by 8 (the number of bytes in a segment descriptor) and adds the result to the base address of the GDT or LDT (from the GDTR or LDTR register, respectively).

TI (table indicator) flag

(Bit 2) — Specifies the descriptor table to use: clearing this flag selects the GDT; setting this flag selects the current LDT.

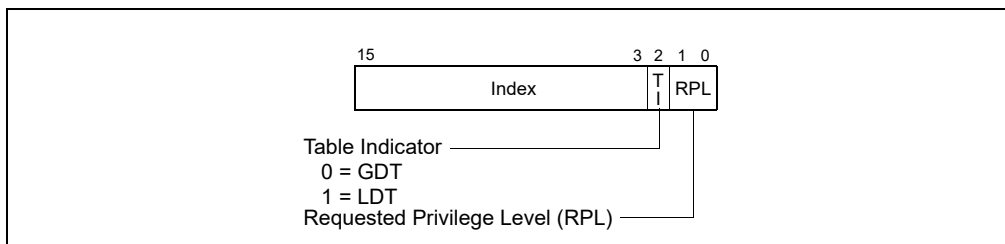


Figure 3-6. Segment Selector

Requested Privilege Level (RPL)

(Bits 0 and 1) — Specifies the privilege level of the selector. The privilege level can range from 0 to 3, with 0 being the most privileged level. See Section 6.5, “Privilege Levels,” for a description of the relationship of the RPL to the CPL of the executing program (or task) and the descriptor privilege level (DPL) of the descriptor the segment selector points to.

The first entry of the GDT is not used by the processor. A segment selector that points to this entry of the GDT (that is, a segment selector with an index of 0 and the TI flag set to 0) is used as a “null segment selector.” The processor does not generate an exception when a segment register (other than the CS or SS registers) is loaded with a null selector. It does, however, generate an exception when a segment register holding a null selector is used to access memory. A null selector can be used to initialize unused segment registers. Loading the CS or SS register with a null segment selector causes a general-protection exception (#GP) to be generated.

Segment selectors are visible to application programs as part of a pointer variable, but the values of selectors are usually assigned or modified by link editors or linking loaders, not application programs.

3.4.3 Segment Registers

To reduce address translation time and coding complexity, the processor provides registers for holding up to 6 segment selectors (see Figure 3-7). Each of these segment registers support a specific kind of memory reference (code, stack, or data). For virtually any kind of program execution to take place, at least the code-segment (CS), data-segment (DS), and stack-segment (SS) registers must be loaded with valid segment selectors. The processor also provides three additional data-segment registers (ES, FS, and GS), which can be used to make additional data segments available to the currently executing program (or task).

For a program to access a segment, the segment selector for the segment must have been loaded in one of the segment registers. So, although a system can define thousands of segments, only 6 can be available for immediate use. Other segments can be made available by loading their segment selectors into these registers during program execution.

Visible Part		Hidden Part	
Segment Selector	Base Address, Limit, Access Information		
			CS
			SS
			DS
			ES
			FS
			GS

Figure 3-7. Segment Registers

Every segment register has a “visible” part and a “hidden” part. (The hidden part is sometimes referred to as a “descriptor cache” or a “shadow register.”) When a segment selector is loaded into the visible part of a segment register, the processor also loads the hidden part of the segment register with the base address, segment limit, and access control information from the segment descriptor pointed to by the segment selector. The information cached in the segment register (visible and hidden) allows the processor to translate addresses without taking extra bus cycles to read the base address and limit from the segment descriptor. In systems in which multiple processors have access to the same descriptor tables, it is the responsibility of software to reload the segment registers when the descriptor tables are modified. If this is not done, an old segment descriptor cached in a segment register might be used after its memory-resident version has been modified.

Two kinds of load instructions are provided for loading the segment registers:

- 1. Direct load instructions such as the MOV, POP, LDS, LES, LSS, LGS, and LFS instructions. These instructions explicitly reference the segment registers.
- 2. Implied load instructions such as the far pointer versions of the CALL, JMP, and RET instructions, the SYSENTER and SYSEXIT instructions, and the IRET, INT *n*, INTO, INT3, and INT1 instructions. These instructions change

the contents of the CS register (and sometimes other segment registers) as an incidental part of their operation.

The MOV instruction can also be used to store the visible part of a segment register in a general-purpose register.

3.4.4 Segment Loading Instructions in IA-32e Mode

Because ES, DS, and SS segment registers are not used in 64-bit mode, their fields (base, limit, and attribute) in segment descriptor registers are ignored. Some forms of segment load instructions are also invalid (for example, LDS, POP ES). Address calculations that reference the ES, DS, or SS segments are treated as if the segment base is zero.

The processor performs linear-address pre-processing (Chapter 4) instead of performing limit checks. Mode switching does not change the contents of the segment registers or the associated descriptor registers. These registers are also not changed during 64-bit mode execution, unless explicit segment loads are performed.

In order to set up compatibility mode for an application, segment-load instructions (MOV to Sreg, POP Sreg) work normally in 64-bit mode. An entry is read from the system descriptor table (GDT or LDT) and is loaded in the hidden portion of the segment register. The descriptor-register base, limit, and attribute fields are all loaded. However, the contents of the data and stack segment selector and the descriptor registers are ignored.

When FS and GS segment overrides are used in 64-bit mode, their respective base addresses are used in the linear address calculation: (FS or GS).base + index + displacement. FS.base and GS.base are then expanded to the full linear-address size supported by the implementation. The resulting effective address calculation can wrap across positive and negative addresses; the resulting address is subject to linear-address pre-processing.

In 64-bit mode, memory accesses using FS-segment and GS-segment overrides are not checked for a runtime limit nor subjected to attribute-checking. Normal segment loads (MOV to Sreg and POP Sreg) into FS and GS load a standard 32-bit base value in the hidden portion of the segment register. The base address bits above the standard 32 bits are cleared to 0 to allow consistency for implementations that use less than 64 bits.

The hidden descriptor register fields for FS.base and GS.base are physically mapped to MSRs in order to load all address bits supported by a 64-bit implementation. Software with CPL = 0 (privileged software) can load all supported linear-address bits into FS.base or GS.base using WRMSR. Addresses written into the 64-bit FS.base and GS.base registers must be in canonical form. A WRMSR instruction that attempts to write a non-canonical address to those registers causes a #GP fault. See Section 4.5.2.

When in compatibility mode, FS and GS overrides operate as defined by 32-bit mode behavior regardless of the value loaded into the upper 32 linear-address bits of the hidden descriptor register base field. Compatibility mode ignores the upper 32 bits when calculating an effective address.

A new 64-bit mode instruction, SWAPGS, can be used to load GS base. SWAPGS exchanges the kernel data structure pointer from the IA32_KERNEL_GS_BASE MSR with the GS base register. The kernel can then use the GS prefix on normal memory references to access the kernel data structures. An attempt to write a non-canonical value (using WRMSR) to the IA32_KERNEL_GS_BASE MSR causes a #GP fault; see Section 4.5.2.

3.4.5 Segment Descriptors

A segment descriptor is a data structure in a GDT or LDT that provides the processor with the size and location of a segment, as well as access control and status information. Segment descriptors are typically created by compilers, linkers, loaders, or the operating system or executive, but not application programs. Figure 3-8 illustrates the general descriptor format for all types of segment descriptors.

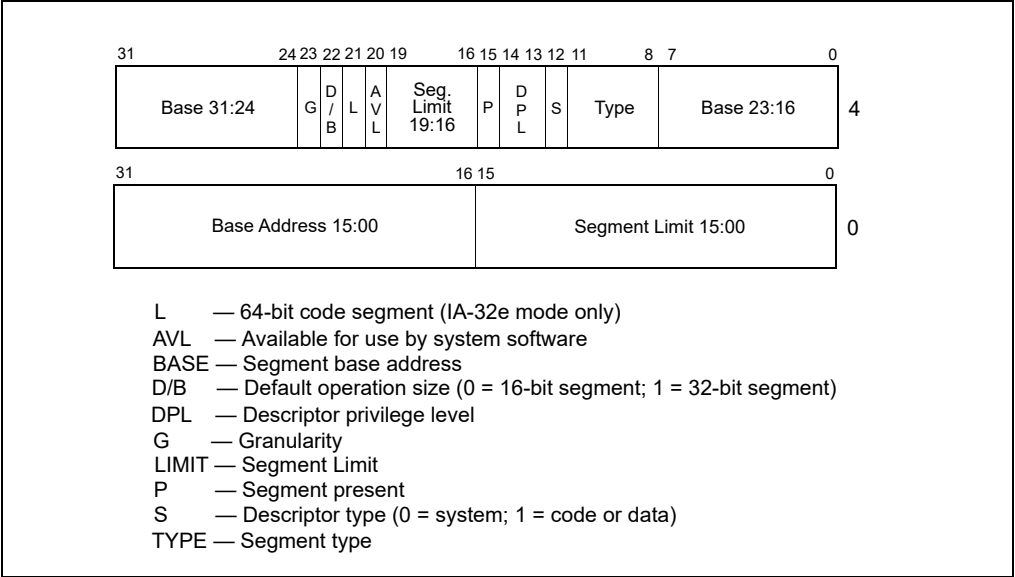


Figure 3-8. Segment Descriptor

The flags and fields in a segment descriptor are as follows:

Segment limit field

Specifies the size of the segment. The processor puts together the two segment limit fields to form a 20-bit value. The processor interprets the segment limit in one of two ways, depending on the setting of the G (granularity) flag:

- If the granularity flag is clear, the segment size can range from 1 byte to 1 MByte, in byte increments.
- If the granularity flag is set, the segment size can range from 4 KBytes to 4 GBytes, in 4-KByte increments.

The processor uses the segment limit in two different ways, depending on whether the segment is an expand-up or an expand-down segment. See Section 3.4.5.1, "Code- and Data-Segment Descriptor Types," for more information about segment types. For expand-up segments, the offset in a logical address can range from 0 to the segment limit. Offsets greater than the segment limit generate general-protection exceptions (#GP, for all segments other than SS) or stack-fault exceptions (#SS for the SS segment). For expand-down segments, the segment limit has the reverse function; the offset can range from the segment limit plus 1 to FFFFFFFFH or FFFFH, depending on the setting of the B flag. Offsets less than or equal to the segment limit generate general-protection exceptions or stack-fault exceptions. Decreasing the value in the segment limit field for an expand-down segment allocates new memory at the bottom of the segment's address space, rather than at the top. IA-32 architecture stacks always grow downwards, making this mechanism convenient for expandable stacks.

Base address fields

Defines the location of byte 0 of the segment within the 4-GByte linear address space. The processor puts together the three base address fields to form a single 32-bit value. Segment base addresses should be aligned to 16-byte boundaries. Although 16-byte alignment is not required, this alignment allows programs to maximize performance by aligning code and data on 16-byte boundaries.

Type field

Indicates the segment or gate type and specifies the kinds of access that can be made to the segment and the direction of growth. The interpretation of this field depends on whether the descriptor type flag specifies an application (code or data) descriptor or a system descriptor. The encoding of the type field is different for code, data, and system descriptors (see Figure 6-1). See Section 3.4.5.1, "Code- and Data-Segment Descriptor Types," for a description of how this field is used to specify code and data-segment types.

S (descriptor type) flag

Specifies whether the segment descriptor is for a system segment (S flag is clear) or a code or data segment (S flag is set).

DPL (descriptor privilege level) field

Specifies the privilege level of the segment. The privilege level can range from 0 to 3, with 0 being the most privileged level. The DPL is used to control access to the segment. See Section 6.5, "Privilege Levels," for a description of the relationship of the DPL to the CPL of the executing code segment and the RPL of a segment selector.

P (segment-present) flag

Indicates whether the segment is present in memory (set) or not present (clear). If this flag is clear, the processor generates a segment-not-present exception (#NP) when a segment selector that points to the segment descriptor is loaded into a segment register. Memory management software can use this flag to control which segments are actually loaded into physical memory at a given time. It offers a control in addition to paging for managing virtual memory.

Figure 3-9 shows the format of a segment descriptor when the segment-present flag is clear. When this flag is clear, the operating system or executive is free to use the locations marked "Available" to store its own data, such as information regarding the whereabouts of the missing segment.

D/B (default operation size/default stack pointer size and/or upper bound) flag

Performs different functions depending on whether the segment descriptor is an executable code segment, an expand-down data segment, or a stack segment. (This flag should always be set to 1 for 32-bit code and data segments and to 0 for 16-bit code and data segments.)

- **Executable code segment.** The flag is called the D flag and it indicates the default length for effective addresses and operands referenced by instructions in the segment. If the flag is set, 32-bit addresses and 32-bit or 8-bit operands are assumed; if it is clear, 16-bit addresses and 16-bit or 8-bit operands are assumed.
The instruction prefix 66H can be used to select an operand size other than the default, and the prefix 67H can be used select an address size other than the default.
- **Stack segment (data segment pointed to by the SS register).** The flag is called the B (big) flag and it specifies the size of the stack pointer used for implicit stack operations (such as pushes, pops, and calls). If the flag is set, a 32-bit stack pointer is used, which is stored in the 32-bit ESP register; if the flag is clear, a 16-bit stack pointer is used, which is stored in the 16-bit SP register. If the stack segment is set up to be an expand-down data segment (described in the next paragraph), the B flag also specifies the upper bound of the stack segment.
- **Expand-down data segment.** The flag is called the B flag and it specifies the upper bound of the segment. If the flag is set, the upper bound is FFFFFFFFH (4 GBytes); if the flag is clear, the upper bound is FFFFH (64 KBytes).

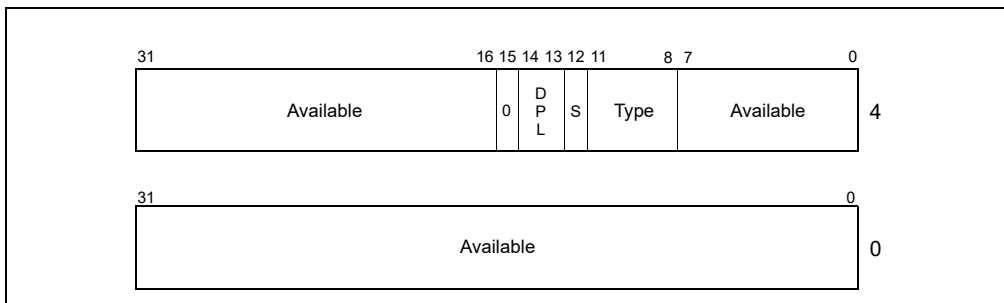


Figure 3-9. Segment Descriptor When Segment-Present Flag Is Clear

G (granularity) flag

Determines the scaling of the segment limit field. When the granularity flag is clear, the segment limit is interpreted in byte units; when flag is set, the segment limit is interpreted in 4-KByte units. (This flag does not affect the granularity of the base address; it is always byte granular.) When the granularity flag is set, the twelve least significant bits of an offset are not tested when checking the

offset against the segment limit. For example, when the granularity flag is set, a limit of 0 results in valid offsets from 0 to 4095.

L (64-bit code segment) flag

In IA-32e mode, bit 21 of the second doubleword of the segment descriptor indicates whether a code segment contains native 64-bit code. A value of 1 indicates instructions in this code segment are executed in 64-bit mode. A value of 0 indicates the instructions in this code segment are executed in compatibility mode. If the L-bit is set, then the D-bit must be cleared. Bit 21 is not used outside IA-32e mode (or for data segments). Because an attempt to activate IA-32e mode will fault if the current CS has the L-bit set (see Section 11.8.5), software operating outside IA-32e mode should avoid loading CS from a descriptor that sets the L-bit.

Available and reserved bits

Bit 20 of the second doubleword of the segment descriptor is available for use by system software.

3.4.5.1 Code- and Data-Segment Descriptor Types

When the S (descriptor type) flag in a segment descriptor is set, the descriptor is for either a code or a data segment. The highest order bit of the type field (bit 11 of the second double word of the segment descriptor) then determines whether the descriptor is for a data segment (clear) or a code segment (set).

For data segments, the three low-order bits of the type field (bits 8, 9, and 10) are interpreted as accessed (A), write-enable (W), and expansion-direction (E). See Table 3-1 for a description of the encoding of the bits in the type field for code and data segments. Data segments can be read-only or read/write segments, depending on the setting of the write-enable bit.

Table 3-1. Code- and Data-Segment Types

Type Field					Descriptor Type	Description
Decimal	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

Stack segments are data segments which must be read/write segments. Loading the SS register with a segment selector for a nonwritable data segment generates a general-protection exception (#GP). If the size of a stack segment needs to be changed dynamically, the stack segment can be an expand-down data segment (expansion-

direction flag set). Here, dynamically changing the segment limit causes stack space to be added to the bottom of the stack. If the size of a stack segment is intended to remain static, the stack segment may be either an expand-up or expand-down type.

The accessed bit indicates whether the segment has been accessed since the last time the operating-system or executive cleared the bit. The processor sets this bit whenever it loads a segment selector for the segment into a segment register, assuming that the type of memory that contains the segment descriptor supports processor writes. The bit remains set until explicitly cleared. This bit can be used both for virtual memory management and for debugging.

For code segments, the three low-order bits of the type field are interpreted as accessed (A), read enable (R), and conforming (C). Code segments can be execute-only or execute/read, depending on the setting of the read-enable bit. An execute/read segment might be used when constants or other static data have been placed with instruction code in a ROM. Here, data can be read from the code segment either by using an instruction with a CS override prefix or by loading a segment selector for the code segment in a data-segment register (the DS, ES, FS, or GS registers). In protected mode, code segments are not writable.

Code segments can be either conforming or nonconforming. A transfer of execution into a more-privileged conforming segment allows execution to continue at the current privilege level. A transfer into a nonconforming segment at a different privilege level results in a general-protection exception (#GP), unless a call gate or task gate is used (see Section 6.8.1, "Direct Calls or Jumps to Code Segments," for more information on conforming and nonconforming code segments). System utilities that do not access protected facilities and handlers for some types of exceptions (such as, divide error or overflow) may be loaded in conforming code segments. Utilities that need to be protected from less privileged programs and procedures should be placed in nonconforming code segments.

NOTE

Execution cannot be transferred by a call or a jump to a less-privileged (numerically higher privilege level) code segment, regardless of whether the target segment is a conforming or nonconforming code segment. Attempting such an execution transfer will result in a general-protection exception.

All data segments are nonconforming, meaning that they cannot be accessed by less privileged programs or procedures (code executing at numerically higher privilege levels). Unlike code segments, however, data segments can be accessed by more privileged programs or procedures (code executing at numerically lower privilege levels) without using a special access gate.

If the segment descriptors in the GDT or an LDT are placed in ROM, the processor can enter an indefinite loop if software or the processor attempts to update (write to) the ROM-based segment descriptors. To prevent this problem, set the accessed bits for all segment descriptors placed in a ROM. Also, remove operating-system or executive code that attempts to modify segment descriptors located in ROM.

3.5 SYSTEM DESCRIPTOR TYPES

When the S (descriptor type) flag in a segment descriptor is clear, the descriptor type is a system descriptor. The processor recognizes the following types of system descriptors:

- Local descriptor-table (LDT) segment descriptor.
- Task-state segment (TSS) descriptor.
- Call-gate descriptor.
- Interrupt-gate descriptor.
- Trap-gate descriptor.
- Task-gate descriptor.

These descriptor types fall into two categories: system-segment descriptors and gate descriptors. System-segment descriptors point to system segments (LDT and TSS segments). Gate descriptors are in themselves "gates," which hold pointers to procedure entry points in code segments (call, interrupt, and trap gates) or which hold segment selectors for TSS's (task gates).

Table 3-2 shows the encoding of the type field for system-segment descriptors and gate descriptors. Note that system descriptors in IA-32e mode are 16 bytes instead of 8 bytes.

Table 3-2. System-Segment and Gate-Descriptor Types

Type Field					Description	
Decimal	11	10	9	8	32-Bit Mode	IA-32e Mode
0	0	0	0	0	Reserved	Reserved
1	0	0	0	1	16-bit TSS (Available)	Reserved
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-bit TSS (Busy)	Reserved
4	0	1	0	0	16-bit Call Gate	Reserved
5	0	1	0	1	Task Gate	Reserved
6	0	1	1	0	16-bit Interrupt Gate	Reserved
7	0	1	1	1	16-bit Trap Gate	Reserved
8	1	0	0	0	Reserved	Reserved
9	1	0	0	1	32-bit TSS (Available)	64-bit TSS (Available)
10	1	0	1	0	Reserved	Reserved
11	1	0	1	1	32-bit TSS (Busy)	64-bit TSS (Busy)
12	1	1	0	0	32-bit Call Gate	64-bit Call Gate
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	32-bit Interrupt Gate	64-bit Interrupt Gate
15	1	1	1	1	32-bit Trap Gate	64-bit Trap Gate

See also: Section 3.5.1, “Segment Descriptor Tables,” and Section 9.2.2, “TSS Descriptor,” (for more information on the system-segment descriptors); see Section 6.8.3, “Call Gates,” Section 7.11, “IDT Descriptors,” and Section 9.2.5, “Task-Gate Descriptor,” (for more information on the gate descriptors).

3.5.1 Segment Descriptor Tables

A segment descriptor table is an array of segment descriptors (see Figure 3-10). A descriptor table is variable in length and can contain up to 8192 (2^{13}) 8-byte descriptors. There are two kinds of descriptor tables:

- The global descriptor table (GDT).
- The local descriptor tables (LDT).

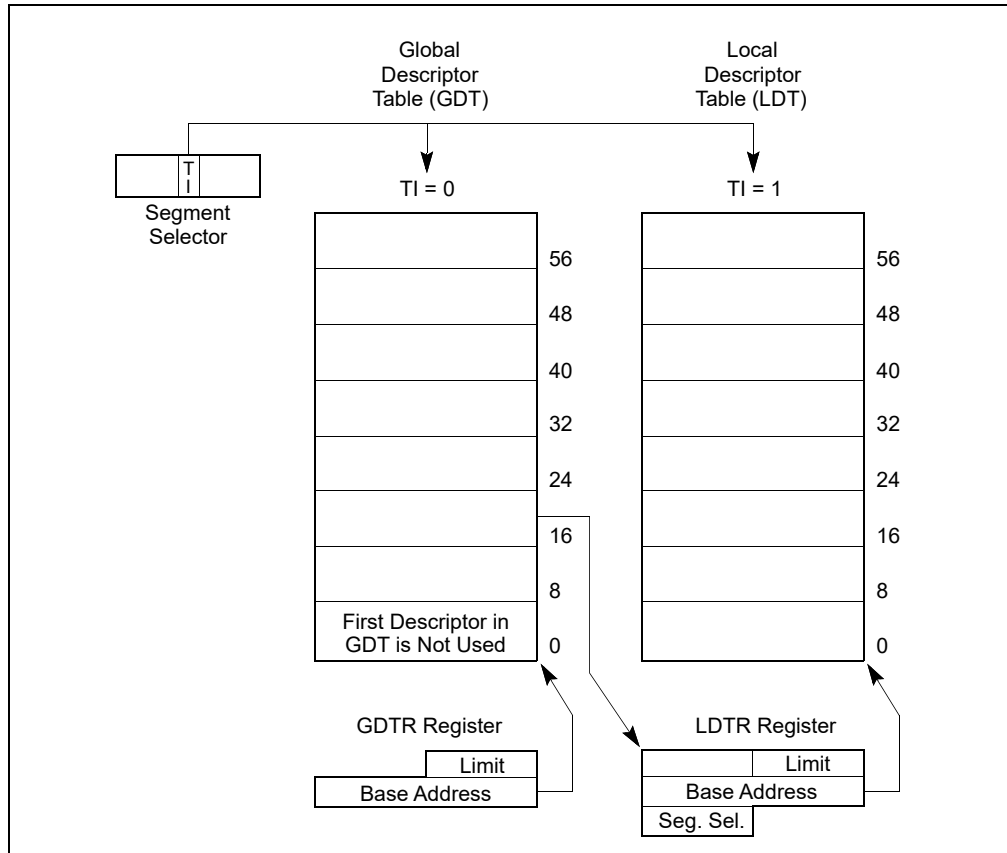


Figure 3-10. Global and Local Descriptor Tables

Each system must have one GDT defined, which may be used for all programs and tasks in the system. Optionally, one or more LDTs can be defined. For example, an LDT can be defined for each separate task being run, or some or all tasks can share the same LDT.

The GDT is not a segment itself; instead, it is a data structure in linear address space. The base linear address and limit of the GDT must be loaded into the GDTR register (see Section 2.4, “Memory-Management Registers”). The base address of the GDT should be aligned on an eight-byte boundary to yield the best processor performance. The limit value for the GDT is expressed in bytes. As with segments, the limit value is added to the base address to get the address of the last valid byte. A limit value of 0 results in exactly one valid byte. Because segment descriptors are always 8 bytes long, the GDT limit should always be one less than an integral multiple of eight (that is, $8N - 1$).

The first descriptor in the GDT is not used by the processor. A segment selector to this “null descriptor” does not generate an exception when loaded into a data-segment register (DS, ES, FS, or GS), but it always generates a general-protection exception (#GP) when an attempt is made to access memory using the descriptor. By initializing the segment registers with this segment selector, accidental reference to unused segment registers can be guaranteed to generate an exception.

The LDT is located in a system segment of the LDT type. The GDT must contain a segment descriptor for the LDT segment. If the system supports multiple LDTs, each must have a separate segment selector and segment descriptor in the GDT. The segment descriptor for an LDT can be located anywhere in the GDT. See Section 3.5, “System Descriptor Types,” for information on the LDT segment-descriptor type.

An LDT is accessed with its segment selector. To eliminate address translations when accessing the LDT, the segment selector, base linear address, limit, and access rights of the LDT are stored in the LDTR register (see Section 2.4, “Memory-Management Registers”).

When the GDTR register is stored (using the SGDT instruction), a 48-bit “pseudo-descriptor” is stored in memory (see top diagram in Figure 3-11). To avoid alignment check faults in user mode (privilege level 3), the pseudo-descriptor should be located at an odd word address (that is, address MOD 4 is equal to 2). This causes the

processor to store an aligned word, followed by an aligned doubleword. User-mode programs normally do not store pseudo-descriptors, but the possibility of generating an alignment check fault can be avoided by aligning pseudo-descriptors in this way. The same alignment should be used when storing the IDTR register using the SIDT instruction. When storing the LDTR or task register (using the SLDT or STR instruction, respectively), the pseudo-descriptor should be located at a doubleword address (that is, address MOD 4 is equal to 0).

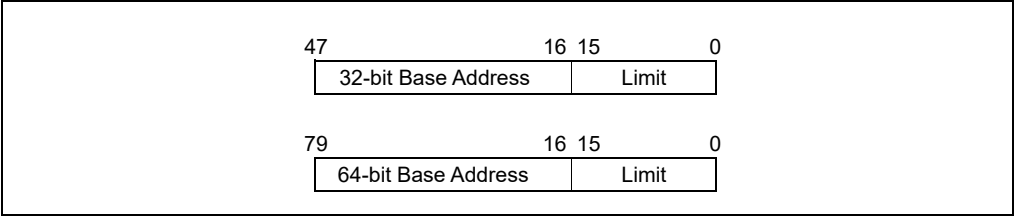


Figure 3-11. Pseudo-Descrptor Formats

3.5.2 Segment Descriptor Tables in IA-32e Mode

In IA-32e mode, a segment descriptor table can contain up to 8192 (2^{13}) 8-byte descriptors. An entry in the segment descriptor table can be 8 bytes. System descriptors are expanded to 16 bytes (occupying the space of two entries).

GDTR and LDTR registers are expanded to hold 64-bit base address. The corresponding pseudo-descriptor is 80 bits. (see the bottom diagram in Figure 3-11).

The following system descriptors expand to 16 bytes:

- Call gate descriptors (see Section 6.8.3.1, “IA-32e Mode Call Gates”).
- IDT gate descriptors (see Section 7.14.1, “64-Bit Mode IDT”).
- LDT and TSS descriptors (see Section 9.2.3, “TSS Descriptor in 64-bit mode”).

10. Updates to Chapter 5, Volume 3A

Change bars and violet text show changes to Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Added content for MAXPHYADDR under control bits CPUID.800000008H:EAX[7:0] in Section 5.1.4, "Enumeration of Paging Features by CPUID".
- Added clarification for MAXPHYADDR for reserved bits with PAE paging in Section 5.4.2, "Linear-Address Translation with PAE Paging".
- Added 5-level paging to footnote 1 in Section 5.5, "4-Level Paging and 5-Level Paging".
- Updated list of reserved bits in a paging-structure entry with bits 51:M in Section 5.5.4, "Linear-Address Translation with 4-Level Paging and 5-Level Paging".
- Added footnote for MAXPHYADDR for the following:
 - Table 5-8, "Format of a PAE Page-Directory-Pointer-Table Entry (PDPTE)"
 - Table 5-9, "Format of a PAE Page-Directory Entry that Maps a 2-MByte Page"
 - Table 5-10, "Format of a PAE Page-Directory Entry that References a Page Table"
 - Table 5-11, "Format of a PAE Page-Table Entry that Maps a 4-KByte Page"
 - Table 5-12, "Use of CR3 with 4-Level Paging and 5-level Paging and CR4.PCIDE = 0"
 - Table 5-13, "Use of CR3 with 4-Level Paging and 5-Level Paging and CR4.PCIDE = 1"
 - Table 5-14, "Format of a PML5 Entry (PML5E) that References a PML4 Table"
 - Table 5-15, "Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table"
 - Table 5-16, "Format of a Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page"
 - Table 5-17, "Format of a Page-Directory-Pointer-Table Entry (PDPTE) that References a Page Directory"
 - Table 5-18, "Format of a Page-Directory Entry that Maps a 2-MByte Page"
 - Table 5-19, "Format of a Page-Directory Entry that References a Page Table"
 - Table 5-20, "Format of a Page-Table Entry that Maps a 4-KByte Page"
 - Figure 5-7, "Formats of CR3 and Paging-Structure Entries with PAE Paging"
 - Figure 5-11, "Formats of CR3 and Paging-Structure Entries with 4-Level Paging and 5-Level Paging"

Chapter 3 explains how segmentation converts logical addresses to linear addresses. **Paging** (or linear-address translation) is the process of translating linear addresses so that they can be used to access memory or I/O devices. Paging translates each linear address to a **physical address** and determines, for each translation, what accesses to the linear address are allowed (the address's **access rights**) and the type of caching used for such accesses (the address's **memory type**).

Intel-64 processors support four different paging modes. These modes are identified and defined in Section 5.1. Section 5.2 gives an overview of the translation mechanism that is used in all modes. Section 5.3, Section 5.4, and Section 5.5 discuss the four paging modes in detail.

Section 5.6 details how paging determines and uses access rights. Section 5.7 discusses exceptions that may be generated by paging (page-fault exceptions). Section 5.8 considers data which the processor writes in response to linear-address accesses (accessed and dirty flags).

Section 5.9 describes how paging determines the memory types used for accesses to linear addresses. Section 5.10 provides details of how a processor may cache information about linear-address translation. Section 5.11 outlines interactions between paging and certain VMX features. Section 5.12 gives an overview of how paging can be used to implement virtual memory.

5.1 PAGING MODES AND CONTROL BITS

Paging behavior is controlled by the following control bits:

- The WP and PG flags in control register CR0 (bit 16 and bit 31, respectively).
- The PSE, PAE, PGE, LA57, PCIDE, SMEP, SMAP, PKE, CET, and PKS flags in control register CR4 (bit 4, bit 5, bit 7, bit 12, bit 17, bit 20, bit 21, bit 22, bit 23, and bit 24, respectively).
- The LME and NXE flags in the IA32_EFER MSR (bit 8 and bit 11, respectively).
- The AC flag in the EFLAGS register (bit 18).
- The “enable HLAT” VM-execution control (tertiary processor-based VM-execution control bit 1; see Section 26.6.2, “Processor-Based VM-Execution Controls,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C).

Software enables paging by using the MOV to CR0 instruction to set CR0.PG. Before doing so, software should ensure control register CR3 contains the physical address of the first paging structure that the processor will use for linear-address translation (see Section 5.2) and that structure is initialized as desired. See Table 5-3, Table 5-7, and Table 5-12 for the use of CR3 in the different paging modes.

Section 5.1.1 describes how the values of CR0.PG, CR4.PAE, CR4.LA57, and IA32_EFER.LME determine whether paging is enabled and, if so, which of four paging modes is in use. Section 5.1.2 explains how to manage these bits to establish or make changes in paging modes. Section 5.1.3 discusses how CR0.WP, CR4.PSE, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE modify the operation of the different paging modes.

5.1.1 Four Paging Modes

If CR0.PG = 0, paging is not used. The logical processor treats all linear addresses as if they were physical addresses. CR4.PAE, CR4.LA57, and IA32_EFER.LME are ignored by the processor, as are CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and IA32_EFER.NXE. (CR4.CET is also ignored insofar as it affects linear-address access rights.)

Paging is enabled if CR0.PG = 1. Paging can be enabled only if protection is enabled (CR0.PE = 1). If paging is enabled, one of four paging modes is used. The values of CR4.PAE, CR4.LA57, and IA32_EFER.LME determine which paging mode is used:

- If CR4.PAE = 0, **32-bit paging** is used. 32-bit paging is detailed in Section 5.3. 32-bit paging uses CR0.WP, CR4.PSE, CR4.PGE, CR4.SMEP, CR4.SMAP, and CR4.CET as described in Section 5.1.3 and Section 5.6.
- If CR4.PAE = 1 and IA32_EFER.LME = 0, **PAE paging** is used. PAE paging is detailed in Section 5.4. PAE paging uses CR0.WP, CR4.PGE, CR4.SMEP, CR4.SMAP, CR4.CET, and IA32_EFER.NXE as described in Section 5.1.3 and Section 5.6.
- If CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 0, **4-level paging**¹ is used.² 4-level paging is detailed in Section 5.5 (along with 5-level paging). 4-level paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE as described in Section 5.1.3 and Section 5.6.
- If CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 1, **5-level paging** is used. 5-level paging is detailed in Section 5.5 (along with 4-level paging). 5-level paging uses CR0.WP, CR4.PGE, CR4.PCIDE, CR4.SMEP, CR4.SMAP, CR4.PKE, CR4.CET, CR4.PKS, and IA32_EFER.NXE as described in Section 5.1.3 and Section 5.6.

NOTE

32-bit paging and PAE paging can be used only in legacy protected mode (IA32_EFER.LME = 0). In contrast, 4-level paging and 5-level paging can be used only in IA-32e mode (IA32_EFER.LME = 1).

The four paging modes differ with regard to the following details:

- Linear-address width. The size of the linear addresses that can be translated.
- Physical-address width. The size of the physical addresses produced by paging.
- Page size. The granularity at which linear addresses are translated. Linear addresses on the same page are translated to corresponding physical addresses on the same page.
- Support for execute-disable access rights. In some paging modes, software can be prevented from fetching instructions from pages that are otherwise readable.
- Support for PCIDs. With 4-level paging and 5-level paging, software can enable a facility by which a logical processor caches information for multiple linear-address spaces. The processor may retain cached information when software switches between different linear-address spaces.
- Support for protection keys. With 4-level paging and 5-level paging, each linear address is associated with a **protection key**. Software can use the protection-key rights registers to disable, for each protection key, how certain accesses to linear addresses associated with that protection key.

Table 5-1 illustrates the principal differences between the four paging modes.

Table 5-1. Properties of Different Paging Modes

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	LA57 in CR4	Lin.-Addr. Width	Phys.-Addr. Width ¹	Page Sizes	Supports Execute-Disable?	Supports PCIDs and protection keys?
None	0	N/A	N/A	N/A	32	32	N/A	No	No
32-bit	1	0	0 ²	N/A	32	Up to 40 ³	4 KB 4 MB ⁴	No	No
PAE	1	1	0	N/A	32	Up to 52	4 KB 2 MB	Yes ⁵	No
4-level	1	1	1	0	48	Up to 52	4 KB 2 MB 1 GB ⁶	Yes ⁵	Yes ⁷

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

2. The LMA flag in the IA32_EFER MSR (bit 10) is a status bit that indicates whether the logical processor is in IA-32e mode (and thus uses either 4-level paging or 5-level paging). The processor always sets IA32_EFER.LMA to CR0.PG & IA32_EFER.LME. Software cannot directly modify IA32_EFER.LMA; an execution of WRMSR to the IA32_EFER MSR ignores bit 10 of its source operand.

Table 5-1. Properties of Different Paging Modes (Contd.)

Paging Mode	PG in CR0	PAE in CR4	LME in IA32_EFER	LA57 in CR4	Lin.-Addr. Width	Phys.-Addr. Width ¹	Page Sizes	Supports Execute-Disable?	Supports PCIDs and protection keys?
5-level	1	1	1	1	57	Up to 52	4 KB 2 MB 1 GB ⁶	Yes ⁵	Yes ⁷

NOTES:

1. The physical-address width is always bounded by MAXPHYADDR; see Section 5.1.4.
2. The processor ensures that IA32_EFER.LME must be 0 if CR0.PG = 1 and CR4.PAE = 0.
3. 32-bit paging supports physical-address widths of more than 32 bits only for 4-MByte pages and only if the PSE-36 mechanism is supported; see Section 5.1.4 and Section 5.3.
4. 32-bit paging uses 4-MByte pages only if CR4.PSE = 1; see Section 5.3.
5. Execute-disable access rights are applied only if IA32_EFER.NXE = 1; see Section 5.6.
6. Processors that support 4-level paging or 5-level paging do not necessarily support 1-GByte pages; see Section 5.1.4.
7. PCIDs are used only if CR4.PCIDE = 1; see Section 5.10.1. Protection keys are used only if certain conditions hold; see Section 5.6.2.

Because 32-bit paging and PAE paging are used only in legacy protected mode and because legacy protected mode cannot produce linear addresses larger than 32 bits, 32-bit paging and PAE paging translate 32-bit linear addresses.

4-level paging and 5-level paging are used only in IA-32e mode. IA-32e mode has two sub-modes:

- Compatibility mode. This sub-mode uses only 32-bit linear addresses. In this sub-mode, 4-level paging and 5-level paging treat bits 63:32 of such an address as all 0. These addresses are subject to linear-address pre-processing, specifically linear-address-space separation (Section 4.3).
- 64-bit mode. This sub-mode produces 64-bit linear addresses. These addresses are then subject to linear-address pre-processing (Chapter 4). As part of this, the processor enforces **canonicity** (Section 4.5), ensuring that the upper bits of such an address are identical: bits 63:47 for 4-level paging and bits 63:56 for 5-level paging. 4-level paging (respectively, 5-level paging) does not use bits 63:48 (respectively, bits 63:57) of such addresses.

5.1.2 Paging-Mode Enabling

If CR0.PG = 1, a logical processor is in one of four paging modes, depending on the values of CR4.PAE, IA32_EFER.LME, and CR4.LA57. Figure 5-1 illustrates how software can enable these modes and make transitions between them. The following items identify certain limitations and other details:

- IA32_EFER.LME cannot be modified while paging is enabled (CR0.PG = 1). Attempts to do so using WRMSR cause a general-protection exception (#GP(0)).
- Paging cannot be enabled (by setting CR0.PG to 1) while CR4.PAE = 0 and IA32_EFER.LME = 1. Attempts to do so using MOV to CR0 cause a general-protection exception (#GP(0)).
- One node in Figure 5-1 is labeled “IA-32e mode.” This node represents either 4-level paging (if CR4.LA57 = 0) or 5-level paging (if CR4.LA57 = 1). As noted in the following items, software cannot modify CR4.LA57 (effecting transition between 4-level paging and 5-level paging) without first disabling paging.
- CR4.PAE and CR4.LA57 cannot be modified while either 4-level paging or 5-level paging is in use (when CR0.PG = 1 and IA32_EFER.LME = 1). Attempts to do so using MOV to CR4 cause a general-protection exception (#GP(0)).
- Regardless of the current paging mode, software can disable paging by clearing CR0.PG with MOV to CR0.¹

1. If the logical processor is in 64-bit mode or if CR4.PCIDE = 1, an attempt to clear CR0.PG causes a general-protection exception (#GP). Software should transition to compatibility mode and clear CR4.PCIDE before attempting to disable paging.

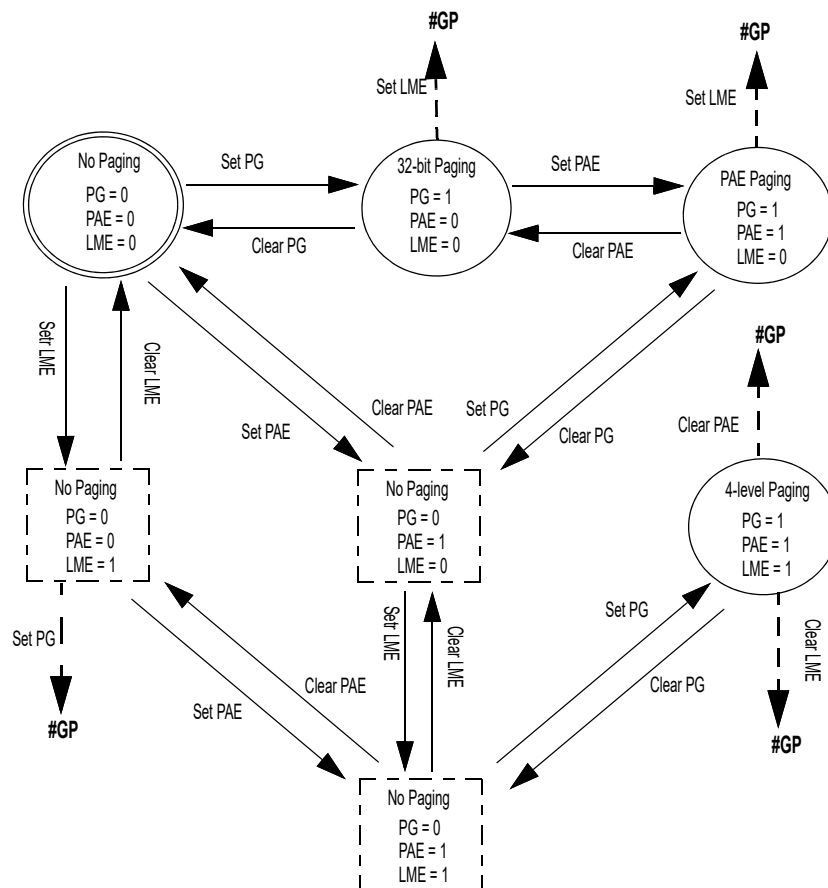


Figure 5-1. Enabling and Changing Paging Modes

- Software can transition between 32-bit paging and PAE paging by changing the value of CR4.PAE with MOV to CR4.
- Software cannot transition directly between 4-level paging (or 5-level paging) and any of other paging mode. It must first disable paging (by clearing CR0.PG with MOV to CR0), then set CR4.PAE, IA32_EFER.LME, and CR4.LA57 to the desired values (with MOV to CR4 and WRMSR), and then re-enable paging (by setting CR0.PG with MOV to CR0). As noted earlier, an attempt to modify CR4.PAE, IA32_EFER.LME, or CR4.LA57 while 4-level paging or 5-level paging is enabled causes a general-protection exception (#GP(0)).
- VMX transitions allow transitions between paging modes that are not possible using MOV to CR or WRMSR. This is because VMX transitions can load CR0, CR4, and IA32_EFER in one operation. See Section 5.11.1.

5.1.3 Paging-Mode Modifiers

Details of how each paging mode operates are determined by the following control bits:

- The WP flag in CR0 (bit 16).
- The PSE, PGE, PCIDE, SMEP, SMAP, PKE, CET, and PKS flags in CR4 (bit 4, bit 7, bit 17, bit 20, bit 21, bit 22, bit 23, and bit 24, respectively).
- The NXE flag in the IA32_EFER MSR (bit 11).
- The “enable HLAT” VM-execution control (tertiary processor-based VM-execution control bit 1).

CR0.WP allows pages to be protected from supervisor-mode writes. If CR0.WP = 0, supervisor-mode write accesses are allowed to linear addresses with read-only access rights; if CR0.WP = 1, they are not. (User-mode write accesses are never allowed to linear addresses with read-only access rights, regardless of the value of CR0.WP.) Section 5.6 explains how access rights are determined, including the definition of supervisor-mode and user-mode accesses.

CR4.PSE enables 4-MByte pages for 32-bit paging. If CR4.PSE = 0, 32-bit paging can use only 4-KByte pages; if CR4.PSE = 1, 32-bit paging can use both 4-KByte pages and 4-MByte pages. See Section 5.3 for more information. (PAE paging, 4-level paging, and 5-level paging can use multiple page sizes regardless of the value of CR4.PSE.)

CR4.PGE enables global pages. If CR4.PGE = 0, no translations are shared across address spaces; if CR4.PGE = 1, specified translations may be shared across address spaces. See Section 5.10.2.4 for more information.

CR4.PCIDE enables process-context identifiers (PCIDs) for 4-level paging and 5-level paging. PCIDs allow a logical processor to cache information for multiple linear-address spaces. See Section 5.10.1 for more information.

CR4.SMEP allows pages to be protected from supervisor-mode instruction fetches. If CR4.SMEP = 1, software operating in supervisor mode cannot fetch instructions from linear addresses that are accessible in user mode. Section 5.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

CR4.SMAP allows pages to be protected from supervisor-mode data accesses. If CR4.SMAP = 1, software operating in supervisor mode cannot access data at linear addresses that are accessible in user mode. Software can override this protection by setting EFLAGS.AC. Section 5.6 explains how access rights are determined, including the definition of supervisor-mode accesses and user-mode accessibility.

CR4.PKE and CR4.PKS enable specification of access rights based on **protection keys**. 4-level paging and 5-level paging associate each linear address with a protection key. When CR4.PKE = 1, the PKRU register specifies, for each protection key, whether user-mode linear addresses with that protection key can be read or written. When CR4.PKS = 1, the IA32_PKRS MSR does the same for supervisor-mode linear addresses. See Section 5.6 for more information.

CR4.CET enables **control-flow enforcement technology**, including the shadow-stack feature. If CR4.CET = 1, certain memory accesses are identified as **shadow-stack accesses** and certain linear addresses translate to **shadow-stack pages**. Section 5.6 explains how access rights are determined for these accesses and pages. (The processor allows CR4.CET to be set only if CR0.WP is also set.)

IA32_EFER.NXE enables execute-disable access rights for PAE paging, 4-level paging, and 5-level paging. If IA32_EFER.NXE = 1, instruction fetches can be prevented from specified linear addresses (even if data reads from the addresses are allowed). Section 5.6 explains how access rights are determined. (IA32_EFER.NXE has no effect with 32-bit paging. Software that wants to use this feature to limit instruction fetches from readable pages must use PAE paging, 4-level paging, or 5-level paging.)

The “enable HLAT” VM-execution control enables **HLAT paging** for 4-level paging and 5-level paging. HLAT paging does not use control register CR3 to identify the address of the first paging structure used for linear-address translation; instead, that structure is located using a field in the virtual-machine control structure (VMCS). In addition, HLAT paging interprets certain bits in paging-structure entries differently than ordinary paging. See Section 5.5 for details.

5.1.4 Enumeration of Paging Features by CPUID

Software can discover support for different paging features using the CPUID instruction:

- PSE: page-size extensions for 32-bit paging. If CPUID.01H:EDX.PSE[3] = 1, CR4.PSE may be set to 1, enabling support for 4-MByte pages with 32-bit paging (see Section 5.3).
- PAE: physical-address extension. If CPUID.01H:EDX.PAE[6] = 1, CR4.PAE may be set to 1, enabling PAE paging (this setting is also required for 4-level paging and 5-level paging).
- PGE: global-page support. If CPUID.01H:EDX.PGE[13] = 1, CR4.PGE may be set to 1, enabling the global-page feature (see Section 5.10.2.4).
- PAT: page-attribute table. If CPUID.01H:EDX.PAT[16] = 1, the 8-entry page-attribute table (PAT) is supported. When the PAT is supported, three bits in certain paging-structure entries select a memory type (used to determine type of caching used) from the PAT (see Section 5.9.2).

- PSE-36: page-size extensions with 40-bit physical-address extension. If CPUID.01H:EDX.PSE_36[17] = 1, the PSE-36 mechanism is supported, indicating that translations using 4-MByte pages with 32-bit paging may produce physical addresses with up to 40 bits (see Section 5.3).
- PCID: process-context identifiers. If CPUID.01H:ECX.PCID[17] = 1, CR4.PCIDE may be set to 1, enabling process-context identifiers (see Section 5.10.1).
- SMEP: supervisor-mode execution prevention. If CPUID.07H.00H:EBX.SMEP[7] = 1, CR4.SMEP may be set to 1, enabling supervisor-mode execution prevention (see Section 5.6).
- SMAP: supervisor-mode access prevention. If CPUID.07H.00H:EBX.SMAP[20] = 1, CR4.SMAP may be set to 1, enabling supervisor-mode access prevention (see Section 5.6).
- PKU: protection keys for user-mode pages. If CPUID.07H.00H:ECX.PKU[3] = 1, CR4.PKE may be set to 1, enabling protection keys for user-mode pages (see Section 5.6).
- OSPKE: enabling of protection keys for user-mode pages. CPUID.07H.00H:ECX.OSPKE[4] returns the value of CR4.PKE. Thus, protection keys for user-mode pages are enabled if this flag is 1 (see Section 5.6).
- CET: control-flow enforcement technology. If CPUID.07H.00H:ECX.CET_SS[7] = 1, CR4.CET may be set to 1, enabling shadow-stack pages (see Section 5.6).
- LA57: 57-bit linear addresses and 5-level paging. If CPUID.07H.00H:ECX.LA57[16] = 1, CR4.LA57 may be set to 1, enabling 5-level paging.
- PKS: protection keys for supervisor-mode pages. If CPUID.07H.00H:ECX.PKS[31] = 1, CR4.PKS may be set to 1, enabling protection keys for supervisor-mode pages (see Section 5.6).
- NX: execute disable. If CPUID.80000001H:EDX.EXECUTE_DIS[20] = 1, IA32_EFER.NXE may be set to 1, allowing software to disable execute access to selected pages (see Section 5.6). (Processors that do not support CPUID.80000001H do not allow IA32_EFER.NXE to be set to 1.)
- Page1GB: 1-GByte pages. If CPUID.80000001H:EDX.PAGE_1GB[26] = 1, 1-GByte pages may be supported with 4-level paging and 5-level paging (see Section 5.5).
- LM: IA-32e mode support. If CPUID.80000001H:EDX.INTEL64[29] = 1, IA32_EFER.LME may be set to 1, enabling IA-32e mode (with either 4-level paging or 5-level paging). (Processors that do not support CPUID.80000001H do not allow IA32_EFER.LME to be set to 1.)
- CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. (For processors that do not support CPUID.80000008H, the width is generally 36 if CPUID.01H:EDX.PAE[6] = 1 and 32 otherwise.) This width is at most 52 and is used to determine the value of MAXPHYADDR, which is used by paging.

If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.¹

For example, if CPUID.80000008H:EAX[7:0] = 48, IA32_TME_ACTIVATE[39:36] = 4, and TME has been configured, MAXPHYADDR is 44 outside SEAM (but remains 48 in SEAM).

If the logical processor is in VMX non-root operation and EPT is enabled (see Section 5.11.2), the addresses in CR3 and in paging-structure entries are treated as **guest-physical addresses**. The width of these addresses is limited only by the value enumerated by CPUID.80000008H:EAX[7:0] without any regard to TME or SEAM.

- CPUID.80000008H:EAX[15:8] reports the linear-address width supported by the processor. Generally, this value is reported as follows:
 - If CPUID.80000001H:EDX.INTEL64[29] = 0, the value is reported as 32.
 - If CPUID.80000001H:EDX.INTEL64[29] = 1 and CPUID.07H.00H:ECX.LA57[16] = 0, the value is reported as 48.
 - If CPUID.07H.00H:ECX.LA57[16] = 1, the value is reported as 57.
 (Processors that do not support CPUID.80000008H, support a linear-address width of 32.)

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

5.2 HIERARCHICAL PAGING STRUCTURES: AN OVERVIEW

All four paging modes translate linear addresses using **hierarchical paging structures**. This section provides an overview of their operation. Section 5.3, Section 5.4, Section 5.5, and Section 5.6 provide details for the four paging modes.

Every paging structure is 4096 Bytes in size and comprises a number of individual **entries**. With 32-bit paging, each entry is 32 bits (4 bytes); there are thus 1024 entries in each structure. With the other paging modes, each entry is 64 bits (8 bytes); there are thus 512 entries in each structure. (PAE paging includes one exception, a paging structure that is 32 bytes in size, containing 4 64-bit entries.)

The processor uses the upper portion of a linear address to identify a series of paging-structure entries. The last of these entries identifies the physical address of the region to which the linear address translates (called the **page frame**). The lower portion of the linear address (called the **page offset**) identifies the specific address within that region to which the linear address translates.

Each paging-structure entry contains a physical address, which is either the address of another paging structure or the address of a page frame. In the first case, the entry is said to **reference** the other paging structure; in the latter, the entry is said to **map a page**.

The first paging structure used for any translation is located at the physical address in CR3.¹ A linear address is translated using the following iterative procedure. A portion of the linear address (initially the uppermost bits) selects an entry in a paging structure (initially the one located using CR3). If that entry references another paging structure, the process continues with that paging structure and with the portion of the linear address immediately below that just used. If instead the entry maps a page, the process completes: the physical address in the entry is that of the page frame and the remaining lower portion of the linear address is the page offset.

The following items give an example for each of the four paging modes (each example locates a 4-KByte page frame):

- With 32-bit paging, each paging structure comprises $1024 = 2^{10}$ entries. For this reason, the translation process uses 10 bits at a time from a 32-bit linear address. Bits 31:22 identify the first paging-structure entry and bits 21:12 identify a second. The latter identifies the page frame. Bits 11:0 of the linear address are the page offset within the 4-KByte page frame. (See Figure 5-2 for an illustration.)
- With PAE paging, the first paging structure comprises only $4 = 2^2$ entries. Translation thus begins by using bits 31:30 from a 32-bit linear address to identify the first paging-structure entry. Other paging structures comprise $512 = 2^9$ entries, so the process continues by using 9 bits at a time. Bits 29:21 identify a second paging-structure entry and bits 20:12 identify a third. This last identifies the page frame. (See Figure 5-5 for an illustration.)
- With 4-level paging, each paging structure comprises $512 = 2^9$ entries and translation uses 9 bits at a time from a 48-bit linear address. Bits 47:39 identify the first paging-structure entry, bits 38:30 identify a second, bits 29:21 a third, and bits 20:12 identify a fourth. Again, the last identifies the page frame. (See Figure 5-8 for an illustration.)
- 5-level paging is similar to 4-level paging except that 5-level paging translates 57-bit linear addresses. Bits 56:48 identify the first paging-structure entry, while the remaining bits are used as with 4-level paging.

The translation process in each of the examples above completes by identifying a page frame; the page frame is part of the **translation** of the original linear address. In some cases, however, the paging structures may be configured so that the translation process terminates before identifying a page frame. This occurs if the process encounters a paging-structure entry that is marked “not present” (because its P flag — bit 0 — is clear) or in which a reserved bit is set. In this case, there is no translation for the linear address; an access to that address causes a page-fault exception (see Section 5.7).

In the examples above, a paging-structure entry maps a page with a 4-KByte page frame when only 12 bits remain in the linear address; entries identified earlier always reference other paging structures. That may not apply in other cases. The following items identify when an entry maps a page and when it references another paging structure:

- If more than 12 bits remain in the linear address, bit 7 (PS — page size) of the current paging-structure entry is consulted. If the bit is 0, the entry references another paging structure; if the bit is 1, the entry maps a page.

1. If HLAT paging is in use, a different mechanism is used to identify the first paging structure. See Section 5.5 for more information.

- If only 12 bits remain in the linear address, the current paging-structure entry always maps a page (bit 7 is used for other purposes).

If a paging-structure entry maps a page when more than 12 bits remain in the linear address, the entry identifies a page frame larger than 4 KBytes. For example, 32-bit paging uses the upper 10 bits of a linear address to locate the first paging-structure entry; 22 bits remain. If that entry maps a page, the page frame is 2^{22} Bytes = 4 MBytes. 32-bit paging can use 4-MByte pages if CR4.PSE = 1. The other paging modes can use 2-MByte pages (regardless of the value of CR4.PSE). 4-level paging and 5-level paging can use 1-GByte pages if the processor supports them (see Section 5.1.4).

Paging structures are given different names based on their uses in the translation process. Table 5-2 gives the names of the different paging structures. It also provides, for each structure, the source of the physical address used to locate it (CR3 or a different paging-structure entry); the bits in the linear address used to select an entry from the structure; and details of whether and how such an entry can map a page.

Table 5-2. Paging Structures in the Different Paging Modes

Paging Structure	Entry Name	Paging Mode	Physical Address of Structure	Bits Selecting Entry	Page Mapping
PML5 table	PML5E	32-bit, PAE, 4-level	N/A		
		5-level	CR3 ¹	56:48	N/A (PS must be 0)
PML4 table	PML4E	32-bit, PAE	N/A		
		4-level	CR3 ¹	47:39	N/A (PS must be 0)
		5-level	PML5E		
Page-directory-pointer table	PDPTE	32-bit	N/A		
		PAE	CR3	31:30	N/A (PS must be 0)
		4-level, 5-level	PML4E	38:30	1-GByte page if PS = 1 ²
Page directory	PDE	32-bit	CR3	31:22	4-MByte page if PS = 1 ³
		PAE, 4-level, 5-level	PDPTE	29:21	2-MByte page if PS = 1
Page table	PTE	32-bit	PDE	21:12	4-KByte page
		PAE, 4-level, 5-level		20:12	

NOTES:

1. If HLAT paging is in use, a different mechanism is used to identify the first paging structure. See Section 5.5 for more information.
2. Not all processors support 1-GByte pages; see Section 5.1.4.
3. 32-bit paging ignores the PS flag in a PDE (and uses the entry to reference a page table) unless CR4.PSE = 1. Not all processors support 4-MByte pages with 32-bit paging; see Section 5.1.4.

5.3 32-BIT PAGING

A logical processor uses 32-bit paging if $CR0.PG = 1$ and $CR4.PAE = 0$. 32-bit paging translates 32-bit linear addresses to 40-bit physical addresses.¹ Although 40 bits corresponds to 1 TByte, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

32-bit paging uses a hierarchy of paging structures to produce a translation for a linear address. CR3 is used to locate the first paging-structure, the page directory. Table 5-3 illustrates how CR3 is used with 32-bit paging.

32-bit paging may map linear addresses to either 4-KByte pages or 4-MByte pages. Figure 5-2 illustrates the translation process when it uses a 4-KByte page; Figure 5-3 covers the case of a 4-MByte page. The following items describe the 32-bit paging process in more detail as well as how the page size is determined:

- A 4-KByte naturally aligned page directory is located at the physical address specified in bits 31:12 of CR3 (see Table 5-3). A page directory comprises 1024 32-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
 - Bits 39:32 are all 0.
 - Bits 31:12 are from CR3.
 - Bits 11:2 are bits 31:22 of the linear address.
 - Bits 1:0 are 0.

Because a PDE is identified using bits 31:22 of the linear address, it controls access to a 4-Mbyte region of the linear-address space. Use of the PDE depends on CR4.PSE and the PDE's PS flag (bit 7):

- If $CR4.PSE = 1$ and the PDE's PS flag is 1, the PDE maps a 4-MByte page (see Table 5-4). The final physical address is computed as follows:
 - Bits 39:32 are bits 20:13 of the PDE.
 - Bits 31:22 are bits 31:22 of the PDE.²
 - Bits 21:0 are from the original linear address.
- If $CR4.PSE = 0$ or the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 31:12 of the PDE (see Table 5-5). A page table comprises 1024 32-bit entries (PTEs). A PTE is selected using the physical address defined as follows:
 - Bits 39:32 are all 0.
 - Bits 31:12 are from the PDE.
 - Bits 11:2 are bits 21:12 of the linear address.
 - Bits 1:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 5-6). The final physical address is computed as follows:
 - Bits 39:32 are all 0.
 - Bits 31:12 are from the PTE.
 - Bits 11:0 are from the original linear address.

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 5.7).

-
1. Bits in the range 39:32 are 0 in any physical address used by 32-bit paging except those used to map 4-MByte pages. If the processor does not support the PSE-36 mechanism, this is true also for physical addresses used to map 4-MByte pages. If the processor does support the PSE-36 mechanism and $MAXPHYADDR < 40$, bits in the range 39:MAXPHYADDR are 0 in any physical address used to map a 4-MByte page. (The corresponding bits are reserved in PDEs.) See Section 5.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.
 2. The upper bits in the final physical address do not all come from corresponding positions in the PDE; the physical-address bits in the PDE are not all contiguous.

With 32-bit paging, there are reserved bits only if CR4.PSE = 1:

- If the P flag and the PS flag (bit 7) of a PDE are both 1, the bits reserved depend on MAXPHYADDR, and whether the PSE-36 mechanism is supported:¹
 - If the PSE-36 mechanism is not supported, bits 21:13 are reserved.
 - If the PSE-36 mechanism is supported, bits 21:M-19 are reserved, where M is the minimum of 40 and MAXPHYADDR.
- If the PAT is not supported:²
 - If the P flag of a PTE is 1, bit 7 is reserved.
 - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

(If CR4.PSE = 0, no bits are reserved with 32-bit paging.)

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 5.6.

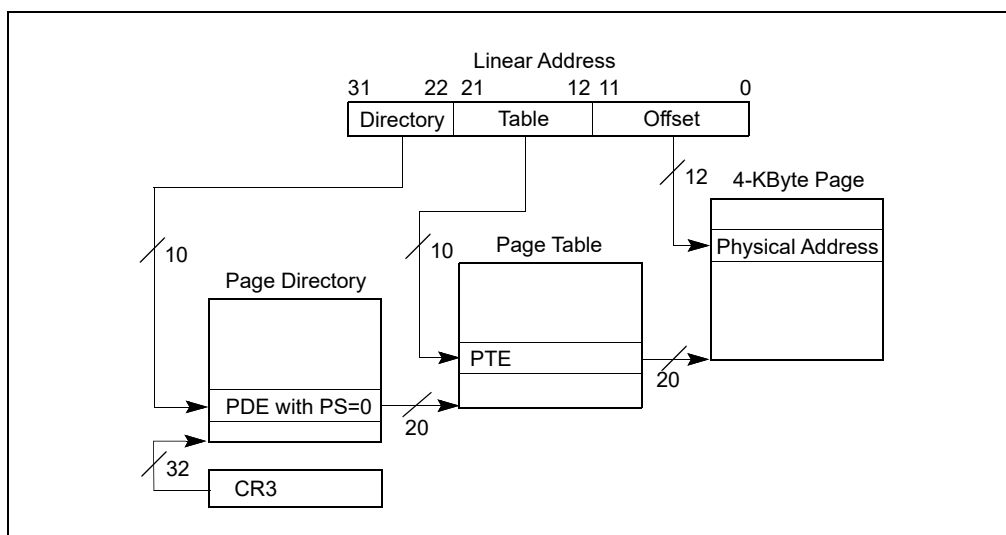


Figure 5-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

1. See Section 5.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

2. See Section 5.1.4 for how to determine whether the PAT is supported.

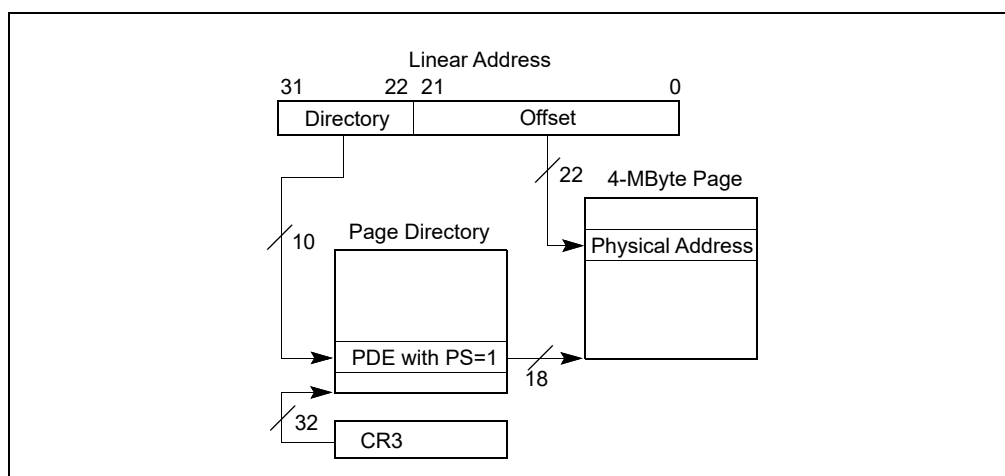


Figure 5-3. Linear-Address Translation to a 4-MByte Page using 32-Bit Paging

Figure 5-4 gives a summary of the formats of CR3 and the paging-structure entries with 32-bit paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted because they determine how such an entry is used.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory ¹																				Ignored					P C D	P W T	Ignored			CR3		
Bits 31:22 of address of 4MB page frame										Reserved (must be 0)				Bits 39:32 of address ²		P A T	Ignored	G	1	D	A	P C D	P W T	U / S	R / W	1	PDE: 4MB page					
Address of page table																				Ignored			0	I g n	A	P C D	P W T	U / S	R / W	1	PDE: page table	
Ignored																										0	PDE: not present					
Address of 4KB page frame																				Ignored		G	P A T	D	A	P C D	P W T	U / S	R / W	1	PTE: 4KB page	
Ignored																										0	PTE: not present					

Figure 5-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

NOTES:

1. CR3 has 64 bits on processors supporting the Intel-64 architecture. These bits are ignored with 32-bit paging.
2. This example illustrates a processor in which MAXPHYADDR is 36. If this value is larger or smaller, the number of bits reserved in positions 20:13 of a PDE mapping a 4-MByte page will change.

Table 5-3. Use of CR3 with 32-Bit Paging

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory during linear-address translation (see Section 5.9)
11:5	Ignored
31:12	Physical address of the 4-KByte aligned page directory used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

Table 5-4. Format of a 32-Bit Page-Directory Entry that Maps a 4-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 5.9)
5 (A)	Accessed; indicates whether software has accessed the 4-MByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 4-MByte page referenced by this entry (see Section 5.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 5-5)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-MByte page referenced by this entry (see Section 5.9.2); otherwise, reserved (must be 0) ¹
M-20:13	Bits M-1:32 of physical address of the 4-MByte page referenced by this entry ²
21:M-19	Reserved (must be 0)
31:22	Bits 31:22 of physical address of the 4-MByte page referenced by this entry

NOTES:

1. See Section 5.1.4 for how to determine whether the PAT is supported.
2. If the PSE-36 mechanism is not supported, M is 32, and this row does not apply. If the PSE-36 mechanism is supported, M is the minimum of 40 and MAXPHYADDR (this row does not apply if MAXPHYADDR = 32). See Section 5.1.4 for how to determine MAXPHYADDR and whether the PSE-36 mechanism is supported.

Table 5-5. Format of a 32-Bit Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte region controlled by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte region controlled by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 5.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 5.8)
6	Ignored
7 (PS)	If CR4.PSE = 1, must be 0 (otherwise, this entry maps a 4-MByte page; see Table 5-4); otherwise, ignored
11:8	Ignored
31:12	Physical address of 4-KByte aligned page table referenced by this entry

Table 5-6. Format of a 32-Bit Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 5.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2); otherwise, reserved (must be 0) ¹
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
11:9	Ignored
31:12	Physical address of the 4-KByte page referenced by this entry

NOTES:

1. See Section 5.1.4 for how to determine whether the PAT is supported.

5.4 PAE PAGING

A logical processor uses PAE paging if $CR0.PG = 1$, $CR4.PAE = 1$, and $IA32\_EFER.LME = 0$. PAE paging translates 32-bit linear addresses to 52-bit physical addresses.¹ Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 32 bits; at most 4 GBytes of linear-address space may be accessed at any given time.

With PAE paging, a logical processor maintains a set of four (4) PDPTe registers, which are loaded from an address in CR3. Linear address are translated using 4 hierarchies of in-memory paging structures, each located using one of the PDPTe registers. (This is different from the other paging modes, in which there is one hierarchy referenced by CR3.)

Section 5.4.1 discusses the PDPTe registers. Section 5.4.2 describes linear-address translation with PAE paging.

5.4.1 PDPTe Registers

When PAE paging is used, CR3 references the base of a 32-Byte **page-directory-pointer table**. Table 5-7 illustrates how CR3 is used with PAE paging.

Table 5-7. Use of CR3 with PAE Paging

Bit Position(s)	Contents
4:0	Ignored
31:5	Physical address of the 32-Byte aligned page-directory-pointer table used for linear-address translation
63:32	Ignored (these bits exist only on processors supporting the Intel-64 architecture)

The page-directory-pointer-table comprises four (4) 64-bit entries called PDPTes. Each PDPTe controls access to a 1-GByte region of the linear-address space. Corresponding to the PDPTes, the logical processor maintains a set of four (4) internal, non-architectural PDPTe registers, called PDPTe0, PDPTe1, PDPTe2, and PDPTe3. The logical processor loads these registers from the PDPTes in memory as part of certain operations:

- If PAE paging would be in use following an execution of MOV to CR0 or MOV to CR4 (see Section 5.1.1) and the instruction is modifying any of CR0.CD, CR0.NW, CR0.PG, CR4.PAE, CR4.PGE, CR4.PSE, or CR4.SMEP; then the PDPTes are loaded from the address in CR3.
- If MOV to CR3 is executed while the logical processor is using PAE paging, the PDPTes are loaded from the address being loaded into CR3.
- If PAE paging is in use and a task switch changes the value of CR3, the PDPTes are loaded from the address in the new CR3 value.
- Certain VMX transitions load the PDPTe registers. See Section 5.11.1.

Table 5-8 gives the format of a PDPTe. If any of the PDPTes sets both the P flag (bit 0) and any reserved bit, the MOV to CR instruction causes a general-protection exception (#GP(0)) and the PDPTes are not loaded.² As shown in Table 5-8, bits 2:1, 8:5, and 63:MAXPHYADDR are reserved in the PDPTes.

1. If $MAXPHYADDR < 52$, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by PAE paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 5.1.4 for how to determine MAXPHYADDR.

2. On some processors, reserved bits are checked even in PDPTes in which the P flag (bit 0) is 0.

Table 5-8. Format of a PAE Page-Directory-Pointer-Table Entry (PDPTE)

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
2:1	Reserved (must be 0)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 5.9)
8:5	Reserved (must be 0)
11:9	Ignored
M-1:12	Physical address of 4-KByte aligned page directory referenced by this entry ¹
63:M	Reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

5.4.2 Linear-Address Translation with PAE Paging

PAE paging may map linear addresses to either 4-KByte pages or 2-MByte pages. Figure 5-5 illustrates the translation process when it produces a 4-KByte page; Figure 5-6 covers the case of a 2-MByte page. The following items describe the PAE paging process in more detail as well as how the page size is determined:

- Bits 31:30 of the linear address select a PDPTE register (see Section 5.4.1); this is PDPTE_{*i*}, where *i* is the value of bits 31:30.¹ Because a PDPTE register is identified using bits 31:30 of the linear address, it controls access to a 1-GByte region of the linear-address space. If the P flag (bit 0) of PDPTE_{*i*} is 0, the processor ignores bits 63:1, and there is no mapping for the 1-GByte region controlled by PDPTE_{*i*}. A reference using a linear address in this region causes a page-fault exception (see Section 5.7).
- If the P flag of PDPTE_{*i*} is 1, 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of PDPTE_{*i*} (see Table 5-8 in Section 5.4.1). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
 - Bits 51:12 are from PDPTE_{*i*}.
 - Bits 11:3 are bits 29:21 of the linear address.
 - Bits 2:0 are 0.

Because a PDE is identified using bits 31:21 of the linear address, it controls access to a 2-Mbyte region of the linear-address space. Use of the PDE depends on its PS flag (bit 7):

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table 5-9). The final physical address is computed as follows:
 - Bits 51:21 are from the PDE.
 - Bits 20:0 are from the original linear address.
- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 5-10). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:

1. With PAE paging, the processor does not use CR3 when translating a linear address (as it does in the other paging modes). It does not access the PDPTes in the page-directory-pointer table during linear-address translation.

- Bits 51:12 are from the PDE.
- Bits 11:3 are bits 20:12 of the linear address.
- Bits 2:0 are 0.
- Because a PTE is identified using bits 31:12 of the linear address, every PTE maps a 4-KByte page (see Table 5-11). The final physical address is computed as follows:
 - Bits 51:12 are from the PTE.
 - Bits 11:0 are from the original linear address.

If the P flag (bit 0) of a PDE or a PTE is 0 or if a PDE or a PTE sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 5.7).

The following bits are reserved with PAE paging:

- If the P flag (bit 0) of a PDE or a PTE is 1, bits 62:M are reserved, where M is MAXPHYADDR (unless EPT is enabled, in which case M is the value enumerated by CPUID.80000008H:EAX[7:0]). See Section 5.1.4.
- If the P flag and the PS flag (bit 7) of a PDE are both 1, bits 20:13 are reserved.
- If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.
- If the PAT is not supported:¹
 - If the P flag of a PTE is 1, bit 7 is reserved.
 - If the P flag and the PS flag of a PDE are both 1, bit 12 is reserved.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 5.6.

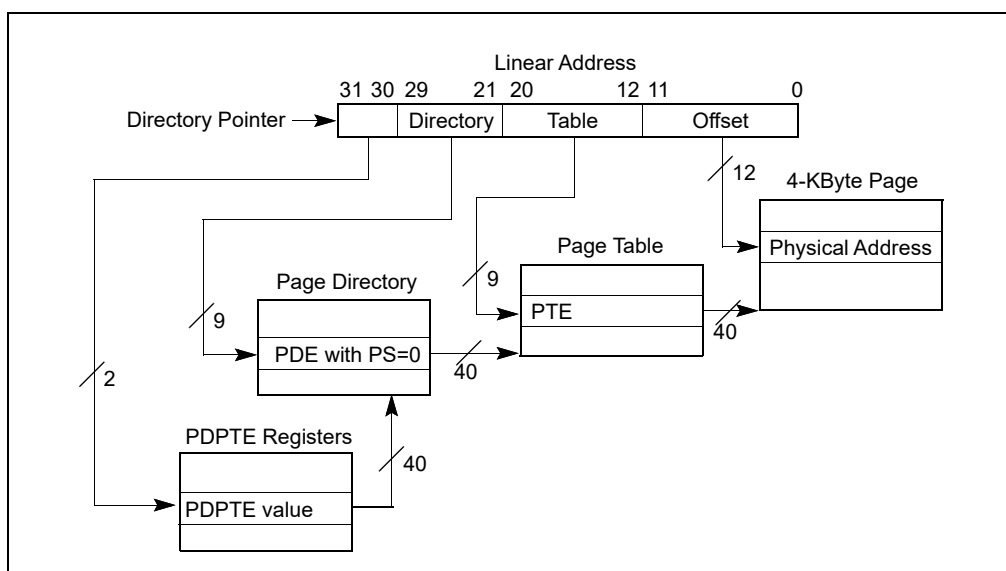


Figure 5-5. Linear-Address Translation to a 4-KByte Page using PAE Paging

1. See Section 5.1.4 for how to determine whether the PAT is supported.

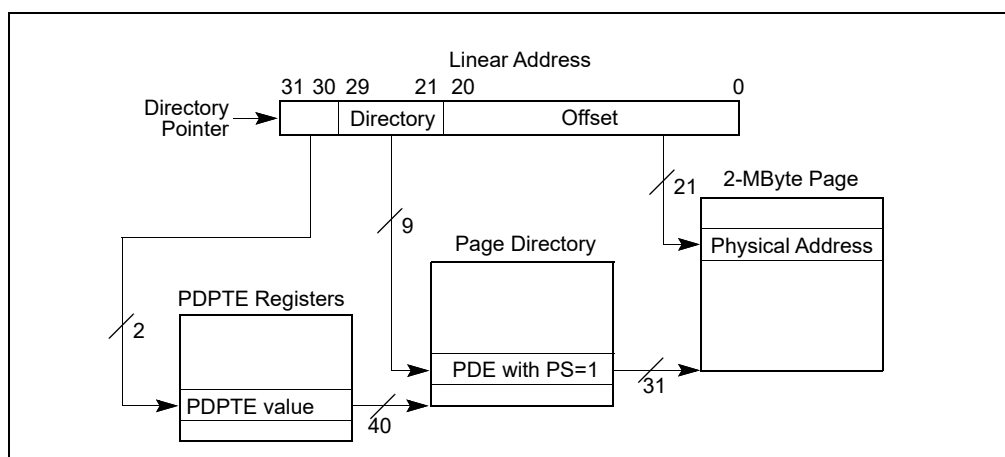


Figure 5-6. Linear-Address Translation to a 2-MByte Page using PAE Paging

Table 5-9. Format of a PAE Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 5.9)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 5.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 5-10)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
11:9	Ignored
12 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 5.9.2); otherwise, reserved (must be 0) ¹
20:13	Reserved (must be 0)
M-1:21	Physical address of the 2-MByte page referenced by this entry ²
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. See Section 5.1.4 for how to determine whether the PAT is supported.

2. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-10. Format of a PAE Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 5.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 5.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 5-9)
11:8	Ignored
M-1:12	Physical address of 4-KByte aligned page table referenced by this entry ¹
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 5.8)

Table 5-11. Format of a PAE Page-Table Entry that Maps a 4-KByte Page (Contd.)

Bit Position(s)	Contents
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 5.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2); otherwise, reserved (must be 0) ¹
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
11:9	Ignored
M-1:12	Physical address of the 4-KByte page referenced by this entry ²
62:M	Reserved (must be 0)
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. See Section 5.1.4 for how to determine whether the PAT is supported.
2. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EX[7:0]. See Section 5.1.4.

Figure 5-7 gives a summary of the formats of CR3 and the paging-structure entries with PAE paging. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

6	6	6	6	5	5	5	5	5	5	5	5	5			M ¹	M-1				3	3	3	3	2	2	2	2	2	2	2	2	1	0	9	8	7	6	5	4	3	2	1	0	
Ignored ²															Address of page-directory-pointer table															Ignored					CR3									
Reserved ³										Address of page directory															Ign.		Rsvd.		P C D	P W T	R s v d	1	PDPTE: present											
Ignored																														0		PDPTE: not present												
X D 4	Reserved										Address of 2MB page frame										Reserved			P A T	Ign.		G	1	D	A	P C D	P W T	U / S	R / W	1	PDE: 2MB page								
X D	Reserved										Address of page table												Ign.			0	I g n	A	P C D	P W T	U / S	R / W	1	PDE: page table										
Ignored																														0		PDE: not present												
X D	Reserved										Address of 4KB page frame												Ign.		G	P A T	D	A	P C D	P W T	U / S	R / W	1	PTE: 4KB page										
Ignored																														0		PTE: not present												

Figure 5-7. Formats of CR3 and Paging-Structure Entries with PAE Paging

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.
2. CR3 has 64 bits only on processors supporting the Intel-64 architecture. These bits are ignored with PAE paging.
3. Reserved fields must be 0.
4. If IA32_EFER.NXE = 0 and the P flag of a PDE or a PTE is 1, the XD flag (bit 63) is reserved.

5.5 4-LEVEL PAGING AND 5-LEVEL PAGING

Because the operation of 4-level paging and 5-level paging is very similar, they are described together in this section. The following items highlight the distinctions between the two paging modes:

- A logical processor uses 4-level paging if CR0.PG = 1, CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 0. 4-level paging translates 48-bit linear addresses to 52-bit physical addresses.¹ Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time.
- A logical processor uses 5-level paging if CR0.PG = 1, CR4.PAE = 1, IA32_EFER.LME = 1, and CR4.LA57 = 1. 5-level paging translates 57-bit linear addresses to 52-bit physical addresses. Thus, 5-level paging supports a linear-address space sufficient to access the entire physical-address space.

5.5.1 Ordinary Paging and HLAT Paging

There are two forms of 4-level paging and 5-level paging that differ principally with regard to how linear-address translation identifies the first paging structure.

The normal form is called **ordinary paging**, and it uses CR3 to locate the first paging structure, similar to what is done for 32-bit paging. Section 5.5.2 provides details of this use of CR3.

An alternative form of paging may be used with the VMX feature called hypervisor-managed linear-address translation (HLAT). Called **HLAT paging**, this form is used only in VMX non-root operation and only if the “enable HLAT” VM-execution control is 1.² HLAT paging locates the first paging structure using a VM-execution control field in the VMCS called the **HLAT pointer (HLATP)**. Section 5.5.3 provides details.

Whether HLAT paging is used to translate a specific linear address depends on the address and on the value of a VM-execution control field in the VMCS called the **HLAT prefix size**:

- If the HLAT prefix size is zero, every linear address is translated using HLAT paging.
- If the HLAT prefix size is not zero, a linear address is translated using HLAT paging if bit 63 of the address is 1.³ The address is translated using ordinary paging if bit 63 of the address is 0.

In some cases, HLAT paging may specify that a translation of a linear address must be restarted. When this occurs, the linear address is then translated using ordinary paging (starting with a paging structure identified using CR3). The situations leading to this restart are detailed in Section 5.5.4, and additional details of the restart process are given in Section 5.5.5.

-
1. If MAXPHYADDR < 52, bits in the range 51:MAXPHYADDR will be 0 in any physical address used by 4-level paging or 5-level paging. (The corresponding bits are reserved in the paging-structure entries.) See Section 5.1.4 for how to determine MAXPHYADDR.
 2. HLAT paging is used only with 4-level paging and 5-level paging. It is never used with 32-bit paging or PAE paging, regardless of the value of the “enable HLAT” VM-execution control.
 3. This behavior applies if the CPU enumerates a maximum HLAT prefix size of 1 in IA32_VMX_EPT_VPID_CAP[53:48] (see Appendix A.10). Behavior when a different value is enumerated is not currently defined.

5.5.2 Use of CR3 with Ordinary 4-Level Paging and 5-Level Paging

Ordinary 4-level paging and 5-level paging each translate linear addresses using a hierarchy of in-memory paging structures located using the contents of CR3, which is used to locate the first paging structure. For 4-level paging, this is the PML4 table, and for 5-level paging it is the PML5 table. Use of CR3 with 4-level paging and 5-level paging depends on whether process-context identifiers (PCIDs) have been enabled by setting CR4.PCIDE:

- Table 5-12 illustrates how CR3 is used with 4-level paging and 5-level paging if CR4.PCIDE = 0.

Table 5-12. Use of CR3 with 4-Level Paging and 5-level Paging and CR4.PCIDE = 0

Bit Position(s)	Contents
2:0	Ignored
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table or PML5 table during linear-address translation (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table or PML5 table during linear-address translation (see Section 5.9.2)
11:5	Ignored
M-1:12	Physical address of the 4-KByte aligned PML4 table or PML5 table used for linear-address translation ¹
60:M	Reserved (must be 0)
61	Enables LAM57 for user pointers; Section 4.4. ²
62	Enables LAM48 for user pointers; ignored if bit 61 is set. ²
63	Reserved (must be 0) ³

NOTES:

- If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.
- LAM is not a paging feature.
- See Section 5.10.4.1 for use of bit 63 of the source operand of the MOV to CR3 instruction.

- Table 5-13 illustrates how CR3 is used with 4-level paging and 5-level paging if CR4.PCIDE = 1.

Table 5-13. Use of CR3 with 4-Level Paging and 5-Level Paging and CR4.PCIDE = 1

Bit Position(s)	Contents
11:0	PCID (see Section 5.10.1) ¹
M-1:12	Physical address of the 4-KByte aligned PML4 table used for linear-address translation ²
60:M	Reserved (must be 0)
61	Enables LAM57 for user pointers; Section 4.4. ³
62	Enables LAM48 for user pointers; ignored if bit 61 is set. ³
63	Reserved (must be 0) ⁴

NOTES:

- Section 5.9.2 explains how the processor determines the memory type used to access the PML4 table during linear-address translation with CR4.PCIDE = 1.

2. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.
3. LAM is not a paging feature.
4. See Section 5.10.4.1 for use of bit 63 of the source operand of the MOV to CR3 instruction.

After software modifies the value of CR4.PCIDE, the logical processor immediately begins using CR3 as specified for the new value. For example, if software changes CR4.PCIDE from 1 to 0, the current PCID immediately changes from CR3[11:0] to 000H (see also Section 5.10.4.1). In addition, the logical processor subsequently determines the memory type used to access the PML4 table using CR3.PWT and CR3.PCD, which had been bits 4:3 of the PCID.

5.5.3 Use of HLATP with HLAT 4-Level Paging and 5-Level Paging

With HLAT paging, 4-level paging and 5-level paging each translate linear addresses using a hierarchy of in-memory paging structures located using the value of HLATP (a VM-execution control field in the VMCS), which is used to locate the first paging structure. For 4-level paging, this is the PML4 table, and for 5-level paging it is the PML5 table.

HLATP has the same format as that given for CR3 in Table 5-12, with the exception that bits 2:0 and bits 11:5 are reserved and must be zero (these bits are checked by VM entry). HLATP does not contain a PCID value. HLAT paging with CR4.PCIDE = 1 uses the PCID value in CR3[11:0].

5.5.4 Linear-Address Translation with 4-Level Paging and 5-Level Paging

4-level paging and 5-level paging may map linear addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages.¹ Figure 5-8 illustrates the translation process for 4-level paging when it produces a 4-KByte page; Figure 5-9 covers the case of a 2-MByte page, and Figure 5-10 the case of a 1-GByte page. (The process for 5-level paging is similar.)

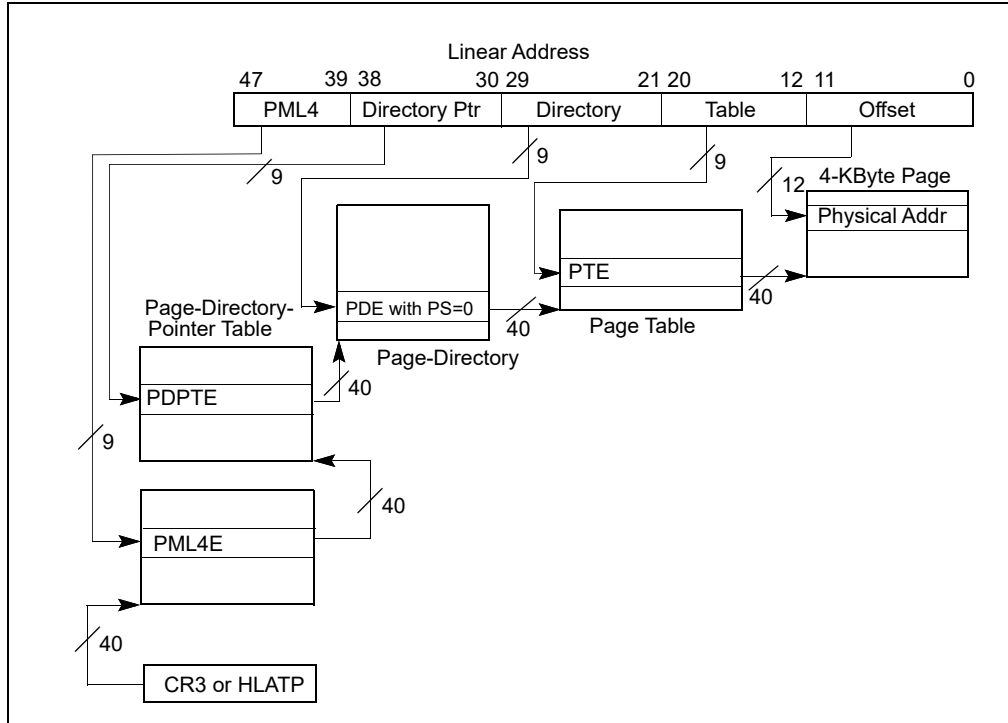


Figure 5-8. Linear-Address Translation to a 4-KByte Page Using 4-Level Paging

1. Not all processors support 1-GByte pages; see Section 5.1.4.

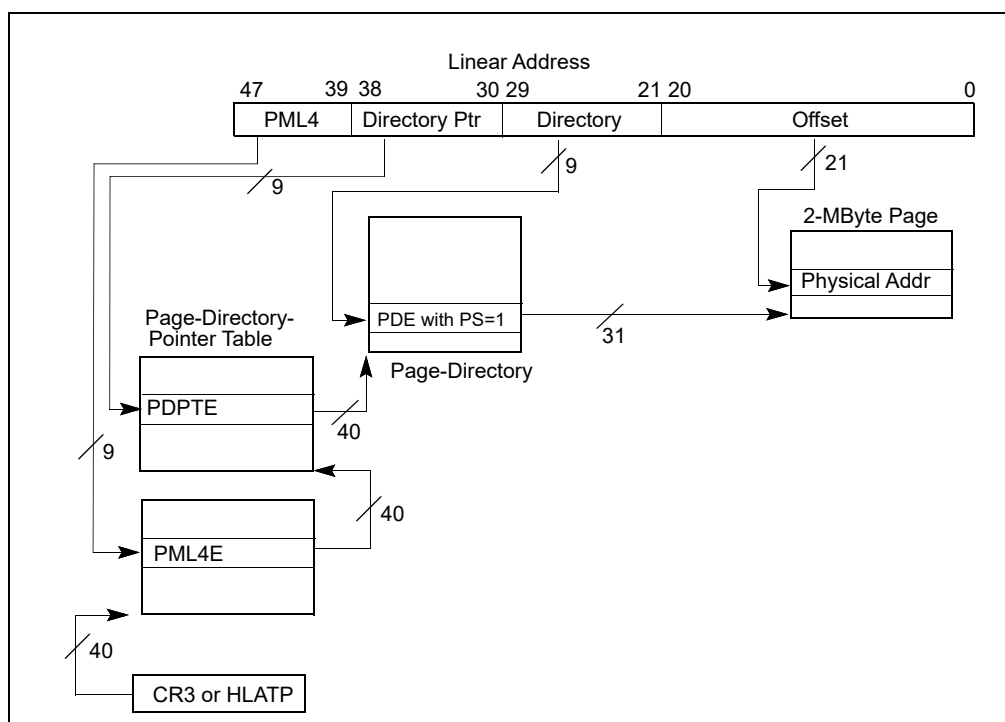


Figure 5-9. Linear-Address Translation to a 2-MByte Page using 4-Level Paging

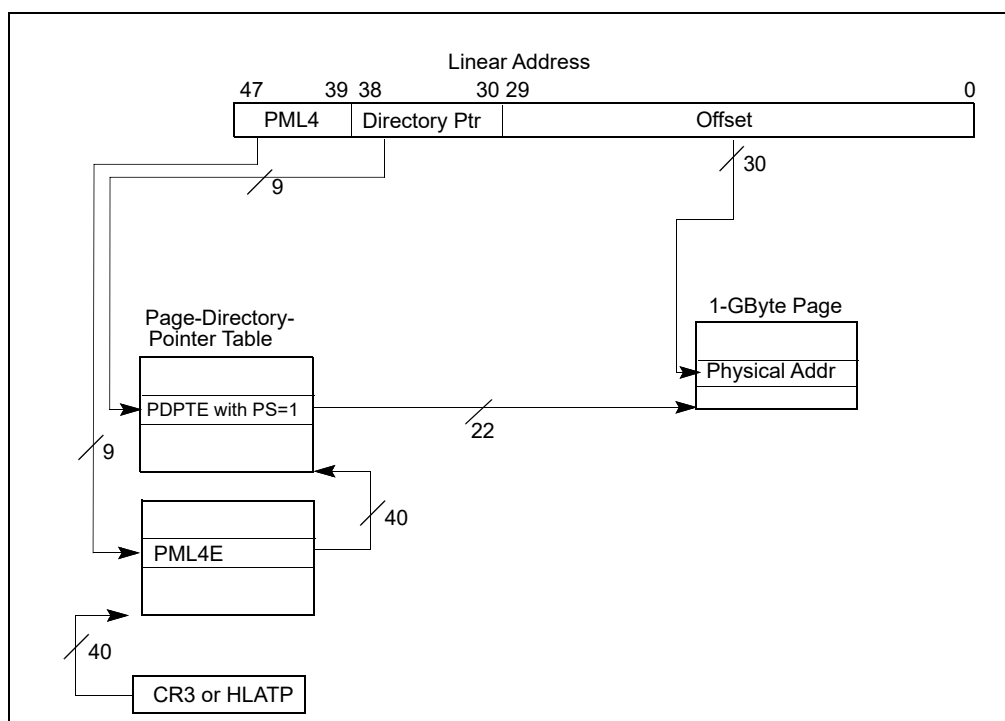


Figure 5-10. Linear-Address Translation to a 1-GBByte Page using 4-Level Paging

4-level paging and 5-level paging associate with each linear address a **protection key**. Section 5.6 explains how the processor uses the protection key in its determination of the access rights of each linear address.

The remainder of this section describes the translation process used by 4-level paging and 5-level paging in more detail, as well as how the page size and protection key are determined. Because the process used by the two paging modes is similar, they are described together, with any differences identified, in the following items:

- With 5-level paging, a 4-KByte naturally aligned PML5 table is located at the physical address specified in bits 51:12 of CR3 (see Table 5-12). (4-level paging does not use a PML5 table and omits this step.) A PML5 table comprises 512 64-bit entries (PML5Es). A PML5E is selected using the physical address defined as follows:
 - Bits 51:12 are from CR3 or HLATP.
 - Bits 11:3 are bits 56:48 of the linear address.
 - Bits 2:0 are all 0.

Because a PML5E is identified using bits 56:48 of the linear address, it controls access to a 256-TByte region of the linear-address space.

With HLAT paging, if bit 11 of the PML5E is 1, translation is restarted with ordinary paging with a maximum page size of 256-TBytes (see Section 5.5.5). Otherwise, the translation process continues as described in the next item.

- A 4-KByte naturally aligned PML4 table is located at the physical address specified in bits 51:12 of CR3 (for 4-level paging; see Table 5-12) or in bits 51:12 of the PML5E (for 5-level paging; see Table 5-14). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows:
 - Bits 51:12 are from CR3 or the HLATP (for 4-level paging) or in bits 51:12 of the PML5E (for 5-level paging).
 - Bits 11:3 are bits 47:39 of the linear address.
 - Bits 2:0 are all 0.

Because a PML4E is identified using bits 47:39 of the linear address, it controls access to a 512-GByte region of the linear-address space.

With HLAT paging, if bit 11 of the PML4E is 1, translation is restarted with ordinary paging with a maximum page size of 512-GBytes (see Section 5.5.5). Otherwise, the translation process continues as described in the next item.

- A 4-KByte naturally aligned page-directory-pointer table is located at the physical address specified in bits 51:12 of the PML4E (see Table 5-15). A page-directory-pointer table comprises 512 64-bit entries (PDPTEs). A PDPTE is selected using the physical address defined as follows:
 - Bits 51:12 are from the PML4E.
 - Bits 11:3 are bits 38:30 of the linear address.
 - Bits 2:0 are all 0.

Because a PDPTE is identified using bits 47:30 of the linear address, it controls access to a 1-GByte region of the linear-address space.

With HLAT paging, if bit 11 of the PDPTE is 1, translation is restarted with ordinary paging with a maximum page size of 1-GByte (see Section 5.5.5). Otherwise, the translation process continues as described below.

Use of the PDPTE depends on its PS flag (bit 7):¹

- If the PDPTE's PS flag is 1, the PDPTE maps a 1-GByte page (see Table 5-16). The final physical address is computed as follows:
 - Bits 51:30 are from the PDPTE.
 - Bits 29:0 are from the original linear address.

The linear address's protection key is the value of bits 62:59 of the PDPTE (see Section 5.6.2).

1. The PS flag of a PDPTE is reserved and must be 0 (if the P flag is 1) if 1-GByte pages are not supported. See Section 5.1.4 for how to determine whether 1-GByte pages are supported.

- If the PDPTE's PS flag is 0, a 4-KByte naturally aligned page directory is located at the physical address specified in bits 51:12 of the PDPTE (see Table 5-17). A page directory comprises 512 64-bit entries (PDEs). A PDE is selected using the physical address defined as follows:
 - Bits 51:12 are from the PDPTE.
 - Bits 11:3 are bits 29:21 of the linear address.
 - Bits 2:0 are all 0.

Because a PDE is identified using bits 47:21 of the linear address, it controls access to a 2-MByte region of the linear-address space.

With HLAT paging, if bit 11 of the PDE is 1, translation is restarted with ordinary paging with a maximum page size of 2-MBytes (see Section 5.5.5). Otherwise, the translation process continues as described below.

Use of the PDE depends on its PS flag:

- If the PDE's PS flag is 1, the PDE maps a 2-MByte page (see Table 5-18). The final physical address is computed as follows:
 - Bits 51:21 are from the PDE.
 - Bits 20:0 are from the original linear address.

The linear address's protection key is the value of bits 62:59 of the PDE (see Section 5.6.2).

- If the PDE's PS flag is 0, a 4-KByte naturally aligned page table is located at the physical address specified in bits 51:12 of the PDE (see Table 5-19). A page table comprises 512 64-bit entries (PTEs). A PTE is selected using the physical address defined as follows:

- Bits 51:12 are from the PDE.
- Bits 11:3 are bits 20:12 of the linear address.
- Bits 2:0 are all 0.

- Because a PTE is identified using bits 47:12 of the linear address, every PTE maps a 4-KByte page (see Table 5-20).

With HLAT paging, if bit 11 of the PTE is 1, translation is restarted with ordinary paging with a maximum page size of 4-KBytes (see Section 5.5.5). Otherwise, the final physical address is computed as follows:

- Bits 51:12 are from the PTE.
- Bits 11:0 are from the original linear address.

The linear address's protection key is the value of bits 62:59 of the PTE (see Section 5.6.2).

If a paging-structure entry's P flag (bit 0) is 0 or if the entry sets any reserved bit, the entry is used neither to reference another paging-structure entry nor to map a page. There is no translation for a linear address whose translation would use such a paging-structure entry; a reference to such a linear address causes a page-fault exception (see Section 5.7).

The following bits in a paging-structure entry are reserved with 4-level paging and 5-level paging (assuming that the entry's P flag is 1):

- Bits 51:M are reserved in every paging-structure entry, where M is MAXPHYADDR (unless EPT is enabled, in which case it is the value enumerated by CPUID.80000008H:EAX[7:0]). See Section 5.1.4.
- The PS flag is reserved in a PML5E or a PML4E.
- If 1-GByte pages are not supported, the PS flag is reserved in a PDPTE.¹
- If the PS flag in a PDPTE is 1, bits 29:13 of the entry are reserved.
- If the PS flag in a PDE is 1, bits 20:13 of the entry are reserved.
- If IA32_EFER.NXE = 0, the XD flag (bit 63) is reserved in every paging-structure entry.

A reference using a linear address that is successfully translated to a physical address is performed only if allowed by the access rights of the translation; see Section 5.6.

1. See Section 5.1.4 for how to determine whether 1-GByte pages are supported.

Figure 5-11 gives a summary of the formats of CR3 and the 4-level and 5-level paging-structure entries. For the paging structure entries, it identifies separately the format of entries that map pages, those that reference other paging structures, and those that do neither because they are “not present”; bit 0 (P) and bit 7 (PS) are highlighted because they determine how a paging-structure entry is used.

Table 5-14. Format of a PML5 Entry (PML5E) that References a PML4 Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a PML4 table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 256-TByte region controlled by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 256-TByte region controlled by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 5.8)
6	Ignored
7 (PS)	Reserved (must be 0)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of 4-KByte aligned PML4 table referenced by this entry ¹
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 256-TByte region controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-15. Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page-directory-pointer table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 512-GByte region controlled by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 512-GByte region controlled by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page-directory-pointer table referenced by this entry (see Section 5.9.2)

Table 5-15. Format of a PML4 Entry (PML4E) that References a Page-Directory-Pointer Table (Contd.)

Bit Position(s)	Contents
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 5.8)
6	Ignored
7 (PS)	Reserved (must be 0)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of 4-KByte aligned page-directory-pointer table referenced by this entry ¹
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 512-GByte region controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-16. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 1-GByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 5.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page directory; see Table 5-17)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
12 (PAT)	Indirectly determines the memory type used to access the 1-GByte page referenced by this entry (see Section 5.9.2) ¹
29:13	Reserved (must be 0)

Table 5-16. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GBYTE Page (Contd.)

Bit Position(s)	Contents
M-1:30	Physical address of the 1-GBYTE page referenced by this entry ²
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 5.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GBYTE page controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. The PAT is supported on all processors that support 4-level paging.
2. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-17. Format of a Page-Directory-Pointer-Table Entry (PDPTE) that References a Page Directory

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page directory
1 (R/W)	Read/write; if 0, writes may not be allowed to the 1-GBYTE region controlled by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 1-GBYTE region controlled by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page directory referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 5.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 1-GBYTE page; see Table 5-16)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of 4-KByte aligned page directory referenced by this entry ¹
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 1-GBYTE region controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-18. Format of a Page-Directory Entry that Maps a 2-MByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 2-MByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 5.8)
7 (PS)	Page size; must be 1 (otherwise, this entry references a page table; see Table 5-19)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
12 (PAT)	Indirectly determines the memory type used to access the 2-MByte page referenced by this entry (see Section 5.9.2)
20:13	Reserved (must be 0)
M-1:21	Physical address of the 2-MByte page referenced by this entry ¹
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 5.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte page controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-19. Format of a Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 2-MByte region controlled by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 2-MByte region controlled by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 5.8)
6	Ignored
7 (PS)	Page size; must be 0 (otherwise, this entry maps a 2-MByte page; see Table 5-18)
10:8	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of 4-KByte aligned page table referenced by this entry ¹
51:M	Reserved (must be 0)
62:52	Ignored
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 2-MByte region controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

Table 5-20. Format of a Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 5.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 5.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 5.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 5.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 5.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 5.10); ignored otherwise
10:9	Ignored
11 (R)	For ordinary paging, ignored; for HLAT paging, restart (if 1, linear-address translation is restarted with ordinary paging)
M-1:12	Physical address of the 4-KByte page referenced by this entry ¹
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1 or CR4.PKS = 1, this may control the page's access rights (see Section 5.6.2); otherwise, it is ignored and not used to control access rights.
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 5.6); otherwise, reserved (must be 0)

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by CPUID.80000008H:EAX[7:0]. See Section 5.1.4.

63	62	61	60	59	58	57	56	55	54	53	52	51		M ¹	M-1			32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved ²														Address of PML4 table (4-level paging) or PML5 table (5-level paging)														Ignored				P C D	P W T	Ign.		CR3															
X D 3	Ignored												Rsvd.		Address of PML4 table														R 4	Ign.	R s v d	I g n	A	P C D	P W T	U / S	R / W	1	PML5E: present												
Ignored																											0	PML5E: not present																							
X D	Ignored												Rsvd.		Address of page-directory-pointer table														R	Ign.	R s v d	I g n	A	P C D	P W T	U / S	R / W	1	PML4E: present												
Ignored																											0	PML4E: not present																							
X D	Prot. Key ⁵	Ignored										Rsvd.		Address of 1GB page frame						Reserved								P A T	R	Ign.	G	1	D	A	P C D	P W T	U / S	R / W	1	PDPT: 1GB page											
X D	Ignored												Rsvd.		Address of page directory														R	Ign.	0	I g n	A	P C D	P W T	U / S	R / W	1	PDPT: page directory												
Ignored																											0	PDPT: not present																							
X D	Prot. Key	Ignored										Rsvd.		Address of 2MB page frame								Reserved						P A T	R	Ign.	G	1	D	A	P C D	P W T	U / S	R / W	1	PDE: 2MB page											
X D	Ignored												Rsvd.		Address of page table														R	Ign.	0	I g n	A	P C D	P W T	U / S	R / W	1	PDE: page table												
Ignored																											0	PDE: not present																							
X D	Prot. Key	Ignored										Rsvd.		Address of 4KB page frame														R	Ign.	G	P A T	D	A	P C D	P W T	U / S	R / W	1	PTE: 4KB page												
Ignored																											0	PTE: not present																							

Figure 5-11. Formats of CR3 and Paging-Structure Entries with 4-Level Paging and 5-Level Paging

NOTES:

1. If EPT is not enabled, M is an abbreviation for MAXPHYADDR. If EPT is enabled, M represents the value returned by `CPUID.80000008H:EAX[7:0]`. See Section 5.1.4.
2. Reserved fields must be 0. On processors that support linear-address masking (see Section 4.4), bits 62:61 configure that feature and may be set to 1. Because linear-address masking is not a paging feature, those bits are not illustrated here.
3. If `IA32_EFER.NXE = 0` and the P flag of a paging-structure entry is 1, the XD flag (bit 63) is reserved.
4. Bit 11 is R (restart) only for HLAT paging; it is ignored for ordinary paging.
5. The protection key is used only if software has enabled the appropriate feature; see Section 5.6.2. It is ignored otherwise.

5.5.5 Restart of HLAT Paging

As noted in Section 5.5.1, HLAT paging may specify that a translation of a linear address must be restarted. Specifically, this occurs when HLAT paging encounters a paging-structure entry that sets bit 11 (see Section 5.5.4).

When this occurs, translation of the linear address is restarted using ordinary paging (starting with a paging structure identified using CR3). The restarted translation proceeds just as if the HLAT feature were not enabled. The entire linear address is translated again, including those portions that had been used by HLAT paging prior to the restart.

The process of restarting HLAT paging (using ordinary paging) always specifies a maximum page size to be used when a resulting translation is cached in the TLBs. This maximum page size depends on the level of the paging-structure entry that restarts the translation by setting bit 11; details are given in Section 5.5.4. The page size of the translation produced by the restarted process is never greater than this maximum page size. See Section 5.10.2.2 for more discussion.

5.6 ACCESS RIGHTS

There is a translation for a linear address if the processes described in Section 5.3, Section 5.4.2, and Section 5.5 (depending upon the paging mode) completes and produces a physical address. Whether an access is permitted by a translation is determined by the access rights specified by the paging-structure entries controlling the translation;¹ paging-mode modifiers in CR0, CR4, and the IA32_EFER MSR; EFLAGS.AC; and the mode of the access.

Section 5.6.1 describes how the processor determines the access rights for each linear address. Section 5.6.2 provides additional information about how protection keys contribute to access-rights determination. (They do so only with 4-level paging and 5-level paging, and only if CR4.PKE = 1 or CR4.PKS = 1.)

NOTE

If HLAT paging is restarted, permissions are determined only by the access rights specified by the paging-structure entries that the subsequent ordinary paging used to translate the linear address. The access rights specified by the entries used earlier by HLAT paging do not apply.

5.6.1 Determination of Access Rights

Every access to a linear address is either a **supervisor-mode access** or a **user-mode access**. For all instruction fetches and most data accesses, this distinction is determined by the current privilege level (CPL): accesses made while CPL < 3 are supervisor-mode accesses, while accesses made while CPL = 3 are user-mode accesses.

Some operations implicitly access system data structures with linear addresses; the resulting accesses to those data structures are supervisor-mode accesses regardless of CPL. Examples of such accesses include the following: accesses to the global descriptor table (GDT) or local descriptor table (LDT) to load a segment descriptor; accesses to the interrupt descriptor table (IDT) when delivering an interrupt or exception; and accesses to the task-state segment (TSS) as part of a task switch or change of CPL. All these accesses are called **implicit supervisor-mode accesses** regardless of CPL. Other accesses made while CPL < 3 are called **explicit supervisor-mode accesses**.

Access rights are also controlled by the **mode** of a linear address as specified by the paging-structure entries controlling the translation of the linear address. If the U/S flag (bit 2) is 0 in at least one of the paging-structure entries, the address is a **supervisor-mode address**. Otherwise, the address is a **user-mode address**.

When the shadow-stack feature of control-flow enforcement technology (CET) is enabled, certain accesses to linear addresses are considered **shadow-stack accesses** (see Section 18.2, “Shadow Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). Like ordinary data accesses, each shadow-stack access is defined as being either a user access or a supervisor access. In general, a shadow-stack access is a user access if CPL = 3 and a supervisor access if CPL < 3. The WRUSS instruction is an exception; although it can be executed only if CPL = 0, the processor treats its shadow-stack accesses as user accesses.

1. With PAE paging, the PDPTes do not determine access rights.

Shadow-stack accesses are allowed only to **shadow-stack addresses**. A linear address is a shadow-stack address if the following are true of the translation of the linear address: (1) the R/W flag (bit 1) is 0 and the dirty flag (bit 6) is 1 in the paging-structure entry that maps the page containing the linear address; and (2) the R/W flag is 1 in every other paging-structure entry controlling the translation of the linear address.

The following items detail how paging determines access rights (only the items noted explicitly apply to shadow-stack accesses):

NOTE

Many of the items below refer to an address with a protection key for which read (or write) access is permitted. Section 5.6.2 provides details on when a protection key will permit (or not permit) a data access (read or write) to a linear address using that protection key.

- For supervisor-mode accesses:
 - Data may be read (implicitly or explicitly) from any supervisor-mode address with a protection key for which read access is permitted.
 - Data reads from user-mode pages.
Access rights depend on the value of CR4.SMAP:
 - If CR4.SMAP = 0, data may be read from any user-mode address with a protection key for which read access is permitted.
 - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
 - If EFLAGS.AC = 1 and the access is explicit, data may be read from any user-mode address with a protection key for which read access is permitted.
 - If EFLAGS.AC = 0 or the access is implicit, data may not be read from any user-mode address.
 - Data writes to supervisor-mode addresses.
Access rights depend on the value of CR0.WP:
 - If CR0.WP = 0, data may be written to any supervisor-mode address with a protection key for which write access is permitted.
 - If CR0.WP = 1, data may be written to any supervisor-mode address with a translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any supervisor-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
 - Data writes to user-mode addresses.
Access rights depend on the value of CR0.WP:
 - If CR0.WP = 0, access rights depend on the value of CR4.SMAP:
 - If CR4.SMAP = 0, data may be written to any user-mode address with a protection key for which write access is permitted.
 - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:
 - If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a protection key for which write access is permitted.
 - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
 - If CR0.WP = 1, access rights depend on the value of CR4.SMAP:
 - If CR4.SMAP = 0, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
 - If CR4.SMAP = 1, access rights depend on the value of EFLAGS.AC and whether the access is implicit or explicit:

- If EFLAGS.AC = 1 and the access is explicit, data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted; data may not be written to any user-mode address with a translation for which the R/W flag is 0 in any paging-structure entry controlling the translation.
 - If EFLAGS.AC = 0 or the access is implicit, data may not be written to any user-mode address.
- Instruction fetches from supervisor-mode addresses.
 - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any supervisor-mode address.
 - For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any supervisor-mode address with a translation for which the XD flag (bit 63) is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any supervisor-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
- Instruction fetches from user-mode addresses.
Access rights depend on the values of CR4.SMEP:
 - If CR4.SMEP = 0, access rights depend on the paging mode and the value of IA32_EFER.NXE:
 - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
 - For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation; instructions may not be fetched from any user-mode address with a translation for which the XD flag is 1 in any paging-structure entry controlling the translation.
 - If CR4.SMEP = 1, instructions may not be fetched from any user-mode address.
- Supervisor-mode shadow-stack accesses are allowed only to supervisor-mode shadow-stack addresses (see above).
- For user-mode accesses:
 - Data reads.
Access rights depend on the mode of the linear address:
 - Data may be read from any user-mode address with a protection key for which read access is permitted.
 - Data may not be read from any supervisor-mode address.
 - Data writes.
Access rights depend on the mode of the linear address:
 - Data may be written to any user-mode address with a translation for which the R/W flag is 1 in every paging-structure entry controlling the translation and with a protection key for which write access is permitted.
 - Data may not be written to any supervisor-mode address.
 - Instruction fetches.
Access rights depend on the mode of the linear address, the paging mode, and the value of IA32_EFER.NXE:
 - For 32-bit paging or if IA32_EFER.NXE = 0, instructions may be fetched from any user-mode address.
 - For other paging modes with IA32_EFER.NXE = 1, instructions may be fetched from any user-mode address with a translation for which the XD flag is 0 in every paging-structure entry controlling the translation.
 - Instructions may not be fetched from any supervisor-mode address.
- User-mode shadow-stack accesses made outside enclave mode are allowed only to user-mode shadow-stack addresses (see above). User-mode shadow-stack accesses made in enclave mode are treated like ordinary data accesses (see above).

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 5.10). These structures may include information about access rights. The processor may enforce access rights based on the TLBs and paging-structure caches instead of on the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change access rights, the processor might not use that change for a subsequent access to an affected linear address (see Section 5.10.4.3). See Section 5.10.4.2 for how software can ensure that the processor uses the modified access rights.

5.6.2 Protection Keys

4-level paging and 5-level paging associate a 4-bit protection key with each linear address (the protection key located in bits 62:59 of the paging-structure entry that mapped the page containing the linear address; see Section 5.5). Two protection key features control accesses to linear addresses based on their protection keys:

- If CR4.PKE = 1, the PKRU register determines, for each protection key, whether user-mode addresses with that protection key may be read or written.
- If CR4.PKS = 1, the IA32_PKRS MSR (MSR index 6E1H) determines, for each protection key, whether supervisor-mode addresses with that protection key may be read or written.

32-bit paging and PAE paging do not associate linear addresses with protection keys. For the purposes of Section 5.6.1, reads and writes are implicitly permitted for all protection keys with either of those paging modes.

The PKRU register (protection-key rights for user pages) is a 32-bit register with the following format: for each i ($0 \leq i \leq 15$), PKRU[2*i*] is the **access-disable bit** for protection key i (AD*i*); PKRU[2*i*+1] is the **write-disable bit** for protection key i (WD*i*). The IA32_PKRS MSR has the same format (bits 63:32 of the MSR are reserved and must be zero).

Software can use the RDPKRU and WRPKRU instructions with ECX = 0 to read and write PKRU. In addition, the PKRU register is XSAVE-managed state and can thus be read and written by instructions in the XSAVE feature set. See Chapter 13, “Managing State Using the XSAVE Feature Set,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about the XSAVE feature set.

Software can use the RDMSR and WRMSR instructions to read and write the IA32_PKRS MSR. Writes to the IA32_PKRS MSR using WRMSR are not serializing. The IA32_PKRS MSR is not XSAVE-managed.

How a linear address’s protection key controls access to the address depends on the mode of the linear address:

- A linear address’s protection key controls only data accesses to the address. It does not in any way affect instructions fetches from the address.
- If CR4.PKE = 0, the protection key of a user-mode address does not control data accesses to the address (for the purposes of Section 5.6.1, reads and writes of user-mode addresses are implicitly permitted for all protection keys).

If CR4.PKE = 1, use of the protection key i of a user-mode address depends on the value of the PKRU register:

- If AD*i* = 1, no data accesses are permitted.
- If WD*i* = 1, permission may be denied to certain data write accesses:
 - User-mode write accesses are not permitted.
 - Supervisor-mode write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, WD*i* does not affect supervisor-mode write accesses to user-mode addresses with protection key i .)

- If CR4.PKS = 0, the protection key of a supervisor-mode address does not control data accesses to the address (for the purposes of Section 5.6.1, reads and writes of supervisor-mode addresses are implicitly permitted for all protection keys).

If CR4.PKS = 1, use of the protection key i of a supervisor-mode address depends on the value of the IA32_PKRS MSR:

- If AD*i* = 1, no data accesses are permitted.
- If WD*i* = 1, write accesses are not permitted if CR0.WP = 1. (If CR0.WP = 0, IA32_PKRS.WD*i* does not affect write accesses to supervisor-mode addresses with protection key i .)

Protection keys apply to shadow-stack accesses just as they do to ordinary data accesses.

5.7 PAGE-FAULT EXCEPTIONS

Accesses using linear addresses may cause **page-fault exceptions** (#PF; exception 14). An access to a linear address may cause a page-fault exception for either of two reasons: (1) there is no translation for the linear address; or (2) there is a translation for the linear address, but its access rights do not permit the access.

As noted in Section 5.3, Section 5.4.2, and Section 5.5, there is no translation for a linear address if the translation process for that address would use a paging-structure entry in which the P flag (bit 0) is 0 or one that sets a reserved bit.¹ If there is a translation for a linear address, its access rights are determined as specified in Section 5.6.

When Intel® Software Guard Extensions (Intel® SGX) are enabled, the processor may deliver exception 14 for reasons unrelated to paging. See Section 37.3, “Access-control Requirements,” and Section 37.20, “Enclave Page Cache Map (EPCM),” in Chapter 37, “Enclave Access Control and Data Structures.” Such an exception is called an **SGX-induced page fault**. The processor uses the error code to distinguish SGX-induced page faults from ordinary page faults.

When a page fault occurs, the processor loads the CR2 register with the linear address that generated the exception. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata. If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the address are cleared.

Figure 5-12 illustrates the error code that the processor provides on delivery of a page-fault exception. The following items explain how the bits in the error code describe the nature of the page-fault exception:

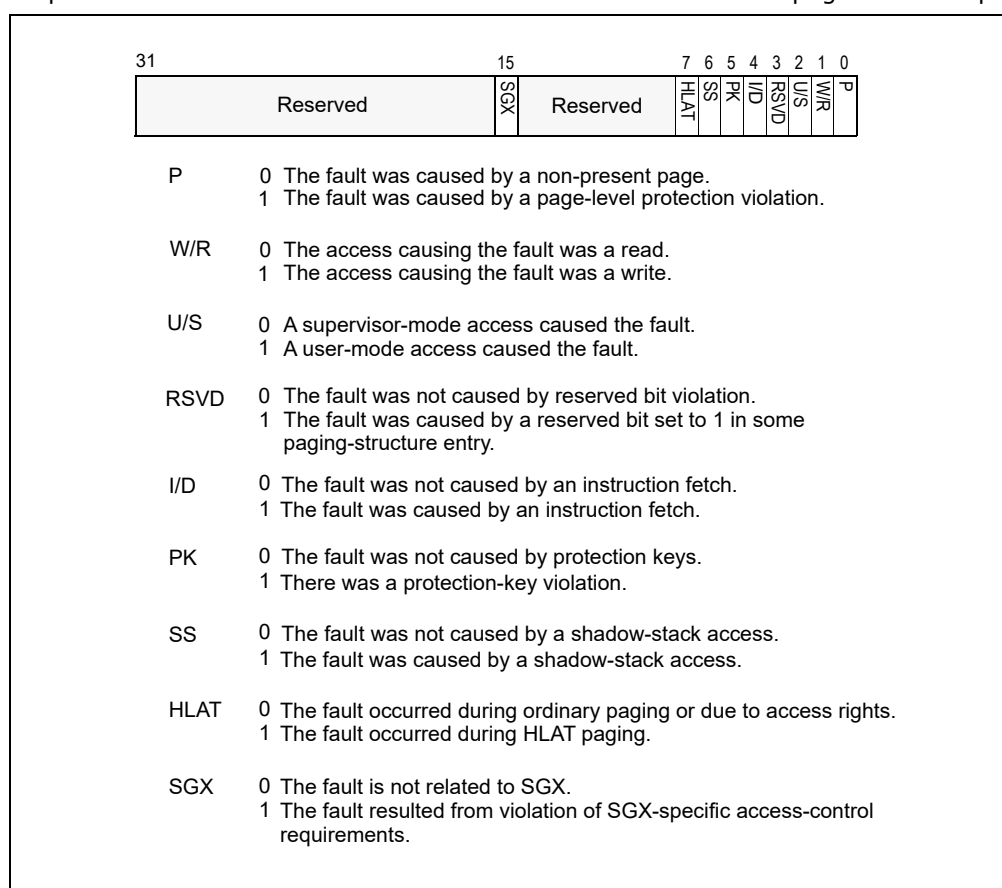


Figure 5-12. Page-Fault Error Code

1. If HLAT paging encounters a paging-structure entry that sets a reserved bit, there is no translation even if the bit 11 of the entry indicates a restart. In this case, there is a page fault and the translation is not restarted.

- P flag (bit 0).
This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
- W/R (bit 1).
If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- U/S (bit 2).
If a user-mode access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging. User-mode and supervisor-mode accesses are defined in Section 5.6.
- RSVD flag (bit 3).
This flag is 1 if there is no translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address. (Because reserved bits are not checked in a paging-structure entry whose P flag is 0, bit 3 of the error code can be set only if bit 0 is also set.¹)

Bits reserved in the paging-structure entries are reserved for future functionality. Software developers should be aware that such bits may be used in the future and that a paging-structure entry that causes a page-fault exception on one processor might not do so in the future.
- I/D flag (bit 4).
This flag is 1 if (1) the access causing the page-fault exception was an instruction fetch; and (2) either (a) CR4.SMEP = 1; or (b) both (i) CR4.PAE = 1 (either PAE paging, 4-level paging, or 5-level paging is in use); and (ii) IA32_EFER.NXE = 1. Otherwise, the flag is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- PK flag (bit 5).
This flag is 1 only for data accesses and only with 4-level paging and 5-level paging. In these cases, the setting depends on the mode of the address being accessed:
 - For accesses to supervisor-mode addresses, the flag is set if (1) CR4.PKS = 1; (2) the linear address has protection key i ; and (3) the IA32_PKRS MSR (see Section 5.6.2) is such that either (a) $AD_i = 1$; or (b) the following all hold: (i) $WD_i = 1$; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access. (Note that this flag may be set on page faults due to user-mode accesses to supervisor-mode addresses.)
 - For accesses to user-mode addresses, the flag is set if (1) CR4.PKE = 1; (2) the linear address has protection key i ; and (3) the PKRU register (see Section 5.6.2) is such that either (a) $AD_i = 1$; or (b) the following all hold: (i) $WD_i = 1$; (ii) the access is a write access; and (iii) either CR0.WP = 1 or the access causing the page-fault exception was a user-mode access.
- SS (bit 6).
If the access causing the page-fault exception was a shadow-stack access (including shadow-stack accesses in enclave mode), this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- HLAT (bit 7).
This flag is 1 if there is no translation for the linear address using HLAT paging because, in one of the paging-structure entries used to translate that address, either the P flag was 0 or a reserved bit was set. An error code will set this flag only if it clears bit 0 or sets bit 3. This flag will not be set by a page fault resulting from a violation of access rights, nor for one encountered during ordinary paging, including the case in which there has been a restart of HLAT paging.
- SGX flag (bit 15).
This flag is 1 if the exception is unrelated to paging and resulted from violation of SGX-specific access-control requirements. Because such a violation can occur only if there is no ordinary page fault, this flag is set only if the P flag (bit 0) is 1 and the RSVD flag (bit 3) and the PK flag (bit 5) are both 0.

Page-fault exceptions occur only due to an attempt to use a linear address. Failures to load the PDPTE registers with PAE paging (see Section 5.4.1) cause general-protection exceptions (#GP(0)) and not page-fault exceptions.

1. Some past processors had errata for some page faults that occur when there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address. Due to these errata, some such page faults produced error codes that cleared bit 0 (P flag) and set bit 3 (RSVD flag).

5.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.¹ For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

The previous two paragraphs apply also to HLAT paging. If HLAT paging encounters a paging-structure entry that sets bit 11, indicating a restart, the processor will set the accessed flag in that entry; it will not set the dirty flag because, if an entry indicates a restart, it does identify the final physical address for the linear address being translated.

NOTE

If software on one logical processor writes to a page while software on another logical processor concurrently clears the R/W flag in the paging-structure entry that maps the page, execution on some processors may result in the entry's dirty flag being set (due to the write on the first logical processor) and the entry's R/W flag being clear (due to the update to the entry on the second logical processor). This will never occur on a processor that supports control-flow enforcement technology (CET). Specifically, a processor that supports CET will never set the dirty flag in a paging-structure entry in which the R/W flag is clear.

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are "sticky," meaning that, once set, the processor does not clear them; only software can clear them.

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 5.10). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the processor might not set the corresponding bit in memory on a subsequent access using an affected linear address (see Section 5.10.4.3). See Section 5.10.4.2 for how software can ensure that these bits are updated as desired.

NOTE

The accesses used by the processor to set these flags may or may not be exposed to the processor's self-modifying code detection logic. If the processor is executing code from the same memory area that is being used for the paging structures, the setting of these flags may or may not result in an immediate change to the executing code stream.

5.9 PAGING AND MEMORY TYPING

The **memory type** of a memory access refers to the type of caching used for that access. Chapter 13, "Memory Cache Control," provides many details regarding memory typing in the Intel-64 and IA-32 architectures. This section describes how paging contributes to the determination of memory typing.

The way in which paging contributes to memory typing depends on whether the processor supports the **Page Attribute Table (PAT)**; see Section 13.12).² Section 5.9.1 and Section 5.9.2 explain how paging contributes to memory typing depending on whether the PAT is supported.

1. With PAE paging, the PDPTes are not used during linear-address translation but only to load the PDPTe registers for some executions of the MOV CR instruction (see Section 5.4.1). For this reason, the PDPTes do not contain accessed flags with PAE paging.
2. The PAT is supported on Pentium III and more recent processor families. See Section 5.1.4 for how to determine whether the PAT is supported.

5.9.1 Paging and Memory Typing When the PAT is Not Supported (Pentium Pro and Pentium II Processors)

NOTE

The PAT is supported on all processors that support 4-level paging or 5-level paging. Thus, this section applies only to 32-bit paging and PAE paging.

If the PAT is not supported, paging contributes to memory typing in conjunction with the memory-type range registers (MTRRs) as specified in Table 13-6 in Section 13.5.2.1.

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a PCD value and a PWT value. The latter two values are determined as follows:

- For an access to a PDE with 32-bit paging, the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, the PCD and PWT values come from the relevant PDPTTE register.
- For an access to a PTE, the PCD and PWT values come from the relevant PDE.
- For an access to the physical address that is the translation of a linear address, the PCD and PWT values come from the relevant PTE (if the translation uses a 4-KByte page) or the relevant PDE (otherwise).
- With PAE paging, the UC memory type is used when loading the PDPTTEs (see Section 5.4.1).

5.9.2 Paging and Memory Typing When the PAT is Supported (Pentium III and More Recent Processor Families)

If the PAT is supported, paging contributes to memory typing in conjunction with the PAT and the memory-type range registers (MTRRs) as specified in Table 13-7 in Section 13.5.2.2.

The PAT is a 64-bit MSR (IA32_PAT; MSR index 277H) comprising eight (8) 8-bit entries (entry i comprises bits $8i+7:8i$ of the MSR).

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a memory type selected from the PAT. Table 13-11 in Section 13.12.3 specifies how a memory type is selected from the PAT. Specifically, it comes from entry i of the PAT, where i is defined as follows:

- For an access to an entry in a paging structure whose address is in CR3 (e.g., the PML4 table with 4-level paging):
 - For 4-level paging or 5-level paging with CR4.PCIDE = 1, $i = 0$.
 - Otherwise, $i = 2*PCD+PWT$, where the PCD and PWT values come from CR3.
- For an access to a PDE with PAE paging, $i = 2*PCD+PWT$, where the PCD and PWT values come from the relevant PDPTTE register.
- For an access to a paging-structure entry X whose address is in another paging-structure entry Y, $i = 2*PCD+PWT$, where the PCD and PWT values come from Y.
- For an access to the physical address that is the translation of a linear address, $i = 4*PAT+2*PCD+PWT$, where the PAT, PCD, and PWT values come from the relevant PTE (if the translation uses a 4-KByte page), the relevant PDE (if the translation uses a 2-MByte page or a 4-MByte page), or the relevant PDPTTE (if the translation uses a 1-GByte page).
- With PAE paging, the WB memory type is used when loading the PDPTTEs (see Section 5.4.1).¹

1. Some older IA-32 processors used the UC memory type when loading the PDPTTEs. Some processors may use the UC memory type if CRO.CD = 1 or if the MTRRs are disabled. These behaviors are model-specific and not architectural.

5.9.3 Caching Paging-Related Information about Memory Typing

A processor may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 5.10). These structures may include information about memory typing. The processor may use memory-typing information from the TLBs and paging-structure caches instead of from the paging structures in memory.

This fact implies that, if software modifies a paging-structure entry to change the memory-typing bits, the processor might not use that change for a subsequent translation using that entry or for access to an affected linear address. See Section 5.10.4.2 for how software can ensure that the processor uses the modified memory typing.

5.10 CACHING TRANSLATION INFORMATION

The Intel-64 and IA-32 architectures may accelerate the address-translation process by caching data from the paging structures on the processor. Because the processor does not ensure that the data that it caches are always consistent with the structures in memory, it is important for software developers to understand how and when the processor may cache such data. They should also understand what actions software can take to remove cached data that may be inconsistent and when it should do so. This section provides software developers information about the relevant processor operation.

Section 5.10.1 introduces process-context identifiers (PCIDs), which a logical processor may use to distinguish information cached for different linear-address spaces. Section 5.10.2 and Section 5.10.3 describe how the processor may cache information in translation lookaside buffers (TLBs) and paging-structure caches, respectively. Section 5.10.4 explains how software can remove inconsistent cached information by invalidating portions of the TLBs and paging-structure caches. Section 5.10.5 describes special considerations for multiprocessor systems.

5.10.1 Process-Context Identifiers (PCIDs)

Process-context identifiers (**PCIDs**) are a facility by which a logical processor may cache information for multiple linear-address spaces. The processor may retain cached information when software switches to a different linear-address space with a different PCID (e.g., by loading CR3; see Section 5.10.4.1 for details).

A PCID is a 12-bit identifier. Non-zero PCIDs are enabled by setting the PCIDE flag (bit 17) of CR4. If CR4.PCIDE = 0, the current PCID is always 000H; otherwise, the current PCID is the value of bits 11:0 of CR3.¹ Not all processors allow CR4.PCIDE to be set to 1; see Section 5.1.4 for how to determine whether this is allowed.

The processor ensures that CR4.PCIDE can be 1 only in IA-32e mode (thus, 32-bit paging and PAE paging use only PCID 000H). In addition, software can change CR4.PCIDE from 0 to 1 only if CR3[11:0] = 000H. These requirements are enforced by the following limitations on the MOV CR instruction:

- MOV to CR4 causes a general-protection exception (#GP) if it would change CR4.PCIDE from 0 to 1 and either IA32_EFER.LMA = 0 or CR3[11:0] ≠ 000H.
- MOV to CR0 causes a general-protection exception if it would clear CR0.PG to 0 while CR4.PCIDE = 1.

When a logical processor creates entries in the TLBs (Section 5.10.2) and paging-structure caches (Section 5.10.3), it associates those entries with the current PCID. When using entries in the TLBs and paging-structure caches to translate a linear address, a logical processor uses only those entries associated with the current PCID (see Section 5.10.2.4 for an exception).

If CR4.PCIDE = 0, a logical processor does not cache information for any PCID other than 000H. This is because (1) if CR4.PCIDE = 0, the logical processor will associate any newly cached information with the current PCID, 000H; and (2) if MOV to CR4 clears CR4.PCIDE, all cached information is invalidated (see Section 5.10.4.1).

1. Note that, while HLAT paging (Section 5.5.3) does not use CR3 to locate the first paging structure, it does use the PCID in CR3[11:0] when CR4.PCIDE = 1.

NOTE

In revisions of this manual that were produced when no processors allowed CR4.PCIDE to be set to 1, Section 5.10, “Caching Translation Information,” discussed the caching of translation information without any reference to PCIDs. While the section now refers to PCIDs in its specification of this caching, this documentation change is not intended to imply any change to the behavior of processors that do not allow CR4.PCIDE to be set to 1.

5.10.2 Translation Lookaside Buffers (TLBs)

A processor may cache information about the translation of linear addresses in translation lookaside buffers (TLBs). In general, TLBs contain entries that map page numbers to page frames; these terms are defined in Section 5.10.2.1. Section 5.10.2.2 describes how information may be cached in TLBs, and Section 5.10.2.3 gives details of TLB usage. Section 5.10.2.4 explains the global-page feature, which allows software to indicate that certain translations should receive special treatment when cached in the TLBs.

5.10.2.1 Page Numbers, Page Frames, and Page Offsets

Section 5.3, Section 5.4.2, and Section 5.5 give details of how the different paging modes translate linear addresses to physical addresses. Specifically, the upper bits of a linear address (called the **page number**) determine the upper bits of the physical address (called the **page frame**); the lower bits of the linear address (called the **page offset**) determine the lower bits of the physical address. The boundary between the page number and the page offset is determined by the **page size**. Specifically:

- 32-bit paging:
 - If the translation does not use a PTE (because CR4.PSE = 1 and the PS flag is 1 in the PDE used), the page size is 4 MBytes and the page number comprises bits 31:22 of the linear address.
 - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- PAE paging:
 - If the translation does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 31:21 of the linear address.
 - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 31:12 of the linear address.
- 4-level paging and 5-level paging:
 - If the translation does not use a PDE (because the PS flag is 1 in the PDPTE used), the page size is 1 GByte and the page number comprises bits 47:30 of the linear address.
 - If the translation does use a PDE but does not use a PTE (because the PS flag is 1 in the PDE used), the page size is 2 MBytes and the page number comprises bits 47:21 of the linear address.
 - If the translation does use a PTE, the page size is 4 KBytes and the page number comprises bits 47:12 of the linear address.
 - The page size identified by the preceding items may be reduced if there has been a restart of HLAT paging (see Section 5.5.5). Restart of HLAT paging always specifies a maximum page size; this page size is determined by the level of the paging-structure entry that caused the restart. The page size used by the translation is the minimum of the maximum page size specified by the restart and the page size determined by the restarted translation (as specified by the previous items).

For example, suppose that HLAT paging encounters a PDE that sets bit 11, indicating a restart. As a result, the restart uses a maximum page size of 2 MBytes. Suppose that the restarted translation encounters a PDPTE that sets bit 7, indicating a 1-GByte page. In this case, the translation produced will have a page size of 2 MBytes (the smaller of the two sizes).

5.10.2.2 Caching Translations in TLBs

The processor may accelerate the paging process by caching individual translations in **translation lookaside buffers (TLBs)**. Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number (see Section 5.6):
 - The logical-AND of the R/W flags.
 - The logical-AND of the U/S flags.
 - The logical-OR of the XD flags (necessary only if IA32_EFER.NXE = 1).
 - The protection key (only with 4-level paging and 5-level paging).
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):
 - The dirty flag (see Section 5.8).
 - The memory type (see Section 5.9).

(TLB entries may contain other information as well. A processor may implement multiple TLBs, and some of these may be for special purposes, e.g., only for instruction fetches. Such special-purpose TLBs may not contain some of this information if it is not necessary. For example, a TLB used only for instruction fetches need not contain information about the R/W and dirty flags.)

As noted in Section 5.10.1, any TLB entries created by a logical processor are associated with the current PCID.

Processors need not implement any TLBs. Processors that do implement TLBs may invalidate any TLB entry at any time. Software should not rely on the existence of TLBs or on the retention of TLB entries.

5.10.2.3 Details of TLB Use

Because the TLBs cache entries only for linear addresses with translations, there can be a TLB entry for a page number only if the P flag is 1 and the reserved bits are 0 in each of the paging-structure entries used to translate that page number. In addition, the processor does not cache a translation for a page number unless the accessed flag is 1 in each of the paging-structure entries used during translation; before caching a translation, the processor sets any of these accessed flags that is not already 1.

Subject to the limitations given in the previous paragraph, the processor may cache a translation for any linear address, even if that address is not used to access memory. For example, the processor may cache translations required for prefetches and for accesses that result from speculative execution that would never actually occur in the executed code path.

If the page number of a linear address corresponds to a TLB entry associated with the current PCID, the processor may use that TLB entry to determine the page frame, access rights, and other attributes for accesses to that linear address. In this case, the processor may not actually consult the paging structures in memory. The processor may retain a TLB entry unmodified even if software subsequently modifies the relevant paging-structure entries in memory. See Section 5.10.4.2 for how software can ensure that the processor uses the modified paging-structure entries.

If the paging structures specify a translation using a page larger than 4 KBytes, some processors may cache multiple smaller-page TLB entries for that translation. Each such TLB entry would be associated with a page number corresponding to the smaller page size (e.g., bits 47:12 of a linear address with 4-level paging), even though part of that page number (e.g., bits 20:12) is part of the offset with respect to the page specified by the paging structures. The upper bits of the physical address in such a TLB entry are derived from the physical address in the PDE used to create the translation, while the lower bits come from the linear address of the access for which the translation is created. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. For example, an execution of INVLPG for a linear address on such a page invalidates any and all smaller-page TLB entries for the translation of any linear address on that page.

If software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes, the TLBs may subsequently contain multiple translations for the address range (one for each page size). A reference to a linear address in the address range may use any of these translations. Which translation is used may vary from one execution to another, and the choice may be implementation-specific.

5.10.2.4 Global Pages

The Intel-64 and IA-32 architectures also allow for **global pages** when the PGE flag (bit 7) is 1 in CR4. If the G flag (bit 8) is 1 in a paging-structure entry that maps a page (either a PTE or a paging-structure entry in which the PS flag is 1), any TLB entry cached for a linear address using that paging-structure entry is considered to be **global**. Because the G flag is used only in paging-structure entries that map a page, and because information from such entries is not cached in the paging-structure caches, the global-page feature does not affect the behavior of the paging-structure caches.

A logical processor may use a global TLB entry to translate a linear address, even if the TLB entry is associated with a PCID different from the current PCID.

5.10.3 Paging-Structure Caches

In addition to the TLBs, a processor may cache other information about the paging structures in memory.

5.10.3.1 Caches for Paging Structures

A processor may support any or all of the following paging-structure caches:

- **PML5E cache** (5-level paging only). Each PML5E-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 56:48 have that value. The entry contains information from the PML5E used to translate such linear addresses:
 - The physical address from the PML5E (the address of the PML4 table).
 - The value of the R/W flag of the PML5E.
 - The value of the U/S flag of the PML5E.
 - The value of the XD flag of the PML5E.
 - The values of the PCD and PWT flags of the PML5E.

The following items detail how a processor may use the PML5E cache:

- If the processor has a PML5E-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E in memory).
- The processor does not create a PML5E-cache entry unless the P flag is 1 and all reserved bits are 0 in the PML5E in memory.
- The processor does not create a PML5E-cache entry unless the accessed flag is 1 in the PML5E in memory; before caching a translation, the processor sets the accessed flag if it is not already 1.
- The processor may create a PML5E-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced PML4 table).
- If the processor creates a PML5E-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E in memory.
- **PML4E cache** (4-level paging and 5-level paging only). The use of the PML4E cache depends on the paging mode:
 - For 4-level paging, each PML4E-cache entry is referenced by a 9-bit value and is used for linear addresses for which bits 47:39 have that value.
 - For 5-level paging, each PML4E-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 56:39 have that value.

A PML4E-cache entry contains information from the PML5E and PML4E used to translate the relevant linear addresses (for 4-level paging, the PML5E does not apply):

- The physical address from the PML4E (the address of the page-directory-pointer table).
- The logical-AND of the R/W flags in the PML5E and the PML4E.
- The logical-AND of the U/S flags in the PML5E and the PML4E.
- The logical-OR of the XD flags in the PML5E and the PML4E.
- The values of the PCD and PWT flags of the PML4E.

The following items detail how a processor may use the PML4E cache:

- If the processor has a PML4E-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E and PML4E in memory).
- The processor does not create a PML4E-cache entry unless the P flags are 1 and all reserved bits are 0 in the PML5E and the PML4E in memory.
- The processor does not create a PML4E-cache entry unless the accessed flags are 1 in the PML5E and the PML4E in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
- The processor may create a PML4E-cache entry even if there are no translations for any linear address that might use that entry (e.g., because the P flags are 0 in all entries in the referenced page-directory-pointer table).
- If the processor creates a PML4E-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML4E in memory.

- **PDPTE cache** (4-level paging and 5-level paging only).¹ The use of the PML4E cache depends on the paging mode:

- For 4-level paging, each PDPTE-cache entry is referenced by an 18-bit value and is used for linear addresses for which bits 47:30 have that value.
- For 5-level paging, each PDPTE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 56:30 have that value.

A PDPTE-cache entry contains information from the PML5E, PML4E, PDPTE used to translate the relevant linear addresses (for 4-level paging, the PML5E does not apply):

- The physical address from the PDPTE (the address of the page directory). (No PDPTE-cache entry is created for a PDPTE that maps a 1-GByte page.)
- The logical-AND of the R/W flags in the PML5E, PML4E, and PDPTE.
- The logical-AND of the U/S flags in the PML5E, PML4E, and PDPTE.
- The logical-OR of the XD flags in the PML5E, PML4E, and PDPTE.
- The values of the PCD and PWT flags of the PDPTE.

The following items detail how a processor may use the PDPTE cache:

- If the processor has a PDPTE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E, PML4E, and PDPTE in memory).
- The processor does not create a PDPTE-cache entry unless the P flags are 1, the PS flags are 0, and the reserved bits are 0 in the PML5E, PML4E, and PDPTE in memory.
- The processor does not create a PDPTE-cache entry unless the accessed flags are 1 in the PML5E, PML4E, and PDPTE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
- The processor may create a PDPTE-cache entry even if there are no translations for any linear address that might use that entry.
- If the processor creates a PDPTE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E, PML4E, or PDPTE in memory.

1. With PAE paging, the PDPTEs are stored in internal, non-architectural registers. The operation of these registers is described in Section 5.4.1 and differs from that described here.

- **PDE cache.** The use of the PDE cache depends on the paging mode:
 - For 32-bit paging, each PDE-cache entry is referenced by a 10-bit value and is used for linear addresses for which bits 31:22 have that value.
 - For PAE paging, each PDE-cache entry is referenced by an 11-bit value and is used for linear addresses for which bits 31:21 have that value.
 - For 4-level paging, each PDE-cache entry is referenced by a 27-bit value and is used for linear addresses for which bits 47:21 have that value.
 - For 5-level paging, each PDE-cache entry is referenced by a 36-bit value and is used for linear addresses for which bits 56:21 have that value.

A PDE-cache entry contains information from the PML5E, PML4E, PDPTE, and PDE used to translate the relevant linear addresses (for 32-bit paging and PAE paging, only the PDE applies; for 4-level paging, the PML5E does not apply):

- The physical address from the PDE (the address of the page table). (No PDE-cache entry is created for a PDE that maps a page.)
- The logical-AND of the R/W flags in the PML5E, PML4E, PDPTE, and PDE.
- The logical-AND of the U/S flags in the PML5E, PML4E, PDPTE, and PDE.
- The logical-OR of the XD flags in the PML5E, PML4E, PDPTE, and PDE.
- The values of the PCD and PWT flags of the PDE.

The following items detail how a processor may use the PDE cache (references below to PML5Es, PML4Es, and PDPTEs apply only to 4-level paging and to 5-level paging, as appropriate):

- If the processor has a PDE-cache entry for a linear address, it may use that entry when translating the linear address (instead of the PML5E, PML4E, PDPTE, and PDE in memory).
- The processor does not create a PDE-cache entry unless the P flags are 1, the PS flags are 0, and the reserved bits are 0 in the PML5E, PML4E, PDPTE, and PDE in memory.
- The processor does not create a PDE-cache entry unless the accessed flag is 1 in the PML5E, PML4E, PDPTE, and PDE in memory; before caching a translation, the processor sets any accessed flags that are not already 1.
- The processor may create a PDE-cache entry even if there are no translations for any linear address that might use that entry.
- If the processor creates a PDE-cache entry, the processor may retain it unmodified even if software subsequently modifies the corresponding PML5E, PML4E, PDPTE, or PDE in memory.

Information from a paging-structure entry can be included in entries in the paging-structure caches for other paging-structure entries referenced by the original entry. For example, if the R/W flag is 0 in a PML4E, then the R/W flag will be 0 in any PDPTE-cache entry for a PDPTE from the page-directory-pointer table referenced by that PML4E. This is because the R/W flag of each such PDPTE-cache entry is the logical-AND of the R/W flags in the appropriate PML4E and PDPTE.

On processors that support HLAT paging (see Section 5.5.1), each entry in a paging-structure cache indicates whether the entry was cached during ordinary paging or HLAT paging. When the processor commences linear-address translation using ordinary paging (respectively, HLAT paging), it will use only entries that indicate that they were cached during ordinary paging (respectively, HLAT paging).

Entries that were cached during HLAT paging also include the restart flag (bit 11) of the original paging-structure entry. When the processor commences HLAT paging using such an entry, it immediately restarts (using ordinary paging) if this cached restart flag is 1.

The paging-structure caches contain information only from paging-structure entries that reference other paging structures (and not those that map pages). Because the G flag is not used in such paging-structure entries, the global-page feature does not affect the behavior of the paging-structure caches.

The processor may create entries in paging-structure caches for translations required for prefetches and for accesses that are a result of speculative execution that would never actually occur in the executed code path.

As noted in Section 5.10.1, any entries created in paging-structure caches by a logical processor are associated with the current PCID.

A processor may or may not implement any of the paging-structure caches. Software should rely on neither their presence nor their absence. The processor may invalidate entries in these caches at any time. Because the processor may create the cache entries at the time of translation and not update them following subsequent modifications to the paging structures in memory, software should take care to invalidate the cache entries appropriately when causing such modifications. The invalidation of TLBs and the paging-structure caches is described in Section 5.10.4.

5.10.3.2 Using the Paging-Structure Caches to Translate Linear Addresses

When a linear address is accessed, the processor uses a procedure such as the following to determine the physical address to which it translates and whether the access should be allowed:

- If the processor finds a TLB entry that is for the page number of the linear address and that is associated with the current PCID (or which is global), it may use the physical address, access rights, and other attributes from that entry.
- If the processor does not find a relevant TLB entry, it may use the upper bits of the linear address to select an entry from the PDE cache that is associated with the current PCID (Section 5.10.3.1 indicates which bits are used in each paging mode). It can then use that entry to complete the translation process (locating a PTE, etc.) as if it had traversed the PDE (and, for 4-level paging and 5-level paging, the PDPTE, PML4E, and PML5E, as appropriate) corresponding to the PDE-cache entry.
- The following items apply when 4-level paging or 5-level paging is used:
 - If the processor does not find a relevant TLB entry or PDE-cache entry, it may use the upper bits of the linear address (for 4-level paging, bits 47:30; for 5-level paging, bits 56:30) to select an entry from the PDPTE cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDE, etc.) as if it had traversed the PDPTE, the PML4E, and (for 5-level paging) the PML5E corresponding to the PDPTE-cache entry.
 - If the processor does not find a relevant TLB entry, PDE-cache entry, or PDPTE-cache entry, it may use the upper bits of the linear address (for 4-level paging, bits 47:39; for 5-level paging, bits 56:39) to select an entry from the PML4E cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PDPTE, etc.) as if it had traversed the corresponding PML4E.
 - With 5-level paging, if the processor does not find a relevant TLB entry, PDE-cache entry, PDPTE-cache entry, or PML4E-cache entry, it may use bits 56:48 of the linear address to select an entry from the PML5E cache that is associated with the current PCID. It can then use that entry to complete the translation process (locating a PML4E, etc.) as if it had traversed the corresponding PML5E.

(Any of the above steps would be skipped if the processor does not support the cache in question.)

If the processor does not find a TLB or paging-structure-cache entry for the linear address, it uses the linear address to traverse the entire paging-structure hierarchy, as described in Section 5.3, Section 5.4.2, and Section 5.5.

5.10.3.3 Multiple Cached Entries for a Single Paging-Structure Entry

The paging-structure caches and TLBs may contain multiple entries associated with a single PCID and with information derived from a single paging-structure entry. The following items give some examples for 4-level paging:

- Suppose that two PML4Es contain the same physical address and thus reference the same page-directory-pointer table. Any PDPTE in that table may result in two PDPTE-cache entries, each associated with a different set of linear addresses. Specifically, suppose that the n_1^{th} and n_2^{th} entries in the PML4 table contain the same physical address. This implies that the physical address in the m^{th} PDPTE in the page-directory-pointer table would appear in the PDPTE-cache entries associated with both p_1 and p_2 , where $(p_1 \gg 9) = n_1$, $(p_2 \gg 9) = n_2$, and $(p_1 \& 1\text{FFH}) = (p_2 \& 1\text{FFH}) = m$. This is because both PDPTE-cache entries use the same PDPTE, one resulting from a reference from the n_1^{th} PML4E and one from the n_2^{th} PML4E.
- Suppose that the first PML4E (i.e., the one in position 0) contains the physical address X in CR3 (the physical address of the PML4 table). This implies the following:

- Any PML4-cache entry associated with linear addresses with 0 in bits 47:39 contains address X.
- Any PDPTE-cache entry associated with linear addresses with 0 in bits 47:30 contains address X. This is because the translation for a linear address for which the value of bits 47:30 is 0 uses the value of bits 47:39 (0) to locate a page-directory-pointer table at address X (the address of the PML4 table). It then uses the value of bits 38:30 (also 0) to find address X again and to store that address in the PDPTE-cache entry.
- Any PDE-cache entry associated with linear addresses with 0 in bits 47:21 contains address X for similar reasons.
- Any TLB entry for page number 0 (associated with linear addresses with 0 in bits 47:12) translates to page frame $X \gg 12$ for similar reasons.

The same PML4E contributes its address X to all these cache entries because the self-referencing nature of the entry causes it to be used as a PML4E, a PDPTE, a PDE, and a PTE.

5.10.4 Invalidation of TLBs and Paging-Structure Caches

As noted in Section 5.10.2 and Section 5.10.3, the processor may create entries in the TLBs and the paging-structure caches when linear addresses are translated, and it may retain these entries even after the paging structures used to create them have been modified. To ensure that linear-address translation uses the modified paging structures, software should take action to invalidate any cached entries that may contain information that has since been modified.

5.10.4.1 Operations that Invalidate TLBs and Paging-Structure Caches

The following instructions invalidate entries in the TLBs and the paging-structure caches:

- **INVLPG.** This instruction takes a single operand, which is a linear address. The instruction invalidates any TLB entries that are for a page number corresponding to the linear address and that are associated with the current PCID. It also invalidates any global TLB entries with that page number, regardless of PCID (see Section 5.10.2.4).¹ INVLPG also invalidates all entries in all paging-structure caches associated with the current PCID, regardless of the linear addresses to which they correspond.
- **INVPCID.** The operation of this instruction is based on instruction operands, called the INVPCID type and the INVPCID descriptor. Four INVPCID types are currently defined:
 - **Individual-address.** If the INVPCID type is 0, the logical processor invalidates mappings—except global translations—associated with the PCID specified in the INVPCID descriptor and that would be used to translate the linear address specified in the INVPCID descriptor.² (The instruction may also invalidate global translations, as well as mappings associated with other PCIDs and for other linear addresses.)
 - **Single-context.** If the INVPCID type is 1, the logical processor invalidates all mappings—except global translations—associated with the PCID specified in the INVPCID descriptor. (The instruction may also invalidate global translations, as well as mappings associated with other PCIDs.)
 - **All-context, including globals.** If the INVPCID type is 2, the logical processor invalidates mappings—including global translations—associated with all PCIDs.
 - **All-context.** If the INVPCID type is 3, the logical processor invalidates mappings—except global translations—associated with all PCIDs. (The instruction may also invalidate global translations.)

See Chapter 3 of the *Intel 64 and IA-32 Architecture Software Developer's Manual, Volume 2A* for details of the INVPCID instruction.

- **MOV to CR0.** The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if it changes the value of CR0.PG from 1 to 0.
- **MOV to CR3.** The behavior of the instruction depends on the value of CR4.PCIDE:

1. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 5.10.2.3), the instruction invalidates all of them.
2. If the paging structures map the linear address using a page larger than 4 KBytes and there are multiple TLB entries for that page (see Section 5.10.2.3), the instruction invalidates all of them.

- If CR4.PCIDE = 0, the instruction invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.
- If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, the instruction invalidates all TLB entries associated with the PCID specified in bits 11:0 of the instruction's source operand except those for global pages. It also invalidates all entries in all paging-structure caches associated with that PCID. It is not required to invalidate entries in the TLBs and paging-structure caches that are associated with other PCIDs.
- If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1, the instruction is not required to invalidate any TLB entries or entries in paging-structure caches.
- MOV to CR4. The behavior of the instruction depends on the bits being modified:
 - The instruction invalidates all TLB entries (including global entries) and all entries in all paging-structure caches (for all PCIDs) if (1) it changes the value of CR4.PGE;¹ or (2) it changes the value of the CR4.PCIDE from 1 to 0.
 - The instruction invalidates all TLB entries and all entries in all paging-structure caches for the current PCID if (1) it changes the value of CR4.PAE; or (2) it changes the value of CR4.SMEP from 0 to 1.
- Task switch. If a task switch changes the value of CR3, it invalidates all TLB entries associated with PCID 000H except those for global pages. It also invalidates all entries in all paging-structure caches associated with PCID 000H.²
- VMX transitions. See Section 5.11.1.

The processor is always free to invalidate additional entries in the TLBs and paging-structure caches. The following are some examples:

- INVLPG may invalidate TLB entries for pages other than the one corresponding to its linear-address operand. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the current PCID.
- INVPCID may invalidate TLB entries for pages other than the one corresponding to the specified linear address. It may invalidate TLB entries and paging-structure-cache entries associated with PCIDs other than the specified PCID.
- MOV to CR0 may invalidate TLB entries even if CR0.PG is not changing. For example, this may occur if either CR0.CD or CR0.NW is modified.
- MOV to CR3 may invalidate TLB entries for global pages. If CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 0, it may invalidate TLB entries and entries in the paging-structure caches associated with PCIDs other than the PCID it is establishing. It may invalidate entries if CR4.PCIDE = 1 and bit 63 of the instruction's source operand is 1.
- MOV to CR4 may invalidate TLB entries when changing CR4.PSE or when changing CR4.SMEP from 1 to 0.
- On a processor supporting Hyper-Threading Technology, invalidations performed on one logical processor may invalidate entries in the TLBs and paging-structure caches used by other logical processors.

(Other instructions and operations may invalidate entries in the TLBs and the paging-structure caches, but the instructions identified above are recommended.)

In addition to the instructions identified above, page faults invalidate entries in the TLBs and paging-structure caches. In particular, a page-fault exception resulting from an attempt to use a linear address will invalidate any TLB entries that are for a page number corresponding to that linear address and that are associated with the current PCID. It also invalidates all entries in the paging-structure caches that would be used for that linear address and that are associated with the current PCID.³ These invalidations ensure that the page-fault exception will not recur (if the faulting instruction is re-executed) if it would not be caused by the contents of the paging structures

1. If CR4.PGE is changing from 0 to 1, there were no global TLB entries before the execution; if CR4.PGE is changing from 1 to 0, there will be no global TLB entries after the execution.

2. Task switches do not occur in IA-32e mode and thus cannot occur with 4-level paging. Since CR4.PCIDE can be set only with 4-level paging, task switches occur only with CR4.PCIDE = 0.

3. Unlike INVLPG, page faults need not invalidate **all** entries in the paging-structure caches, only those that would be used to translate the faulting linear address.

in memory (and if, therefore, it resulted from cached entries that were not invalidated after the paging structures were modified in memory).

As noted in Section 5.10.2, some processors may choose to cache multiple smaller-page TLB entries for a translation specified by the paging structures to use a page larger than 4 KBytes. There is no way for software to be aware that multiple translations for smaller pages have been used for a large page. The INVLPG instruction and page faults provide the same assurances that they provide when a single TLB entry is used: they invalidate all TLB entries corresponding to the translation specified by the paging structures.

5.10.4.2 Recommended Invalidation

The following items provide some recommendations regarding when software should perform invalidations:

- If software modifies a paging-structure entry that maps a page (rather than referencing another paging structure), it should execute INVLPG for any linear address with a page number whose translation uses that paging-structure entry.¹
(If the paging-structure entry may be used in the translation of different page numbers — see Section 5.10.3.3 — software should execute INVLPG for linear addresses with each of those page numbers; alternatively, it could use MOV to CR3 or MOV to CR4.)
- If software modifies a paging-structure entry that references another paging structure, it may use one of the following approaches depending upon the types and number of translations controlled by the modified entry:
 - Execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry. However, if no page numbers that would use the entry have translations (e.g., because the P flags are 0 in all entries in the paging structure referenced by the modified entry), it remains necessary to execute INVLPG at least once.
 - Execute MOV to CR3 if the modified entry controls no global pages.
 - Execute MOV to CR4 to modify CR4.PGE.
- If CR4.PCIDE = 1 and software modifies a paging-structure entry that does not map a page or in which the G flag (bit 8) is 0, additional steps are required if the entry may be used for PCIDs other than the current one. Any one of the following suffices:
 - Execute MOV to CR4 to modify CR4.PGE, either immediately or before again using any of the affected PCIDs. For example, software could use different (previously unused) PCIDs for the processes that used the affected PCIDs.
 - For each affected PCID, execute MOV to CR3 to make that PCID current (and to load the address of the appropriate PML4 table). If the modified entry controls no global pages and bit 63 of the source operand to MOV to CR3 was 0, no further steps are required. Otherwise, execute INVLPG for linear addresses with each of the page numbers with translations that would use the entry; if no page numbers that would use the entry have translations, execute INVLPG at least once.
- If software using PAE paging modifies a PDPTE, it should reload CR3 with the register's current value to ensure that the modified PDPTE is loaded into the corresponding PDPTE register (see Section 5.4.1).
- If the nature of the paging structures is such that a single entry may be used for multiple purposes (see Section 5.10.3.3), software should perform invalidations for all of these purposes. For example, if a single entry might serve as both a PDE and PTE, it may be necessary to execute INVLPG with two (or more) linear addresses, one that uses the entry as a PDE and one that uses it as a PTE. (Alternatively, software could use MOV to CR3 or MOV to CR4.)
- As noted in Section 5.10.2, the TLBs may subsequently contain multiple translations for the address range if software modifies the paging structures so that the page size used for a 4-KByte range of linear addresses changes. A reference to a linear address in the address range may use any of these translations.

Software wishing to prevent this uncertainty should not write to a paging-structure entry in a way that would change, for any linear address, both the page size and either the page frame, access rights, or other attributes. It can instead use the following algorithm: first clear the P flag in the relevant paging-structure entry (e.g.,

1. One execution of INVLPG is sufficient even for a page with size greater than 4 KBytes.

PDE); then invalidate any translations for the affected linear addresses (see above); and then modify the relevant paging-structure entry to set the P flag and establish modified translation(s) for the new page size.

- Software should clear bit 63 of the source operand to a MOV to CR3 instruction that establishes a PCID that had been used earlier for a different linear-address space (e.g., with a different value in bits 51:12 of CR3). This ensures invalidation of any information that may have been cached for the previous linear-address space.

This assumes that both linear-address spaces use the same global pages and that it is thus not necessary to invalidate any global TLB entries. If that is not the case, software should invalidate those entries by executing MOV to CR4 to modify CR4.PGE.

5.10.4.3 Optional Invalidation

The following items describe cases in which software may choose not to invalidate and the potential consequences of that choice:

- If a paging-structure entry is modified to change the P flag from 0 to 1, no invalidation is necessary. This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the P flag is 0.¹
- If a paging-structure entry is modified to change the accessed flag from 0 to 1, no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging-structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, failure to perform an invalidation may result in a “spurious” page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 5.10.4.1).
- If CR4.SMEP = 0 and a paging-structure entry is modified to change the U/S flag from 0 to 1, failure to perform an invalidation may result in a “spurious” page-fault exception (e.g., in response to an attempted user-mode access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 5.10.4.1).
- If a paging-structure entry is modified to change the XD flag from 1 to 0, failure to perform an invalidation may result in a “spurious” page-fault exception (e.g., in response to an attempted instruction fetch) but no other adverse behavior. Such an exception will occur at most once for each affected linear address (see Section 5.10.4.1).
- If a paging-structure entry is modified to change the accessed flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent access to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such an access has not occurred.
- If software modifies a paging-structure entry that identifies the final physical address for a linear address (either a PTE or a paging-structure entry in which the PS flag is 1) to change the dirty flag from 1 to 0, failure to perform an invalidation may result in the processor not setting that bit in response to a subsequent write to a linear address whose translation uses the entry. Software cannot interpret the bit being clear as an indication that such a write has not occurred.
- The read of a paging-structure entry in translating an address being used to fetch an instruction may appear to execute before an earlier write to that paging-structure entry if there is no serializing instruction between the write and the instruction fetch. Note that the invalidating instructions identified in Section 5.10.4.1 are all serializing instructions.
- Section 5.10.3.3 describes situations in which a single paging-structure entry may contain information cached in multiple entries in the paging-structure caches. Because all entries in these caches are invalidated by any execution of INVLPG, it is not necessary to follow the modification of such a paging-structure entry by executing INVLPG multiple times solely for the purpose of invalidating these multiple cached entries. (It may be necessary to do so to invalidate multiple TLB entries.)

1. If it is also the case that no invalidation was performed the last time the P flag was changed from 1 to 0, the processor may use a TLB entry or paging-structure cache entry that was created when the P flag had earlier been 1.

5.10.4.4 Delayed Invalidation

Required invalidations may be delayed under some circumstances. Software developers should understand that, between the modification of a paging-structure entry and execution of the invalidation instruction recommended in Section 5.10.4.2, the processor may use translations based on either the old value or the new value of the paging-structure entry. The following items describe some of the potential consequences of delayed invalidation:

- If a paging-structure entry is modified to change the P flag from 1 to 0, an access to a linear address whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the R/W flag from 0 to 1, write accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the U/S flag from 0 to 1, user-mode accesses to linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.
- If a paging-structure entry is modified to change the XD flag from 1 to 0, instruction fetches from linear addresses whose translation is controlled by this entry may or may not cause a page-fault exception.

As noted in Section 10.1.1, an x87 instruction or an SSE instruction that accesses data larger than a quadword may be implemented using multiple memory accesses. If such an instruction stores to memory and invalidation has been delayed, some of the accesses may complete (writing to memory) while another causes a page-fault exception.¹ In this case, the effects of the completed accesses may be visible to software even though the overall instruction caused a fault.

In some cases, the consequences of delayed invalidation may not affect software adversely. For example, when freeing a portion of the linear-address space (by marking paging-structure entries “not present”), invalidation using INVLPG may be delayed if software does not re-allocate that portion of the linear-address space or the memory that had been associated with it. However, because of speculative execution (or errant software), there may be accesses to the freed portion of the linear-address space before the invalidations occur. In this case, the following can happen:

- Reads can occur to the freed portion of the linear-address space. Therefore, invalidation should not be delayed for an address range that has read side effects.
- The processor may retain entries in the TLBs and paging-structure caches for an extended period of time. Software should not assume that the processor will not use entries associated with a linear address simply because time has passed.
- As noted in Section 5.10.3.1, the processor may create an entry in a paging-structure cache even if there are no translations for any linear address that might use that entry. Thus, if software has marked “not present” all entries in a page table, the processor may subsequently create a PDE-cache entry for the PDE that references that page table (assuming that the PDE itself is marked “present”).
- If software attempts to write to the freed portion of the linear-address space, the processor might not generate a page fault. (Such an attempt would likely be the result of a software error.) For that reason, the page frames previously associated with the freed portion of the linear-address space should not be reallocated for another purpose until the appropriate invalidations have been performed.

5.10.5 Propagation of Paging-Structure Changes to Multiple Processors

As noted in Section 5.10.4, software that modifies a paging-structure entry may need to invalidate entries in the TLBs and paging-structure caches that were derived from the modified entry before it was modified. In a system containing more than one logical processor, software must account for the fact that there may be entries in the TLBs and paging-structure caches of logical processors other than the one used to modify the paging-structure entry. The process of propagating the changes to a paging-structure entry is commonly referred to as “TLB shoot-down.”

TLB shutdown can be done using memory-based semaphores and/or interprocessor interrupts (IPI). The following items describe a simple but inefficient example of a TLB shutdown algorithm for processors supporting the Intel-64 and IA-32 architectures:

1. If the accesses are to different pages, this may occur even if invalidation has not been delayed.

1. Begin barrier: Stop all but one logical processor; that is, cause all but one to execute the HLT instruction or to enter a spin loop.
2. Allow the active logical processor to change the necessary paging-structure entries.
3. Allow all logical processors to perform invalidations appropriate to the modifications to the paging-structure entries.
4. Allow all logical processors to resume normal operation.

Alternative, performance-optimized, TLB shutdown algorithms may be developed; however, software developers must take care to ensure that the following conditions are met:

- All logical processors that are using the paging structures that are being modified must participate and perform appropriate invalidations after the modifications are made.
- If the modifications to the paging-structure entries are made before the barrier or if there is no barrier, the operating system must ensure one of the following: (1) that the affected linear-address range is not used between the time of modification and the time of invalidation; or (2) that it is prepared to deal with the consequences of the affected linear-address range being used during that period. For example, if the operating system does not allow pages being freed to be reallocated for another purpose until after the required invalidations, writes to those pages by errant software will not unexpectedly modify memory that is in use.
- Software must be prepared to deal with reads, instruction fetches, and prefetch requests to the affected linear-address range that are a result of speculative execution that would never actually occur in the executed code path.

When multiple logical processors are using the same linear-address space at the same time, they must coordinate before any request to modify the paging-structure entries that control that linear-address space. In these cases, the barrier in the TLB shutdown routine may not be required. For example, when freeing a range of linear addresses, some other mechanism can assure no logical processor is using that range before the request to free it is made. In this case, a logical processor freeing the range can clear the P flags in the PTEs associated with the range, free the physical page frames associated with the range, and then signal the other logical processors using that linear-address space to perform the necessary invalidations. All the affected logical processors must complete their invalidations before the linear-address range and the physical page frames previously associated with that range can be reallocated.

5.11 INTERACTIONS WITH VIRTUAL-MACHINE EXTENSIONS (VMX)

The architecture for virtual-machine extensions (VMX) includes features that interact with paging. Section 5.11.1 discusses ways in which VMX-specific control transfers, called VMX transitions specially affect paging. Section 5.11.2 gives an overview of VMX features specifically designed to support address translation.

5.11.1 VMX Transitions

The VMX architecture defines two control transfers called **VM entries** and **VM exits**; collectively, these are called **VMX transitions**. VM entries and VM exits are described in detail in Chapter 27 and Chapter 28, respectively, in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C. The following items identify paging-related details:

- VMX transitions modify the CR0 and CR4 registers and the IA32_EFER MSR concurrently. For this reason, they allow transitions between paging modes that would not otherwise be possible:
 - VM entries allow transitions from 4-level paging directly to either 32-bit paging or PAE paging.
 - VM exits allow transitions from either 32-bit paging or PAE paging directly to 4-level paging or 5-level paging.
- VMX transitions that result in PAE paging load the PDPTE registers (see Section 5.4.1) as follows:
 - VM entries load the PDPTE registers either from the physical address being loaded into CR3 or from the virtual-machine control structure (VMCS); see Section 28.3.2.4.
 - VM exits load the PDPTE registers from the physical address being loaded into CR3; see Section 29.5.4.

- VMX transitions invalidate the TLBs and paging-structure caches based on certain control settings. See Section 28.3.2.5 and Section 29.5.5 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

5.11.2 VMX Support for Address Translation

Chapter 30, "VMX Support for Address Translation," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C, describes two features of the virtual-machine extensions (VMX) that interact directly with paging. These are **virtual-processor identifiers (VPIDs)** and the **extended page table** mechanism (**EPT**).

VPIDs provide a way for software to identify to the processor the address spaces for different "virtual processors." The processor may use this identification to maintain concurrently information for multiple address spaces in its TLBs and paging-structure caches, even when non-zero PCIDs are not being used. See Section 30.1 for details.

When EPT is in use, the addresses in the paging-structures are not used as physical addresses to access memory and memory-mapped I/O. Instead, they are treated as **guest-physical** addresses and are translated through a set of EPT paging structures to produce physical addresses. EPT can also specify its own access rights and memory typing; these are used on conjunction with those specified in this chapter. See Section 30.3 for more information.

Both VPIDs and EPT may change the way that a processor maintains information in TLBs and paging structure caches and the ways in which software can manage that information. Some of the behaviors documented in Section 5.10 may change. See Section 30.4 for details.

5.12 USING PAGING FOR VIRTUAL MEMORY

With paging, portions of the linear-address space need not be mapped to the physical-address space; data for the unmapped addresses can be stored externally (e.g., on disk). This method of mapping the linear-address space is referred to as virtual memory or demand-paged virtual memory.

Paging divides the linear address space into fixed-size pages that can be mapped into the physical-address space and/or external storage. When a program (or task) references a linear address, the processor uses paging to translate the linear address into a corresponding physical address if such an address is defined.

If the page containing the linear address is not currently mapped into the physical-address space, the processor generates a page-fault exception as described in Section 5.7. The handler for page-fault exceptions typically directs the operating system or executive to load data for the unmapped page from external storage into physical memory (perhaps writing a different page from physical memory out to external storage in the process) and to map it using paging (by updating the paging structures). When the page has been loaded into physical memory, a return from the exception handler causes the instruction that generated the exception to be restarted.

Paging differs from segmentation through its use of fixed-size pages. Unlike segments, which usually are the same size as the code or data structures they hold, pages have a fixed size. If segmentation is the only form of address translation used, a data structure present in physical memory will have all of its parts in memory. If paging is used, a data structure can be partly in memory and partly in disk storage.

5.13 MAPPING SEGMENTS TO PAGES

The segmentation and paging mechanisms provide support for a wide variety of approaches to memory management. When segmentation and paging are combined, segments can be mapped to pages in several ways. To implement a flat (unsegmented) addressing environment, for example, all the code, data, and stack modules can be mapped to one or more large segments (up to 4-GBytes) that share same range of linear addresses (see Figure 3-2 in Section 3.2.2). Here, segments are essentially invisible to applications and the operating-system or executive. If paging is used, the paging mechanism can map a single linear-address space (contained in a single segment) into virtual memory. Alternatively, each program (or task) can have its own large linear-address space (contained in its own segment), which is mapped into virtual memory through its own paging structures.

Segments can be smaller than the size of a page. If one of these segments is placed in a page which is not shared with another segment, the extra memory is wasted. For example, a small data structure, such as a 1-Byte semaphore, occupies 4 KBytes if it is placed in a page by itself. If many semaphores are used, it is more efficient to pack them into a single page.

The Intel-64 and IA-32 architectures do not enforce correspondence between the boundaries of pages and segments. A page can contain the end of one segment and the beginning of another. Similarly, a segment can contain the end of one page and the beginning of another.

Memory-management software may be simpler and more efficient if it enforces some alignment between page and segment boundaries. For example, if a segment which can fit in one page is placed in two pages, there may be twice as much paging overhead to support access to that segment.

One approach to combining paging and segmentation that simplifies memory-management software is to give each segment its own page table, as shown in Figure 5-13. This convention gives the segment a single entry in the page directory, and this entry provides the access control information for paging the entire segment.

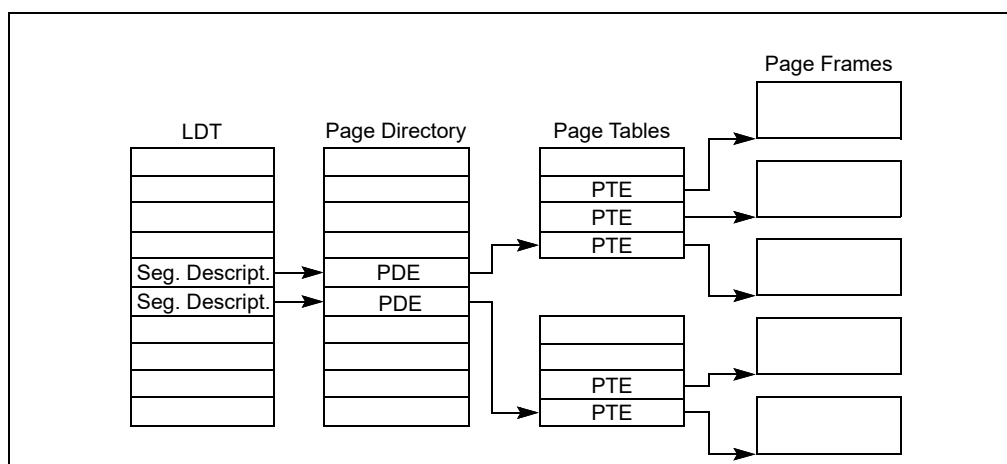


Figure 5-13. Memory Management Convention That Assigns a Page Table to Each Segment

11. Updates to Chapter 6, Volume 3A

Change bars and violet text show changes to Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Updated Section 6.13.3, "Reserved Bit Checking" with MAXPHYADDR. Table 6-8 and Table 6-9 were also updated for MAXPHYADDR.

In protected mode, the Intel 64 and IA-32 architectures provide a protection mechanism that operates at both the segment level and the page level. This protection mechanism provides the ability to limit access to certain segments or pages based on privilege levels (four privilege levels for segments and two privilege levels for pages). For example, critical operating-system code and data can be protected by placing them in more privileged segments than those that contain applications code. The processor's protection mechanism will then prevent application code from accessing the operating-system code and data in any but a controlled, defined manner.

Segment and page protection can be used at all stages of software development to assist in localizing and detecting design problems and bugs. It can also be incorporated into end-products to offer added robustness to operating systems, utilities software, and applications software.

When the protection mechanism is used, each memory reference is checked to verify that it satisfies various protection checks. All checks are made before the memory cycle is started; any violation results in an exception. Because checks are performed in parallel with address translation, there is no performance penalty. The protection checks that are performed fall into the following categories:

- Limit checks.
- Type checks.
- Privilege level checks.
- Restriction of addressable domain.
- Restriction of procedure entry-points.
- Restriction of instruction set.

All protection violation results in an exception being generated. See Chapter 7, "Interrupt and Exception Handling," for an explanation of the exception mechanism. This chapter describes the protection mechanism and the violations which lead to exceptions.

The following sections describe the protection mechanism available in protected mode. See Chapter 22, "8086 Emulation," for information on protection in real-address and virtual-8086 mode.

6.1 ENABLING AND DISABLING SEGMENT AND PAGE PROTECTION

Setting the PE flag in register CR0 causes the processor to switch to protected mode, which in turn enables the segment-protection mechanism. Once in protected mode, there is no control bit for turning the protection mechanism on or off. The part of the segment-protection mechanism that is based on privilege levels can essentially be disabled while still in protected mode by assigning a privilege level of 0 (most privileged) to all segment selectors and segment descriptors. This action disables the privilege level protection barriers between segments, but other protection checks such as limit checking and type checking are still carried out.

Page-level protection is automatically enabled when paging is enabled (by setting the PG flag in register CR0). Here again there is no mode bit for turning off page-level protection once paging is enabled. However, page-level protection can be disabled by performing the following operations:

- Clear the WP flag in control register CR0.
- Set the read/write (R/W) and user/supervisor (U/S) flags for each page-directory and page-table entry.

This action makes each page a writable, user page, which in effect disables page-level protection.

6.2 FIELDS AND FLAGS USED FOR SEGMENT-LEVEL AND PAGE-LEVEL PROTECTION

The processor's protection mechanism uses the following fields and flags in the system data structures to control access to segments and pages:

- **Descriptor type (S) flag** — (Bit 12 in the second doubleword of a segment descriptor.) Determines if the segment descriptor is for a system segment or a code or data segment.
- **Type field** — (Bits 8 through 11 in the second doubleword of a segment descriptor.) Determines the type of code, data, or system segment.
- **Limit field** — (Bits 0 through 15 of the first doubleword and bits 16 through 19 of the second doubleword of a segment descriptor.) Determines the size of the segment, along with the G flag and E flag (for data segments).
- **G flag** — (Bit 23 in the second doubleword of a segment descriptor.) Determines the size of the segment, along with the limit field and E flag (for data segments).
- **E flag** — (Bit 10 in the second doubleword of a data-segment descriptor.) Determines the size of the segment, along with the limit field and G flag.
- **Descriptor privilege level (DPL) field** — (Bits 13 and 14 in the second doubleword of a segment descriptor.) Determines the privilege level of the segment.
- **Requested privilege level (RPL) field** — (Bits 0 and 1 of any segment selector.) Specifies the requested privilege level of a segment selector.
- **Current privilege level (CPL) field** — (Bits 0 and 1 of the CS segment register.) Indicates the privilege level of the currently executing program or procedure. The term current privilege level (CPL) refers to the setting of this field.
- **User/supervisor (U/S) flag** — (Bit 2 of paging-structure entries.) Determines the type of page: user or supervisor.
- **Read/write (R/W) flag** — (Bit 1 of paging-structure entries.) Determines the type of access allowed to a page: read-only or read/write.
- **Execute-disable (XD) flag** — (Bit 63 of certain paging-structure entries.) Determines the type of access allowed to a page: executable or not-executable.

Figure 6-1 shows the location of the various fields and flags in the data-, code-, and system-segment descriptors; Figure 3-6 shows the location of the RPL (or CPL) field in a segment selector (or the CS register); and Chapter 5 identifies the locations of the U/S, R/W, and XD flags in the paging-structure entries.

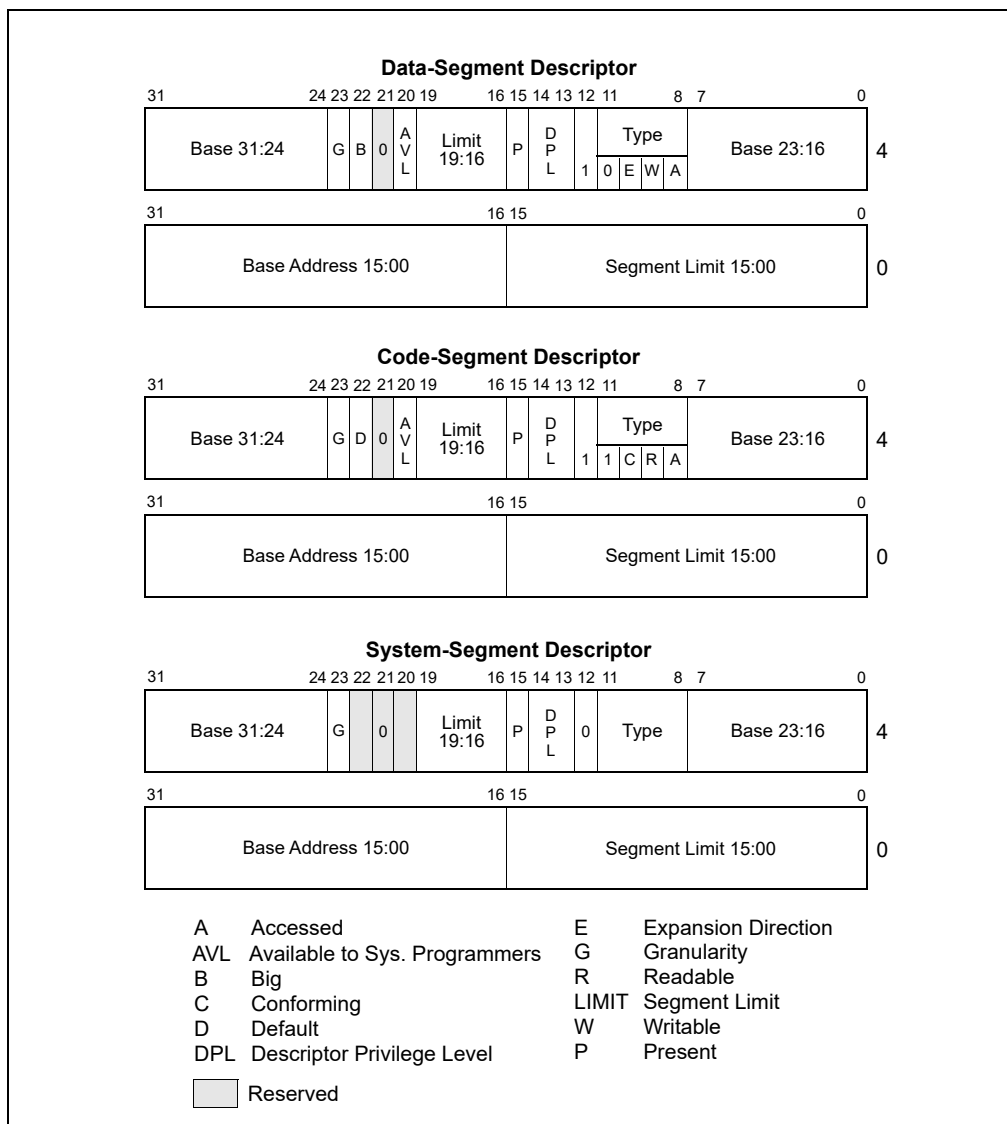


Figure 6-1. Descriptor Fields Used for Protection

Many different styles of protection schemes can be implemented with these fields and flags. When the operating system creates a descriptor, it places values in these fields and flags in keeping with the particular protection style chosen for an operating system or executive. Application programs do not generally access or modify these fields and flags.

The following sections describe how the processor uses these fields and flags to perform the various categories of checks described in the introduction to this chapter.

6.2.1 Code-Segment Descriptor in 64-bit Mode

Code segments continue to exist in 64-bit mode even though, for address calculations, the segment base is treated as zero. Some code-segment (CS) descriptor content (the base address and limit fields) is ignored; the remaining fields function normally (except for the readable bit in the type field).

Code segment descriptors and selectors are needed in IA-32e mode to establish the processor's operating mode and execution privilege-level. The usage is as follows:

- IA-32e mode uses a previously unused bit in the CS descriptor. Bit 53 is defined as the 64-bit (L) flag and is used to select between 64-bit mode and compatibility mode when IA-32e mode is active (IA32_EFER.LMA = 1). See Figure 6-2.
 - If CS.L = 0 and IA-32e mode is active, the processor is running in compatibility mode. In this case, CS.D selects the default size for data and addresses. If CS.D = 0, the default data and address size is 16 bits. If CS.D = 1, the default data and address size is 32 bits.
 - If CS.L = 1 and IA-32e mode is active, the only valid setting is CS.D = 0. This setting indicates a default operand size of 32 bits and a default address size of 64 bits. The CS.L = 1 and CS.D = 1 bit combination is reserved for future use and a #GP fault will be generated on an attempt to use a code segment with these bits set in IA-32e mode.
- In IA-32e mode, the CS descriptor’s DPL is used for execution privilege checks (as in legacy 32-bit mode).

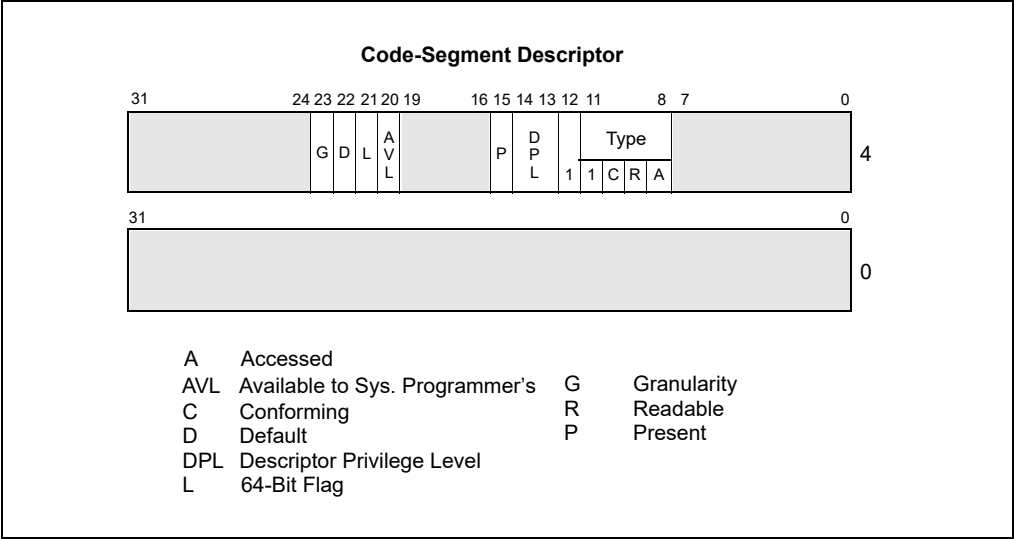


Figure 6-2. Descriptor Fields with Flags used in IA-32e Mode

6.3 LIMIT CHECKING

The limit field of a segment descriptor prevents programs or procedures from addressing memory locations outside the segment. The effective value of the limit depends on the setting of the G (granularity) flag (see Figure 6-1). For data segments, the limit also depends on the E (expansion direction) flag and the B (default stack pointer size and/or upper bound) flag. The E flag is one of the bits in the type field when the segment descriptor is for a data-segment type.

When the G flag is clear (byte granularity), the effective limit is the value of the 20-bit limit field in the segment descriptor. Here, the limit ranges from 0 to FFFFH (1 MByte). When the G flag is set (4-KByte page granularity), the processor scales the value in the limit field by a factor of 2^{12} (4 KBytes). In this case, the effective limit ranges from FFFH (4 KBytes) to FFFFFFFFH (4 GBytes). Note that when scaling is used (G flag is set), the lower 12 bits of a segment offset (address) are not checked against the limit; for example, note that if the segment limit is 0, offsets 0 through FFFH are still valid.

For all types of segments except expand-down data segments, the effective limit is the last address that is allowed to be accessed in the segment, which is one less than the size, in bytes, of the segment. The processor causes a general-protection exception (or, if the segment is SS, a stack-fault exception) any time an attempt is made to access the following addresses in a segment:

- A byte at an offset greater than the effective limit
- A word at an offset greater than the (effective-limit – 1)
- A doubleword at an offset greater than the (effective-limit – 3)
- A quadword at an offset greater than the (effective-limit – 7)

- A double quadword at an offset greater than the (effective limit – 15)

When the effective limit is FFFFFFFFH (4 GBytes), these accesses may or may not cause the indicated exceptions. Behavior is implementation-specific and may vary from one execution to another.

For expand-down data segments, the segment limit has the same function but is interpreted differently. Here, the effective limit specifies the last address that is not allowed to be accessed within the segment; the range of valid offsets is from (effective-limit + 1) to FFFFFFFFH if the B flag is set and from (effective-limit + 1) to FFFFH if the B flag is clear. An expand-down segment has maximum size when the segment limit is 0.

Limit checking catches programming errors such as runaway code, runaway subscripts, and invalid pointer calculations. These errors are detected when they occur, so identification of the cause is easier. Without limit checking, these errors could overwrite code or data in another segment.

In addition to checking segment limits, the processor also checks descriptor table limits. The GDTR and IDTR registers contain 16-bit limit values that the processor uses to prevent programs from selecting a segment descriptors outside the respective descriptor tables. The LDTR and task registers contain 32-bit segment limit value (read from the segment descriptors for the current LDT and TSS, respectively). The processor uses these segment limits to prevent accesses beyond the bounds of the current LDT and TSS. See Section 3.5.1, “Segment Descriptor Tables,” for more information on the GDT and LDT limit fields; see Section 7.10, “Interrupt Descriptor Table (IDT),” for more information on the IDT limit field; and see Section 9.2.4, “Task Register,” for more information on the TSS segment limit field.

6.3.1 Limit Checking in 64-bit Mode

In 64-bit mode, the processor does not perform runtime limit checking on code or data segments. However, the processor does check descriptor-table limits.

6.4 TYPE CHECKING

Segment descriptors contain type information in two places:

- The S (descriptor type) flag.
- The type field.

The processor uses this information to detect programming errors that result in an attempt to use a segment or gate in an incorrect or unintended manner.

The S flag indicates whether a descriptor is a system type or a code or data type. The type field provides 4 additional bits for use in defining various types of code, data, and system descriptors. Table 3-1 shows the encoding of the type field for code and data descriptors; Table 3-2 shows the encoding of the field for system descriptors.

The processor examines type information at various times while operating on segment selectors and segment descriptors. The following list gives examples of typical operations where type checking is performed (this list is not exhaustive):

- **When a segment selector is loaded into a segment register** — Certain segment registers can contain only certain descriptor types, for example:
 - The CS register only can be loaded with a selector for a code segment.
 - Segment selectors for code segments that are not readable or for system segments cannot be loaded into data-segment registers (DS, ES, FS, and GS).
 - Only segment selectors of writable data segments can be loaded into the SS register.
- When a segment selector is loaded into the LDTR or task register — For example:
 - The LDTR can only be loaded with a selector for an LDT.
 - The task register can only be loaded with a segment selector for a TSS.
- **When instructions access segments whose descriptors are already loaded into segment registers** — Certain segments can be used by instructions only in certain predefined ways, for example:
 - No instruction may write into an executable segment.

- No instruction may write into a data segment if it is not writable.
- No instruction may read an executable segment unless the readable flag is set.
- **When an instruction operand contains a segment selector** — Certain instructions can access segments or gates of only a particular type, for example:
 - A far CALL or far JMP instruction can only access a segment descriptor for a conforming code segment, nonconforming code segment, call gate, task gate, or TSS.
 - The LLDT instruction must reference a segment descriptor for an LDT.
 - The LTR instruction must reference a segment descriptor for a TSS.
 - The LAR instruction must reference a segment or gate descriptor for an LDT, TSS, call gate, task gate, code segment, or data segment.
 - The LSL instruction must reference a segment descriptor for a LDT, TSS, code segment, or data segment.
 - IDT entries must be interrupt, trap, or task gates.
- **During certain internal operations** — For example:
 - On a far call or far jump (executed with a far CALL or far JMP instruction), the processor determines the type of control transfer to be carried out (call or jump to another code segment, a call or jump through a gate, or a task switch) by checking the type field in the segment (or gate) descriptor pointed to by the segment (or gate) selector given as an operand in the CALL or JMP instruction. If the descriptor type is for a code segment or call gate, a call or jump to another code segment is indicated; if the descriptor type is for a TSS or task gate, a task switch is indicated.
 - On a call or jump through a call gate (or on an interrupt- or exception-handler call through a trap or interrupt gate), the processor automatically checks that the segment descriptor being pointed to by the gate is for a code segment.
 - On a call or jump to a new task through a task gate (or on an interrupt- or exception-handler call to a new task through a task gate), the processor automatically checks that the segment descriptor being pointed to by the task gate is for a TSS.
 - On a call or jump to a new task by a direct reference to a TSS, the processor automatically checks that the segment descriptor being pointed to by the CALL or JMP instruction is for a TSS.
 - On return from a nested task (initiated by an IRET instruction), the processor checks that the previous task link field in the current TSS points to a TSS.

6.4.1 Null Segment Selector Checking

Attempting to load a null segment selector (see Section 3.4.2, “Segment Selectors”) into the CS or SS segment register generates a general-protection exception (#GP). A null segment selector can be loaded into the DS, ES, FS, or GS register, but any attempt to access a segment through one of these registers when it is loaded with a null segment selector results in a #GP exception being generated. Loading unused data-segment registers with a null segment selector is a useful method of detecting accesses to unused segment registers and/or preventing unwanted accesses to data segments.

6.4.1.1 NULL Segment Checking in 64-bit Mode

In 64-bit mode, the processor does not perform runtime checking on NULL segment selectors. The processor does not cause a #GP fault when an attempt is made to access memory where the referenced segment register has a NULL segment selector.

6.5 PRIVILEGE LEVELS

The processor’s segment-protection mechanism recognizes 4 privilege levels, numbered from 0 to 3. The greater numbers mean lesser privileges. Figure 6-3 shows how these levels of privilege can be interpreted as rings of protection.

The center (reserved for the most privileged code, data, and stacks) is used for the segments containing the critical software, usually the kernel of an operating system. Outer rings are used for less critical software. (Systems that use only 2 of the 4 possible privilege levels should use levels 0 and 3.)

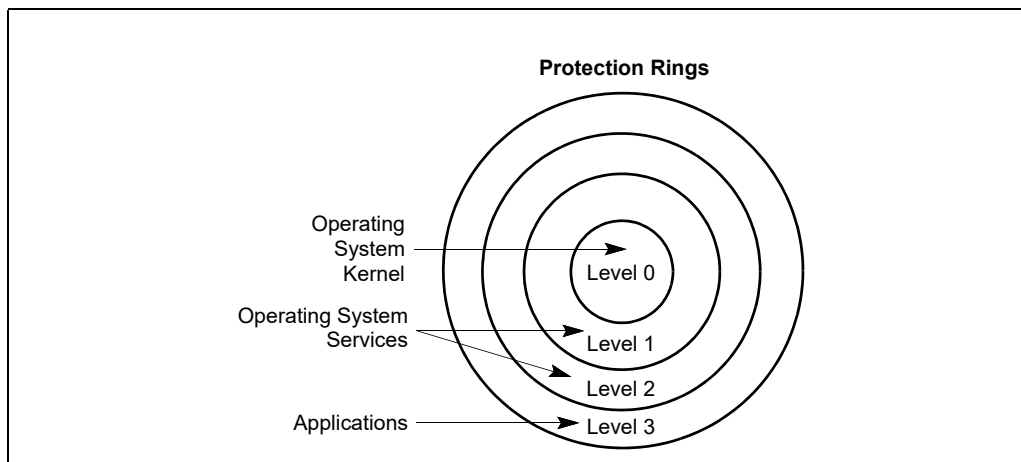


Figure 6-3. Protection Rings

The processor uses privilege levels to prevent a program or task operating at a lesser privilege level from accessing a segment with a greater privilege, except under controlled situations. When the processor detects a privilege level violation, it generates a general-protection exception (#GP).

To carry out privilege-level checks between code segments and data segments, the processor recognizes the following three types of privilege levels:

- **Current privilege level (CPL)** — The CPL is the privilege level of the currently executing program or task. It is stored in bits 0 and 1 of the CS and SS segment registers. Normally, the CPL is equal to the privilege level of the code segment from which instructions are being fetched. The processor changes the CPL when program control is transferred to a code segment with a different privilege level. The CPL is treated slightly differently when accessing conforming code segments. Conforming code segments can be accessed from any privilege level that is equal to or numerically greater (less privileged) than the DPL of the conforming code segment. Also, the CPL is not changed when the processor accesses a conforming code segment that has a different privilege level than the CPL.
- **Descriptor privilege level (DPL)** — The DPL is the privilege level of a segment or gate. It is stored in the DPL field of the segment or gate descriptor for the segment or gate. When the currently executing code segment attempts to access a segment or gate, the DPL of the segment or gate is compared to the CPL and RPL of the segment or gate selector (as described later in this section). The DPL is interpreted differently, depending on the type of segment or gate being accessed:
 - **Data segment** — The DPL indicates the numerically highest privilege level that a program or task can have to be allowed to access the segment. For example, if the DPL of a data segment is 1, only programs running at a CPL of 0 or 1 can access the segment.
 - **Nonconforming code segment (without using a call gate)** — The DPL indicates the privilege level that a program or task must be at to access the segment. For example, if the DPL of a nonconforming code segment is 0, only programs running at a CPL of 0 can access the segment.
 - **Call gate** — The DPL indicates the numerically highest privilege level that the currently executing program or task can be at and still be able to access the call gate. (This is the same access rule as for a data segment.)
 - **Conforming code segment and nonconforming code segment accessed through a call gate** — The DPL indicates the numerically lowest privilege level that a program or task can have to be allowed to access the segment. For example, if the DPL of a conforming code segment is 2, programs running at a CPL of 0 or 1 cannot access the segment.

- **TSS** — The DPL indicates the numerically highest privilege level that the currently executing program or task can be at and still be able to access the TSS. (This is the same access rule as for a data segment.)
- **Requested privilege level (RPL)** — The RPL is an override privilege level that is assigned to segment selectors. It is stored in bits 0 and 1 of the segment selector. The processor checks the RPL along with the CPL to determine if access to a segment is allowed. Even if the program or task requesting access to a segment has sufficient privilege to access the segment, access is denied if the RPL is not of sufficient privilege level. That is, if the RPL of a segment selector is numerically greater than the CPL, the RPL overrides the CPL, and vice versa. The RPL can be used to ensure that privileged code does not access a segment on behalf of an application program unless the program itself has access privileges for that segment. See Section 6.10.4, “Checking Caller Access Privileges (ARPL Instruction),” for a detailed description of the purpose and typical use of the RPL.

Privilege levels are checked when the segment selector of a segment descriptor is loaded into a segment register. The checks used for data access differ from those used for transfers of program control among code segments; therefore, the two kinds of accesses are considered separately in the following sections.

6.6 PRIVILEGE LEVEL CHECKING WHEN ACCESSING DATA SEGMENTS

To access operands in a data segment, the segment selector for the data segment must be loaded into the data-segment registers (DS, ES, FS, or GS) or into the stack-segment register (SS). (Segment registers can be loaded with the MOV, POP, LDS, LES, LFS, LGS, and LSS instructions.) Before the processor loads a segment selector into a segment register, it performs a privilege check (see Figure 6-4) by comparing the privilege levels of the currently running program or task (the CPL), the RPL of the segment selector, and the DPL of the segment’s segment descriptor. The processor loads the segment selector into the segment register if the DPL is numerically greater than or equal to both the CPL and the RPL. Otherwise, a general-protection fault is generated and the segment register is not loaded.

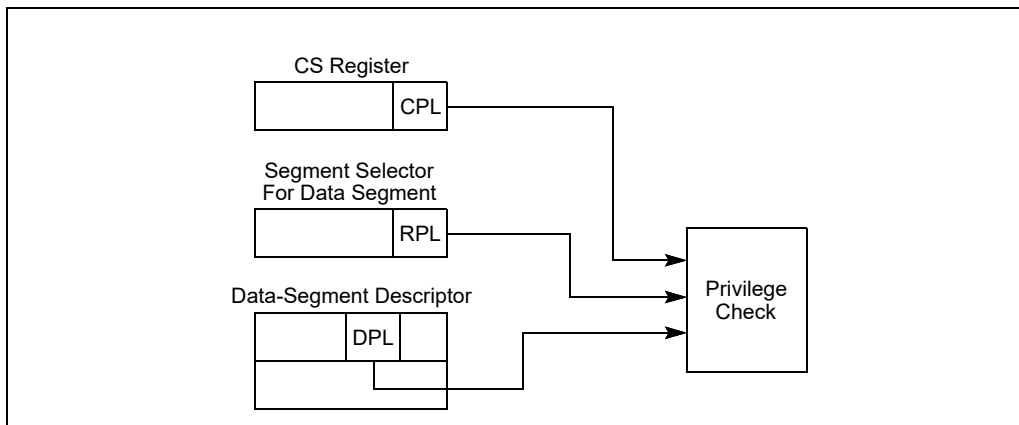


Figure 6-4. Privilege Check for Data Access

Figure 6-5 shows four procedures (located in codes segments A, B, C, and D), each running at different privilege levels and each attempting to access the same data segment.

1. The procedure in code segment A is able to access data segment E using segment selector E1, because the CPL of code segment A and the RPL of segment selector E1 are equal to the DPL of data segment E.
2. The procedure in code segment B is able to access data segment E using segment selector E2, because the CPL of code segment B and the RPL of segment selector E2 are both numerically lower than (more privileged) than the DPL of data segment E. A code segment B procedure can also access data segment E using segment selector E1.
3. The procedure in code segment C is not able to access data segment E using segment selector E3 (dotted line), because the CPL of code segment C and the RPL of segment selector E3 are both numerically greater than (less privileged) than the DPL of data segment E. Even if a code segment C procedure were to use segment selector

E1 or E2, such that the RPL would be acceptable, it still could not access data segment E because its CPL is not privileged enough.

4. The procedure in code segment D should be able to access data segment E because code segment D's CPL is numerically less than the DPL of data segment E. However, the RPL of segment selector E3 (which the code segment D procedure is using to access data segment E) is numerically greater than the DPL of data segment E, so access is not allowed. If the code segment D procedure were to use segment selector E1 or E2 to access the data segment, access would be allowed.

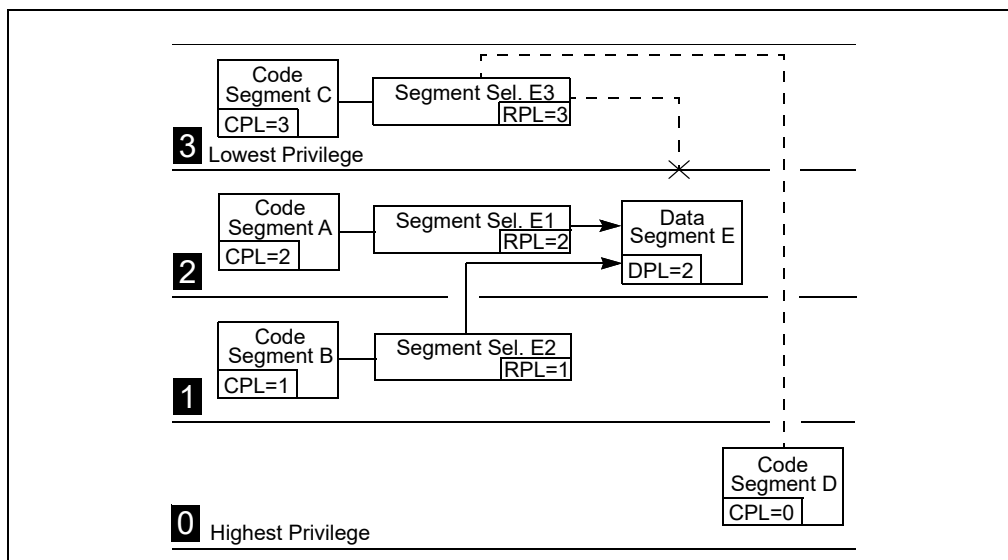


Figure 6-5. Examples of Accessing Data Segments From Various Privilege Levels

As demonstrated in the previous examples, the addressable domain of a program or task varies as its CPL changes. When the CPL is 0, data segments at all privilege levels are accessible; when the CPL is 1, only data segments at privilege levels 1 through 3 are accessible; when the CPL is 3, only data segments at privilege level 3 are accessible.

The RPL of a segment selector can always override the addressable domain of a program or task. When properly used, RPLs can prevent problems caused by accidental (or intentional) use of segment selectors for privileged data segments by less privileged programs or procedures.

It is important to note that the RPL of a segment selector for a data segment is under software control. For example, an application program running at a CPL of 3 can set the RPL for a data-segment selector to 0. With the RPL set to 0, only the CPL checks, not the RPL checks, will provide protection against deliberate, direct attempts to violate privilege-level security for the data segment. To prevent these types of privilege-level-check violations, a program or procedure can check access privileges whenever it receives a data-segment selector from another procedure (see Section 6.10.4, "Checking Caller Access Privileges (ARPL Instruction)").

6.6.1 Accessing Data in Code Segments

In some instances it may be desirable to access data structures that are contained in a code segment. The following methods of accessing data in code segments are possible:

- Load a data-segment register with a segment selector for a nonconforming, readable, code segment.
- Load a data-segment register with a segment selector for a conforming, readable, code segment.
- Use a code-segment override prefix (CS) to read a readable, code segment whose selector is already loaded in the CS register.

The same rules for accessing data segments apply to method 1. Method 2 is always valid because the privilege level of a conforming code segment is effectively the same as the CPL, regardless of its DPL. Method 3 is always valid because the DPL of the code segment selected by the CS register is the same as the CPL.

6.7 PRIVILEGE LEVEL CHECKING WHEN LOADING THE SS REGISTER

Privilege level checking also occurs when the SS register is loaded with the segment selector for a stack segment. Here all privilege levels related to the stack segment must match the CPL; that is, the CPL, the RPL of the stack-segment selector, and the DPL of the stack-segment descriptor must be the same. If the RPL and DPL are not equal to the CPL, a general-protection exception (#GP) is generated.

6.8 PRIVILEGE LEVEL CHECKING WHEN TRANSFERRING PROGRAM CONTROL BETWEEN CODE SEGMENTS

To transfer program control from one code segment to another, the segment selector for the destination code segment must be loaded into the code-segment register (CS). As part of this loading process, the processor examines the segment descriptor for the destination code segment and performs various limit, type, and privilege checks. If these checks are successful, the CS register is loaded, program control is transferred to the new code segment, and program execution begins at the instruction pointed to by the EIP register.

Program control transfers are carried out with the JMP, CALL, RET, SYSENTER, SYSEXIT, SYSCALL, SYSRET, INT *n*, and IRET instructions, as well as by the exception and interrupt mechanisms. Exceptions, interrupts, and the IRET instruction are special cases discussed in Chapter 7, "Interrupt and Exception Handling." This chapter discusses only the JMP, CALL, RET, SYSENTER, SYSEXIT, SYSCALL, and SYSRET instructions.

A JMP or CALL instruction can reference another code segment in any of four ways:

- The target operand contains the segment selector for the target code segment.
- The target operand points to a call-gate descriptor, which contains the segment selector for the target code segment.
- The target operand points to a TSS, which contains the segment selector for the target code segment.
- The target operand points to a task gate, which points to a TSS, which in turn contains the segment selector for the target code segment.

The following sections describe first two types of references. See Section 9.3, "Task Switching," for information on transferring program control through a task gate and/or TSS.

The SYSENTER and SYSEXIT instructions are special instructions for making fast calls to and returns from operating system or executive procedures. These instructions are discussed in Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."

The SYCALL and SYSRET instructions are special instructions for making fast calls to and returns from operating system or executive procedures in 64-bit mode. These instructions are discussed in Section 6.8.8, "Fast System Calls in 64-Bit Mode."

6.8.1 Direct Calls or Jumps to Code Segments

The near forms of the JMP, CALL, and RET instructions transfer program control within the current code segment, so privilege-level checks are not performed. The far forms of the JMP, CALL, and RET instructions transfer control to other code segments, so the processor does perform privilege-level checks.

When transferring program control to another code segment without going through a call gate, the processor examines four kinds of privilege level and type information (see Figure 6-6):

- The CPL. (Here, the CPL is the privilege level of the calling code segment; that is, the code segment that contains the procedure that is making the call or jump.)

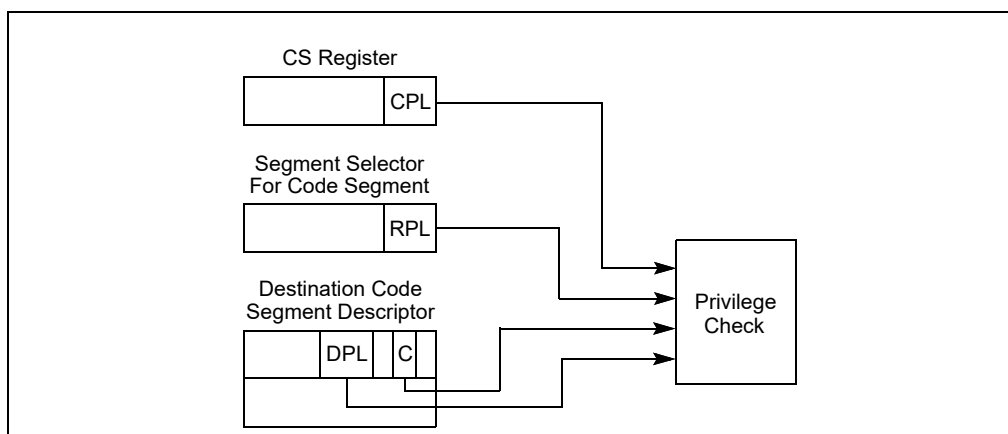


Figure 6-6. Privilege Check for Control Transfer Without Using a Gate

- The DPL of the segment descriptor for the destination code segment that contains the called procedure.
- The RPL of the segment selector of the destination code segment.
- The conforming (C) flag in the segment descriptor for the destination code segment, which determines whether the segment is a conforming (C flag is set) or nonconforming (C flag is clear) code segment. See Section 3.4.5.1, "Code- and Data-Segment Descriptor Types," for more information about this flag.

The rules that the processor uses to check the CPL, RPL, and DPL depends on the setting of the C flag, as described in the following sections.

6.8.1.1 Accessing Nonconforming Code Segments

When accessing nonconforming code segments, the CPL of the calling procedure must be equal to the DPL of the destination code segment; otherwise, the processor generates a general-protection exception (#GP). For example in Figure 6-7:

- Code segment C is a nonconforming code segment. A procedure in code segment A can call a procedure in code segment C (using segment selector C1) because they are at the same privilege level (CPL of code segment A is equal to the DPL of code segment C).
- A procedure in code segment B cannot call a procedure in code segment C (using segment selector C2 or C1) because the two code segments are at different privilege levels.

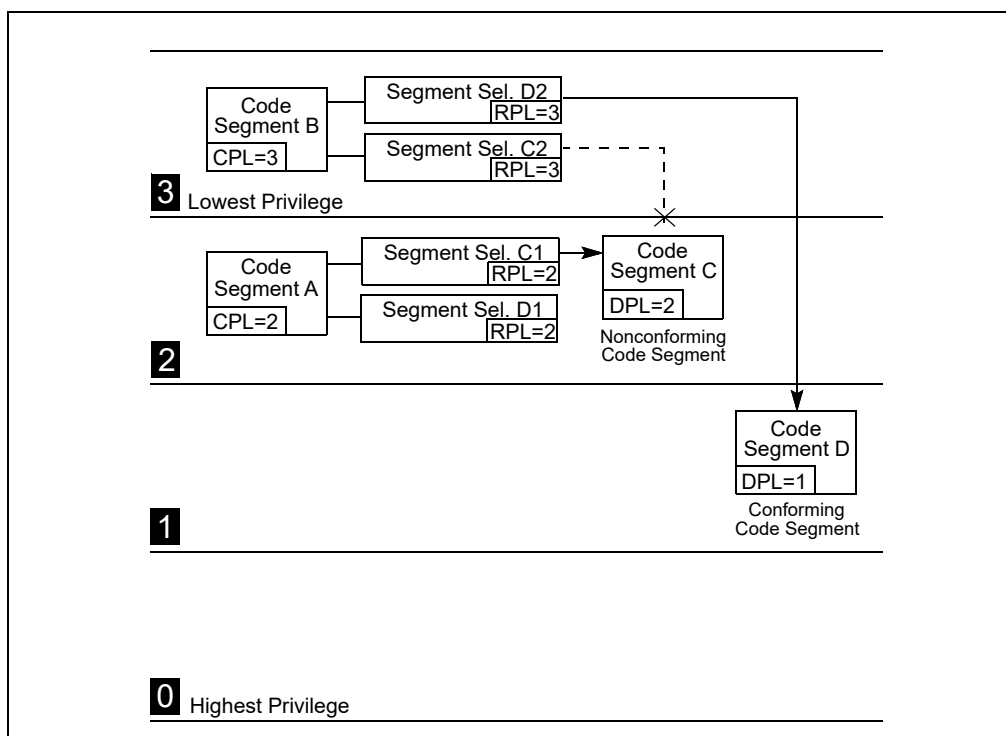


Figure 6-7. Examples of Accessing Conforming and Nonconforming Code Segments From Various Privilege Levels

The RPL of the segment selector that points to a nonconforming code segment has a limited effect on the privilege check. The RPL must be numerically less than or equal to the CPL of the calling procedure for a successful control transfer to occur. So, in the example in Figure 6-7, the RPLs of segment selectors C1 and C2 could legally be set to 0, 1, or 2, but not to 3.

When the segment selector of a nonconforming code segment is loaded into the CS register, the privilege level field is not changed; that is, it remains at the CPL (which is the privilege level of the calling procedure). This is true, even if the RPL of the segment selector is different from the CPL.

6.8.1.2 Accessing Conforming Code Segments

When accessing conforming code segments, the CPL of the calling procedure may be numerically equal to or greater than (less privileged) the DPL of the destination code segment; the processor generates a general-protection exception (#GP) only if the CPL is less than the DPL. (The segment selector RPL for the destination code segment is not checked if the segment is a conforming code segment.)

In the example in Figure 6-7, code segment D is a conforming code segment. Therefore, calling procedures in both code segment A and B can access code segment D (using either segment selector D1 or D2, respectively), because they both have CPLs that are greater than or equal to the DPL of the conforming code segment. **For conforming code segments, the DPL represents the numerically lowest privilege level that a calling procedure may be at to successfully make a call to the code segment.**

(Note that segments selectors D1 and D2 are identical except for their respective RPLs. But since RPLs are not checked when accessing conforming code segments, the two segment selectors are essentially interchangeable.)

When program control is transferred to a conforming code segment, the CPL does not change, even if the DPL of the destination code segment is less than the CPL. This situation is the only one where the CPL may be different from the DPL of the current code segment. Also, since the CPL does not change, no stack switch occurs.

Conforming segments are used for code modules such as math libraries and exception handlers, which support applications but do not require access to protected system facilities. These modules are part of the operating system or executive software, but they can be executed at numerically higher privilege levels (less privileged levels). Keeping the CPL at the level of a calling code segment when switching to a conforming code segment

prevents an application program from accessing nonconforming code segments while at the privilege level (DPL) of a conforming code segment and thus prevents it from accessing more privileged data.

Most code segments are nonconforming. For these segments, program control can be transferred only to code segments at the same level of privilege, unless the transfer is carried out through a call gate, as described in the following sections.

6.8.2 Gate Descriptors

To provide controlled access to code segments with different privilege levels, the processor provides special set of descriptors called gate descriptors. There are four kinds of gate descriptors:

- Call gates
- Trap gates
- Interrupt gates
- Task gates

Task gates are used for task switching and are discussed in Chapter 9, “Task Management.” Trap and interrupt gates are special kinds of call gates used for calling exception and interrupt handlers. The are described in Chapter 7, “Interrupt and Exception Handling.” This chapter is concerned only with call gates.

6.8.3 Call Gates

Call gates facilitate controlled transfers of program control between different privilege levels. They are typically used only in operating systems or executives that use the privilege-level protection mechanism. Call gates are also useful for transferring program control between 16-bit and 32-bit code segments, as described in Section 23.4, “Transferring Control Among Mixed-Size Code Segments.”

Figure 6-8 shows the format of a call-gate descriptor. A call-gate descriptor may reside in the GDT or in an LDT, but not in the interrupt descriptor table (IDT). It performs six functions:

- It specifies the code segment to be accessed.
- It defines an entry point for a procedure in the specified code segment.
- It specifies the privilege level required for a caller trying to access the procedure.

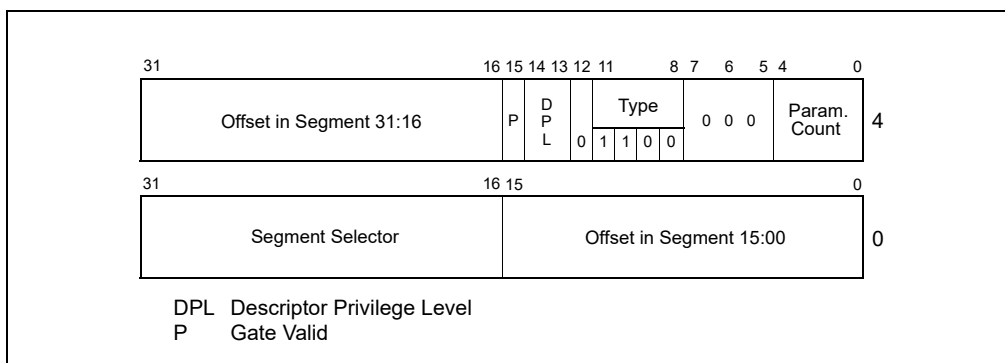


Figure 6-8. Call-Gate Descriptor

- If a stack switch occurs, it specifies the number of optional parameters to be copied between stacks.
- It defines the size of values to be pushed onto the target stack: 16-bit gates force 16-bit pushes and 32-bit gates force 32-bit pushes.
- It specifies whether the call-gate descriptor is valid.

The segment selector field in a call gate specifies the code segment to be accessed. The offset field specifies the entry point in the code segment. This entry point is generally to the first instruction of a specific procedure. The DPL field indicates the privilege level of the call gate, which in turn is the privilege level required to access the selected

procedure through the gate. The P flag indicates whether the call-gate descriptor is valid. (The presence of the code segment to which the gate points is indicated by the P flag in the code segment’s descriptor.) The parameter count field indicates the number of parameters to copy from the calling procedures stack to the new stack if a stack switch occurs (see Section 6.8.5, “Stack Switching”). The parameter count specifies the number of words for 16-bit call gates and doublewords for 32-bit call gates.

Note that the P flag in a gate descriptor is normally always set to 1. If it is set to 0, a not present (#NP) exception is generated when a program attempts to access the descriptor. The operating system can use the P flag for special purposes. For example, it could be used to track the number of times the gate is used. Here, the P flag is initially set to 0 causing a trap to the not-present exception handler. The exception handler then increments a counter and sets the P flag to 1, so that on returning from the handler, the gate descriptor will be valid.

6.8.3.1 IA-32e Mode Call Gates

Call-gate descriptors in 32-bit mode provide a 32-bit offset for the instruction pointer (EIP); 64-bit extensions double the size of 32-bit mode call gates in order to store 64-bit instruction pointers (RIP). See Figure 6-9:

- The first eight bytes (bytes 7:0) of a 64-bit mode call gate are similar but not identical to legacy 32-bit mode call gates. The parameter-copy-count field has been removed.
- Bytes 11:8 hold the upper 32 bits of the target-segment offset in canonical form. A general-protection exception (#GP) is generated if software attempts to use a call gate with a target offset that is not in canonical form.
- 16-byte descriptors may reside in the same descriptor table with 16-bit and 32-bit descriptors. A type field, used for consistency checking, is defined in bits 12:8 of the 64-bit descriptor’s highest dword (cleared to zero). A general-protection exception (#GP) results if an attempt is made to access the upper half of a 64-bit mode descriptor as a 32-bit mode descriptor.

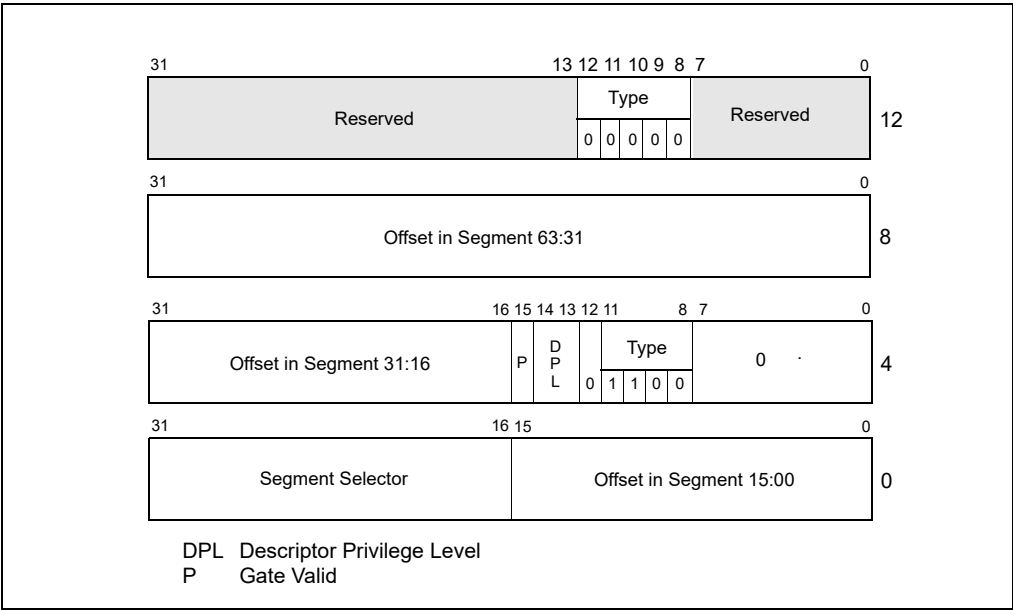


Figure 6-9. Call-Gate Descriptor in IA-32e Mode

- Target code segments referenced by a 64-bit call gate must be 64-bit code segments (CS.L = 1, CS.D = 0). If not, the reference generates a general-protection exception, #GP (CS selector).
- Only 64-bit mode call gates can be referenced in IA-32e mode (64-bit mode and compatibility mode). The legacy 32-bit mode call gate type (0CH) is redefined in IA-32e mode as a 64-bit call-gate type; no 32-bit call-gate type exists in IA-32e mode.

- If a far call references a 16-bit call gate type (04H) in IA-32e mode, a general-protection exception (#GP) is generated.

When a call references a 64-bit mode call gate, actions taken are identical to those taken in 32-bit mode, with the following exceptions:

- Stack pushes are made in eight-byte increments.
- A 64-bit RIP is pushed onto the stack.
- Parameter copying is not performed.

Use a matching far-return instruction size for correct operation (returns from 64-bit calls must be performed with a 64-bit operand-size return to process the stack correctly).

6.8.4 Accessing a Code Segment Through a Call Gate

To access a call gate, a far pointer to the gate is provided as a target operand in a CALL or JMP instruction. The segment selector from this pointer identifies the call gate (see Figure 6-10); the offset from the pointer is required, but not used or checked by the processor. (The offset can be set to any value.)

When the processor has accessed the call gate, it uses the segment selector from the call gate to locate the segment descriptor for the destination code segment. (This segment descriptor can be in the GDT or the LDT.) It then combines the base address from the code-segment descriptor with the offset from the call gate to form the linear address of the procedure entry point in the code segment.

As shown in Figure 6-11, four different privilege levels are used to check the validity of a program control transfer through a call gate:

- The CPL (current privilege level).
- The RPL (requestor privilege level) of the call gate's selector.
- The DPL (descriptor privilege level) of the call gate descriptor.
- The DPL of the segment descriptor of the destination code segment.

The C flag (conforming) in the segment descriptor for the destination code segment is also checked.

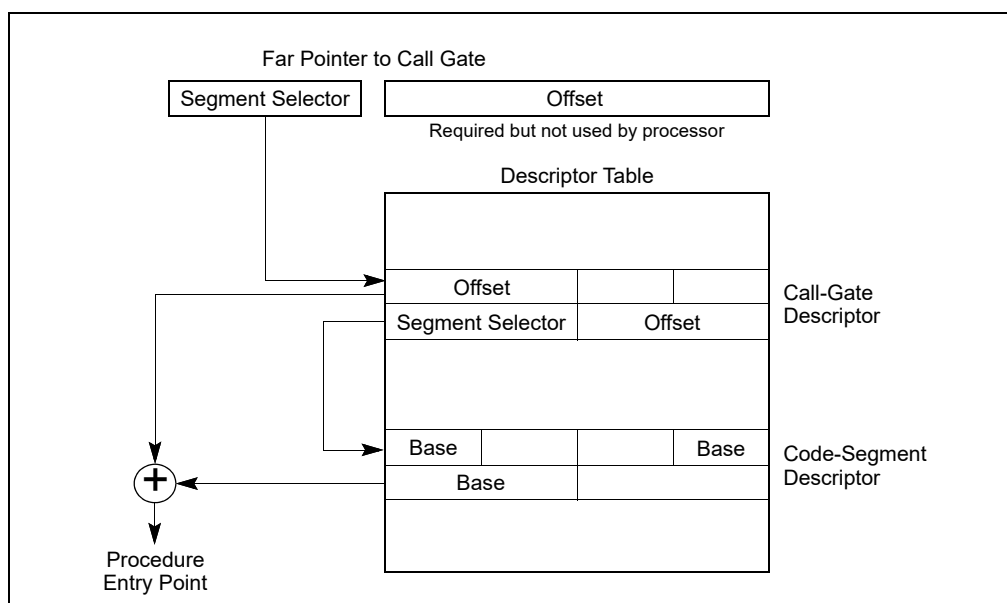


Figure 6-10. Call-Gate Mechanism

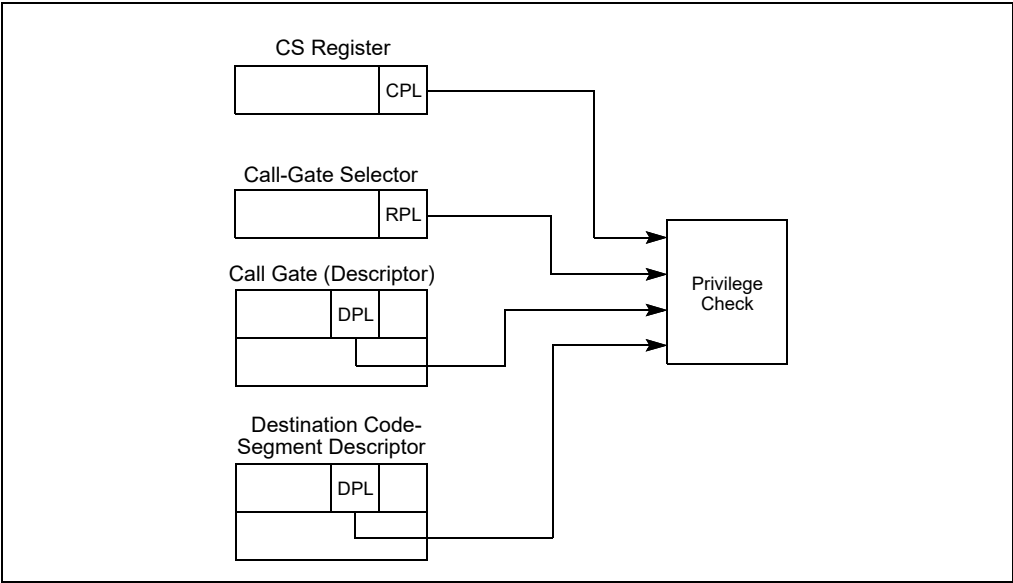


Figure 6-11. Privilege Check for Control Transfer with Call Gate

The privilege checking rules are different depending on whether the control transfer was initiated with a CALL or a JMP instruction, as shown in Table 6-1.

Table 6-1. Privilege Check Rules for Call Gates

Instruction	Privilege Check Rules
CALL	$CPL \leq \text{call gate DPL}$; $RPL \leq \text{call gate DPL}$ Destination conforming code segment $DPL \leq CPL$ Destination nonconforming code segment $DPL \leq CPL$
JMP	$CPL \leq \text{call gate DPL}$; $RPL \leq \text{call gate DPL}$ Destination conforming code segment $DPL \leq CPL$ Destination nonconforming code segment $DPL = CPL$

The DPL field of the call-gate descriptor specifies the numerically highest privilege level from which a calling procedure can access the call gate; that is, to access a call gate, the CPL of a calling procedure must be equal to or less than the DPL of the call gate. For example, in Figure 6-15, call gate A has a DPL of 3. So calling procedures at all CPLs (0 through 3) can access this call gate, which includes calling procedures in code segments A, B, and C. Call gate B has a DPL of 2, so only calling procedures at a CPL of 0, 1, or 2 can access call gate B, which includes calling procedures in code segments B and C. The dotted line shows that a calling procedure in code segment A cannot access call gate B.

The RPL of the segment selector to a call gate must satisfy the same test as the CPL of the calling procedure; that is, the RPL must be less than or equal to the DPL of the call gate. In the example in Figure 6-15, a calling procedure in code segment C can access call gate B using gate selector B2 or B1, but it could not use gate selector B3 to access call gate B.

If the privilege checks between the calling procedure and call gate are successful, the processor then checks the DPL of the code-segment descriptor against the CPL of the calling procedure. Here, the privilege check rules vary between CALL and JMP instructions. Only CALL instructions can use call gates to transfer program control to more privileged (numerically lower privilege level) nonconforming code segments; that is, to nonconforming code segments with a DPL less than the CPL. A JMP instruction can use a call gate only to transfer program control to a nonconforming code segment with a DPL equal to the CPL. CALL and JMP instruction can both transfer program control to a more privileged conforming code segment; that is, to a conforming code segment with a DPL less than or equal to the CPL.

If a call is made to a more privileged (numerically lower privilege level) nonconforming destination code segment, the CPL is lowered to the DPL of the destination code segment and a stack switch occurs (see Section 6.8.5, “Stack Switching”). If a call or jump is made to a more privileged conforming destination code segment, the CPL is not changed and no stack switch occurs.

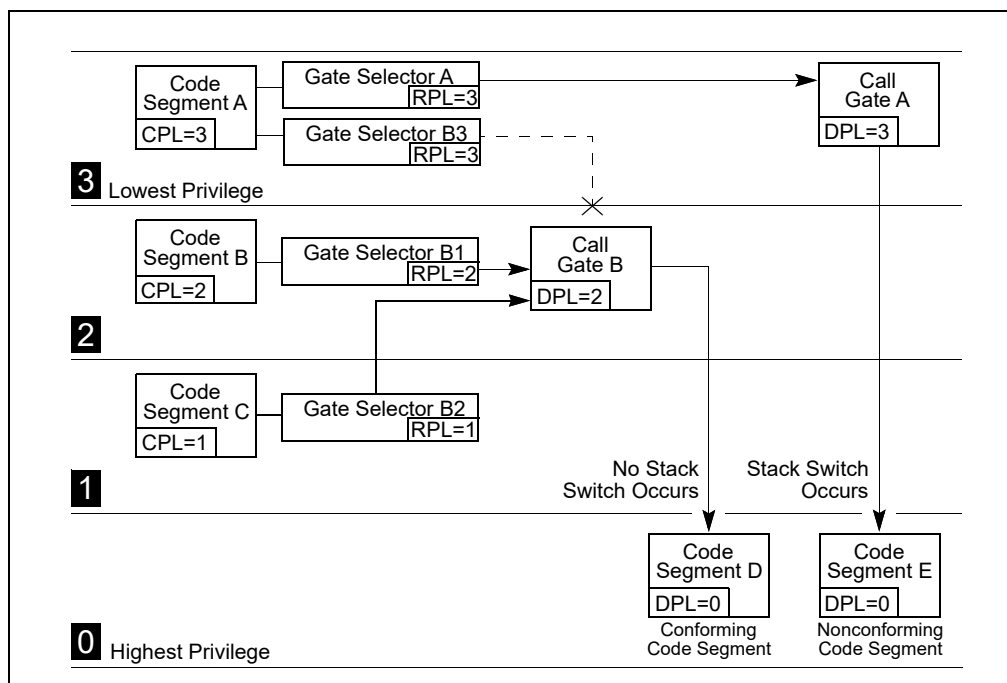


Figure 6-12. Example of Accessing Call Gates At Various Privilege Levels

Call gates allow a single code segment to have procedures that can be accessed at different privilege levels. For example, an operating system located in a code segment may have some services which are intended to be used by both the operating system and application software (such as procedures for handling character I/O). Call gates for these procedures can be set up that allow access at all privilege levels (0 through 3). More privileged call gates (with DPLs of 0 or 1) can then be set up for other operating system services that are intended to be used only by the operating system (such as procedures that initialize device drivers).

6.8.5 Stack Switching

Whenever a call gate is used to transfer program control to a more privileged nonconforming code segment (that is, when the DPL of the nonconforming destination code segment is less than the CPL), the processor automatically switches to the stack for the destination code segment's privilege level. This stack switching is carried out to prevent more privileged procedures from crashing due to insufficient stack space. It also prevents less privileged procedures from interfering (by accident or intent) with more privileged procedures through a shared stack.

Each task must define up to 4 stacks: one for applications code (running at privilege level 3) and one for each of the privilege levels 2, 1, and 0 that are used. (If only two privilege levels are used [3 and 0], then only two stacks must be defined.) Each of these stacks is located in a separate segment and is identified with a segment selector and an offset into the stack segment (a stack pointer).

The segment selector and stack pointer for the privilege level 3 stack is located in the SS and ESP registers, respectively, when privilege-level-3 code is being executed and is automatically stored on the called procedure's stack when a stack switch occurs.

Pointers to the privilege level 0, 1, and 2 stacks are stored in the TSS for the currently running task (see Figure 9-2). Each of these pointers consists of a segment selector and a stack pointer (loaded into the ESP register). These initial pointers are strictly read-only values. The processor does not change them while the task is running. They are used only to create new stacks when calls are made to more privileged levels (numerically lower

privilege levels). These stacks are disposed of when a return is made from the called procedure. The next time the procedure is called, a new stack is created using the initial stack pointer. (The TSS does not specify a stack for privilege level 3 because the processor does not allow a transfer of program control from a procedure running at a CPL of 0, 1, or 2 to a procedure running at a CPL of 3, except on a return.)

The operating system is responsible for creating stacks and stack-segment descriptors for all the privilege levels to be used and for loading initial pointers for these stacks into the TSS. Each stack must be read/write accessible (as specified in the type field of its segment descriptor) and must contain enough space (as specified in the limit field) to hold the following items:

- The contents of the SS, ESP, CS, and EIP registers for the calling procedure.
- The parameters and temporary variables required by the called procedure.
- The EFLAGS register and error code, when implicit calls are made to an exception or interrupt handler.

The stack will need to require enough space to contain many frames of these items, because procedures often call other procedures, and an operating system may support nesting of multiple interrupts. Each stack should be large enough to allow for the worst case nesting scenario at its privilege level.

(If the operating system does not use the processor's multitasking mechanism, it still must create at least one TSS for this stack-related purpose.)

When a procedure call through a call gate results in a change in privilege level, the processor performs the following steps to switch stacks and begin execution of the called procedure at a new privilege level:

1. Uses the DPL of the destination code segment (the new CPL) to select a pointer to the new stack (segment selector and stack pointer) from the TSS.
2. Reads the segment selector and stack pointer for the stack to be switched to from the current TSS. Any limit violations detected while reading the stack-segment selector, stack pointer, or stack-segment descriptor cause an invalid TSS (#TS) exception to be generated.
3. Checks the stack-segment descriptor for the proper privileges and type and generates an invalid TSS (#TS) exception if violations are detected.
4. Temporarily saves the current values of the SS and ESP registers.
5. Loads the segment selector and stack pointer for the new stack in the SS and ESP registers.
6. Pushes the temporarily saved values for the SS and ESP registers (for the calling procedure) onto the new stack (see Figure 6-13).
7. Copies the number of parameter specified in the parameter count field of the call gate from the calling procedure's stack to the new stack. If the count is 0, no parameters are copied.
8. Pushes the return instruction pointer (the current contents of the CS and EIP registers) onto the new stack.
9. Loads the segment selector for the new code segment and the new instruction pointer from the call gate into the CS and EIP registers, respectively, and begins execution of the called procedure.

See the description of the CALL instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a detailed description of the privilege level checks and other protection checks that the processor performs on a far call through a call gate.

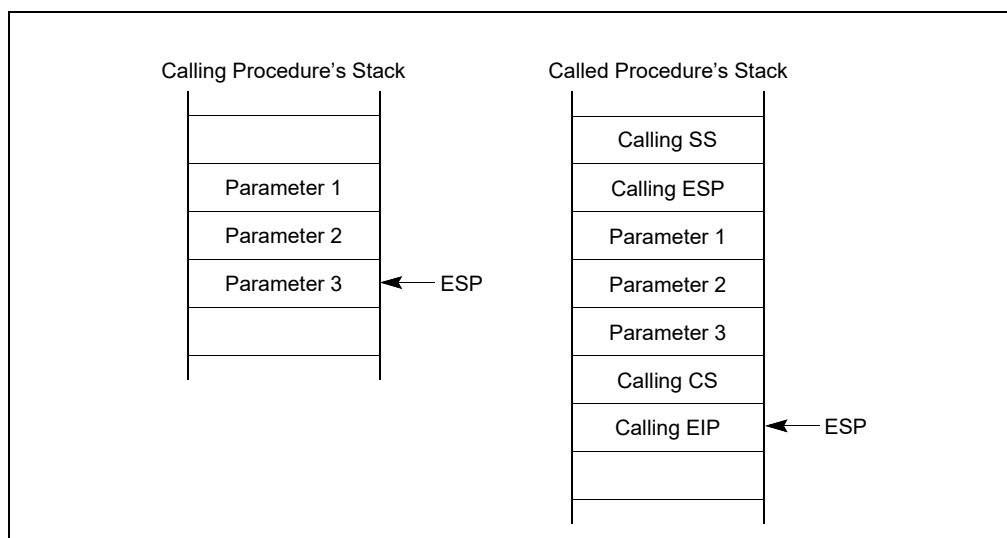


Figure 6-13. Stack Switching During an Interprivilege-Level Call

The parameter count field in a call gate specifies the number of data items (up to 31) that the processor should copy from the calling procedure's stack to the stack of the called procedure. If more than 31 data items need to be passed to the called procedure, one of the parameters can be a pointer to a data structure, or the saved contents of the SS and ESP registers may be used to access parameters in the old stack space. The size of the data items passed to the called procedure depends on the call gate size, as described in Section 6.8.3, "Call Gates."

6.8.5.1 Stack Switching in 64-bit Mode

Although protection-check rules for call gates are unchanged from 32-bit mode, stack-switch changes in 64-bit mode are different.

When stacks are switched as part of a 64-bit mode privilege-level change through a call gate, a new SS (stack segment) descriptor is not loaded; 64-bit mode only loads an inner-level RSP from the TSS. The new SS is forced to NULL and the SS selector's RPL field is forced to the new CPL. The new SS is set to NULL in order to handle nested far transfers (far CALL, INTn, interrupts, and exceptions). The old SS and RSP are saved on the new stack.

On a subsequent far RET, the old SS is popped from the stack and loaded into the SS register. See Table 6-2.

Table 6-2. 64-Bit-Mode Stack Layout After Far CALL with CPL Change

32-bit Mode		ESP	IA-32e mode		RSP
Old SS Selector	+12		+24	Old SS Selector	
Old ESP	+8		+16	Old RSP	
CS Selector	+4		+8	Old CS Selector	
EIP	0		0	RIP	
< 4 Bytes >				< 8 Bytes >	

In 64-bit mode, stack operations resulting from a privilege-level-changing far call or far return are eight-bytes wide and change the RSP by eight. The mode does not support the automatic parameter-copy feature found in 32-bit mode. The call-gate count field is ignored. Software can access the old stack, if necessary, by referencing the old stack-segment selector and stack pointer saved on the new process stack.

In 64-bit mode, far RET is allowed to load a NULL SS under certain conditions. If the target mode is 64-bit mode and the target CPL $\neq$ 3, IRET allows SS to be loaded with a NULL selector. If the called procedure itself is interrupted, the NULL SS is pushed on the stack frame. On the subsequent far RET, the NULL SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

6.8.6 Returning from a Called Procedure

The RET instruction can be used to perform a near return, a far return at the same privilege level, and a far return to a different privilege level. This instruction is intended to execute returns from procedures that were called with a CALL instruction. It does not support returns from a JMP instruction, because the JMP instruction does not save a return instruction pointer on the stack.

A near return only transfers program control within the current code segment; therefore, the processor performs only a limit check. When the processor pops the return instruction pointer from the stack into the EIP register, it checks that the pointer does not exceed the limit of the current code segment.

On a far return at the same privilege level, the processor pops both a segment selector for the code segment being returned to and a return instruction pointer from the stack. Under normal conditions, these pointers should be valid, because they were pushed on the stack by the CALL instruction. However, the processor performs privilege checks to detect situations where the current procedure might have altered the pointer or failed to maintain the stack properly.

A far return that requires a privilege-level change is only allowed when returning to a less privileged level (that is, the DPL of the return code segment is numerically greater than the CPL). The processor uses the RPL field from the CS register value saved for the calling procedure (see Figure 6-13) to determine if a return to a numerically higher privilege level is required. If the RPL is numerically greater (less privileged) than the CPL, a return across privilege levels occurs.

The processor performs the following steps when performing a far return to a calling procedure (see Figures 6-2 and 6-4 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for an illustration of the stack contents prior to and after a return):

1. Checks the RPL field of the saved CS register value to determine if a privilege level change is required on the return.
2. Loads the CS and EIP registers with the values on the called procedure's stack. (Type and privilege level checks are performed on the code-segment descriptor and RPL of the code-segment selector.)
3. (If the RET instruction includes a parameter count operand and the return requires a privilege level change.) Adds the parameter count (in bytes obtained from the RET instruction) to the current ESP register value (after popping the CS and EIP values), to step past the parameters on the called procedure's stack. The resulting value in the ESP register points to the saved SS and ESP values for the calling procedure's stack. (Note that the byte count in the RET instruction must be chosen to match the parameter count in the call gate that the calling procedure referenced when it made the original call multiplied by the size of the parameters.)
4. (If the return requires a privilege level change.) Loads the SS and ESP registers with the saved SS and ESP values and switches back to the calling procedure's stack. The SS and ESP values for the called procedure's stack are discarded. Any limit violations detected while loading the stack-segment selector or stack pointer cause a general-protection exception (#GP) to be generated. The new stack-segment descriptor is also checked for type and privilege violations.
5. (If the RET instruction includes a parameter count operand.) Adds the parameter count (in bytes obtained from the RET instruction) to the current ESP register value, to step past the parameters on the calling procedure's stack. The resulting ESP value is not checked against the limit of the stack segment. If the ESP value is beyond the limit, that fact is not recognized until the next stack operation.
6. (If the return requires a privilege level change.) Checks the contents of the DS, ES, FS, and GS segment registers. If any of these registers refer to segments whose DPL is less than the new CPL (excluding conforming code segments), the segment register is loaded with a null segment selector.

See the description of the RET instruction in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B, for a detailed description of the privilege level checks and other protection checks that the processor performs on a far return.

6.8.7 Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions

The SYSENTER and SYSEXIT instructions were introduced into the IA-32 architecture in the Pentium II processors for the purpose of providing a fast (low overhead) mechanism for calling operating system or executive procedures.

SYSENTER is intended for use by user code running at privilege level 3 to access operating system or executive procedures running at privilege level 0. SYSEXIT is intended for use by privilege level 0 operating system or executive procedures for fast returns to privilege level 3 user code. SYSENTER can be executed from privilege levels 3, 2, 1, or 0; SYSEXIT can only be executed from privilege level 0.

The SYSENTER and SYSEXIT instructions are companion instructions, but they do not constitute a call/return pair. This is because SYSENTER does not save any state information for use by SYSEXIT on a return.

The target instruction and stack pointer for these instructions are not specified through instruction operands. Instead, they are specified through parameters entered in MSRs and general-purpose registers.

For SYSENTER, target fields are generated using the following sources:

- **Target code segment** — Reads this from IA32_SYSENTER_CS.
- **Target instruction** — Reads this from IA32_SYSENTER_EIP.
- **Stack segment** — Computed by adding 8 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Reads this from the IA32_SYSENTER_ESP.

For SYSEXIT, target fields are generated using the following sources:

- **Target code segment** — Computed by adding 16 to the value in the IA32_SYSENTER_CS.
- **Target instruction** — Reads this from EDI.
- **Stack segment** — Computed by adding 24 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Reads this from ECX.

The SYSENTER and SYSEXIT instructions perform “fast” calls and returns because they force the processor into a predefined privilege level 0 state when SYSENTER is executed and into a predefined privilege level 3 state when SYSEXIT is executed. By forcing predefined and consistent processor states, the number of privilege checks ordinarily required to perform a far call to another privilege levels are greatly reduced. Also, by predefining the target context state in MSRs and general-purpose registers eliminates all memory accesses except when fetching the target code.

Any additional state that needs to be saved to allow a return to the calling procedure must be saved explicitly by the calling procedure or be predefined through programming conventions.

6.8.7.1 SYSENTER and SYSEXIT Instructions in IA-32e Mode

For Intel 64 processors, the SYSENTER and SYSEXIT instructions are enhanced to allow fast system calls from user code running at privilege level 3 (in compatibility mode or 64-bit mode) to 64-bit executive procedures running at privilege level 0. IA32_SYSENTER_EIP MSR and IA32_SYSENTER_ESP MSR are expanded to hold 64-bit addresses. If IA-32e mode is inactive, only the lower 32-bit addresses stored in these MSRs are used. The WRMSR instruction ensures that the addresses stored in these MSRs are canonical. Note that, in 64-bit mode, IA32_SYSENTER_CS must not contain a NULL selector.

When SYSENTER transfers control, the following fields are generated and bits set:

- **Target code segment** — Reads non-NUL selector from IA32_SYSENTER_CS.
- **New CS attributes** — CS base = 0, CS limit = FFFFFFFFH.
- **Target instruction** — Reads 64-bit canonical address from IA32_SYSENTER_EIP.
- **Stack segment** — Computed by adding 8 to the value from IA32_SYSENTER_CS.
- **Stack pointer** — Reads 64-bit canonical address from IA32_SYSENTER_ESP.
- **New SS attributes** — SS base = 0, SS limit = FFFFFFFFH.

When the SYSEXIT instruction transfers control to 64-bit mode user code using REX.W, the following fields are generated and bits set:

- **Target code segment** — Computed by adding 32 to the value in IA32_SYSENTER_CS.
- **New CS attributes** — L-bit = 1 (go to 64-bit mode).
- **Target instruction** — Reads 64-bit canonical address in RDX.
- **Stack segment** — Computed by adding 40 to the value of IA32_SYSENTER_CS.

- **Stack pointer** — Update RSP using 64-bit canonical address in RCX.

When SYSEXIT transfers control to compatibility mode user code when the operand size attribute is 32 bits, the following fields are generated and bits set:

- **Target code segment** — Computed by adding 16 to the value in IA32_SYSENTER_CS.
- **New CS attributes** — L-bit = 0 (go to compatibility mode).
- **Target instruction** — Fetch the target instruction from 32-bit address in EDX.
- **Stack segment** — Computed by adding 24 to the value in IA32_SYSENTER_CS.
- **Stack pointer** — Update ESP from 32-bit address in ECX.

6.8.8 Fast System Calls in 64-Bit Mode

The SYSCALL and SYSRET instructions are designed for operating systems that use a flat memory model (segmentation is not used). The instructions, along with SYSENTER and SYSEXIT, are suited for IA-32e mode operation. SYSCALL and SYSRET, however, are not supported in compatibility mode (or in protected mode). Use CPUID to check if SYSCALL and SYSRET are available (CPUID.80000001H:EDX[11] = 1).

SYSCALL is intended for use by user code running at privilege level 3 to access operating system or executive procedures running at privilege level 0. SYSRET is intended for use by privilege level 0 operating system or executive procedures for fast returns to privilege level 3 user code.

Stack pointers for SYSCALL/SYSRET are not specified through model specific registers. The clearing of bits in RFLAGS is programmable rather than fixed. SYSCALL/SYSRET save and restore the RFLAGS register.

For SYSCALL, the processor saves RFLAGS into R11 and the RIP of the next instruction into RCX; it then gets the privilege-level 0 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[47:32].
- **Target instruction pointer** — Reads a 64-bit address from IA32_LSTAR. (The WRMSR instruction ensures that the value of the IA32_LSTAR MSR is canonical.)
- **Stack segment** — Computed by adding 8 to the value in IA32_STAR[47:32].
- **Flags** — The processor sets RFLAGS to the logical-AND of its current value with the complement of the value in the IA32_FMASK MSR.

When SYSRET transfers control to 64-bit mode user code using REX.W, the processor gets the privilege level 3 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[63:48] + 16.
- **Target instruction pointer** — Copies the value in RCX into RIP.
- **Stack segment** — IA32_STAR[63:48] + 8.
- **EFLAGS** — Loaded from R11.

When SYSRET transfers control to 32-bit mode user code using a 32-bit operand size, the processor gets the privilege level 3 target code segment, instruction pointer, stack segment, and flags as follows:

- **Target code segment** — Reads a non-NULL selector from IA32_STAR[63:48].
- **Target instruction pointer** — Copies the value in ECX into EIP.
- **Stack segment** — IA32_STAR[63:48] + 8.
- **EFLAGS** — Loaded from R11.

It is the responsibility of the OS to ensure the descriptors in the GDT/LDT correspond to the selectors loaded by SYSCALL/SYSRET (consistent with the base, limit, and attribute values forced by the instructions).

See Figure 6-14 for the layout of IA32_STAR, IA32_LSTAR, and IA32_FMASK.

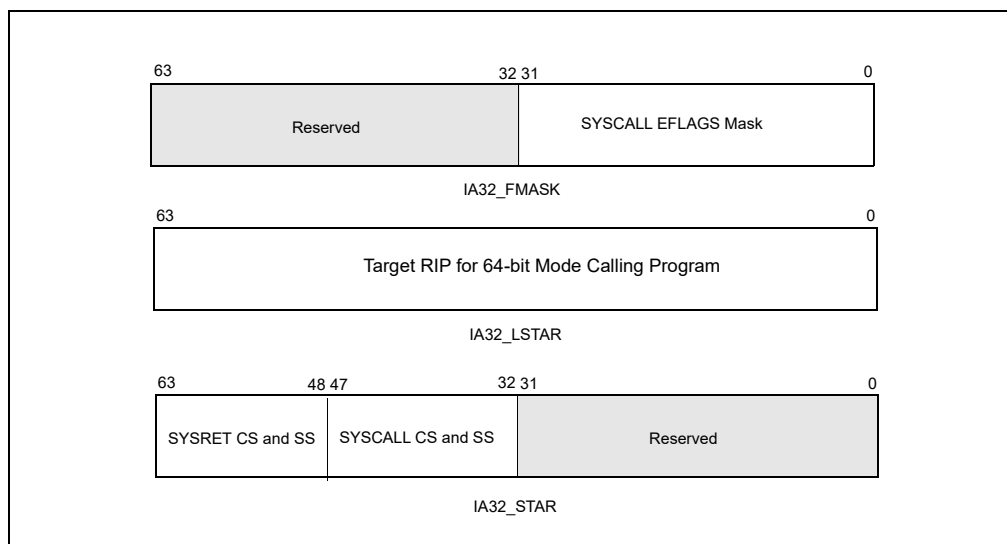


Figure 6-14. MSRs Used by SYSCALL and SYSRET

The SYSCALL instruction does not save the stack pointer, and the SYSRET instruction does not restore it. It is likely that the OS system-call handler will change the stack pointer from the user stack to the OS stack. If so, it is the responsibility of software first to save the user stack pointer. This might be done by user code, prior to executing SYSCALL, or by the OS system-call handler after SYSCALL.

Because the SYSRET instruction does not modify the stack pointer, it is necessary for software to switch back to the user stack. The OS may load the user stack pointer (if it was saved after SYSCALL) before executing SYSRET; alternatively, user code may load the stack pointer (if it was saved before SYSCALL) after receiving control from SYSRET.

If the OS loads the stack pointer before executing SYSRET, it must ensure that the handler of any interrupt or exception delivered between restoring the stack pointer and successful execution of SYSRET is not invoked with the user stack. It can do so using approaches such as the following:

- External interrupts. The OS can prevent an external interrupt from being delivered by clearing EFLAGS.IF before loading the user stack pointer.
- Nonmaskable interrupts (NMIs). The OS can ensure that the NMI handler is invoked with the correct stack by using the interrupt stack table (IST) mechanism for gate 2 (NMI) in the IDT (see Section 7.14.5, “Interrupt Stack Table”).
- General-protection exceptions (#GP). The SYSRET instruction generates #GP(0) if the value of RCX is not canonical. The OS can address this possibility using one or more of the following approaches:
 - Confirming that the value of RCX is canonical before executing SYSRET.
 - Using paging to ensure that the SYSCALL instruction will never save a non-canonical value into RCX.
 - Using the IST mechanism for gate 13 (#GP) in the IDT.

6.9 PRIVILEGED INSTRUCTIONS

Some of the system instructions (called “privileged instructions”) are protected from use by application programs. The privileged instructions control system functions (such as the loading of system registers). They can be executed only when the CPL is 0 (most privileged). If one of these instructions is executed when the CPL is not 0, a general-protection exception (#GP) is generated. The following system instructions are privileged instructions:

- LGDT — Load GDT register.
- LLDT — Load LDT register.

- LTR — Load task register.
- LIDT — Load IDT register.
- MOV (control registers) — Load and store control registers.
- LMSW — Load machine status word.
- CLTS — Clear task-switched flag in register CR0.
- MOV (debug registers) — Load and store debug registers.
- INVD — Invalidate cache, without writeback.
- WBINVD — Invalidate cache, with writeback.
- INVLPG — Invalidate TLB entry.
- HLT — Halt processor.
- RDMSR — Read Model-Specific Registers.
- WRMSR — Write Model-Specific Registers.
- RDPMC — Read Performance-Monitoring Counter.
- RDTSC — Read Time-Stamp Counter.

Some of the privileged instructions are available only in the more recent families of Intel 64 and IA-32 processors (see Section 24.13, “New Instructions In the Pentium and Later IA-32 Processors”).

The PCE and TSD flags in register CR4 (bits 4 and 2, respectively) enable the RDPMC and RDTSC instructions, respectively, to be executed at any CPL.

6.10 POINTER VALIDATION

When operating in protected mode, the processor validates all pointers to enforce protection between segments and maintain isolation between privilege levels. Pointer validation consists of the following checks:

1. Checking access rights to determine if the segment type is compatible with its use.
2. Checking read/write rights.
3. Checking if the pointer offset exceeds the segment limit.
4. Checking if the supplier of the pointer is allowed to access the segment.
5. Checking the offset alignment.

The processor automatically performs first, second, and third checks during instruction execution. Software must explicitly request the fourth check by issuing an ARPL instruction. The fifth check (offset alignment) is performed automatically at privilege level 3 if alignment checking is turned on. Offset alignment does not affect isolation of privilege levels.

6.10.1 Checking Access Rights (LAR Instruction)

When the processor accesses a segment using a far pointer, it performs an access rights check on the segment descriptor pointed to by the far pointer. This check is performed to determine if type and privilege level (DPL) of the segment descriptor are compatible with the operation to be performed. For example, when making a far call in protected mode, the segment-descriptor type must be for a conforming or nonconforming code segment, a call gate, a task gate, or a TSS. Then, if the call is to a nonconforming code segment, the DPL of the code segment must be equal to the CPL, and the RPL of the code segment's segment selector must be less than or equal to the DPL. If type or privilege level are found to be incompatible, the appropriate exception is generated.

To prevent type incompatibility exceptions from being generated, software can check the access rights of a segment descriptor using the LAR (load access rights) instruction. The LAR instruction specifies the segment selector for the segment descriptor whose access rights are to be checked and a destination register. The instruction then performs the following operations:

1. Check that the segment selector is not null.

2. Checks that the segment selector points to a segment descriptor that is within the descriptor table limit (GDT or LDT).
3. Checks that the segment descriptor is a code, data, LDT, call gate, task gate, or TSS segment-descriptor type.
4. If the segment is not a conforming code segment, checks if the segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL).
5. If the privilege level and type checks pass, loads the second doubleword of the segment descriptor into the destination register (masked by the value 00FXFF00H, where X indicates that the corresponding 4 bits are undefined) and sets the ZF flag in the EFLAGS register. If the segment selector is not visible at the current privilege level or is an invalid type for the LAR instruction, the instruction does not modify the destination register and clears the ZF flag.

Once loaded in the destination register, software can perform additional checks on the access rights information.

6.10.2 Checking Read/Write Rights (VERR and VERW Instructions)

When the processor accesses any code or data segment it checks the read/write privileges assigned to the segment to verify that the intended read or write operation is allowed. Software can check read/write rights using the VERR (verify for reading) and VERW (verify for writing) instructions. Both these instructions specify the segment selector for the segment being checked. The instructions then perform the following operations:

1. Check that the segment selector is not null.
2. Checks that the segment selector points to a segment descriptor that is within the descriptor table limit (GDT or LDT).
3. Checks that the segment descriptor is a code or data-segment descriptor type.
4. If the segment is not a conforming code segment, checks if the segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector are less than or equal to the DPL).
5. Checks that the segment is readable (for the VERR instruction) or writable (for the VERW) instruction.

The VERR instruction sets the ZF flag in the EFLAGS register if the segment is visible at the CPL and readable; the VERW sets the ZF flag if the segment is visible and writable. (Code segments are never writable.) The ZF flag is cleared if any of these checks fail.

6.10.3 Checking That the Pointer Offset Is Within Limits (LSL Instruction)

When the processor accesses any segment it performs a limit check to ensure that the offset is within the limit of the segment. Software can perform this limit check using the LSL (load segment limit) instruction. Like the LAR instruction, the LSL instruction specifies the segment selector for the segment descriptor whose limit is to be checked and a destination register. The instruction then performs the following operations:

1. Check that the segment selector is not null.
2. Checks that the segment selector points to a segment descriptor that is within the descriptor table limit (GDT or LDT).
3. Checks that the segment descriptor is a code, data, LDT, or TSS segment-descriptor type.
4. If the segment is not a conforming code segment, checks if the segment descriptor is visible at the CPL (that is, if the CPL and the RPL of the segment selector less than or equal to the DPL).
5. If the privilege level and type checks pass, loads the unscrambled limit (the limit scaled according to the setting of the G flag in the segment descriptor) into the destination register and sets the ZF flag in the EFLAGS register. If the segment selector is not visible at the current privilege level or is an invalid type for the LSL instruction, the instruction does not modify the destination register and clears the ZF flag.

Once loaded in the destination register, software can compare the segment limit with the offset of a pointer.

6.10.4 Checking Caller Access Privileges (ARPL Instruction)

The requestor's privilege level (RPL) field of a segment selector is intended to carry the privilege level of a calling procedure (the calling procedure's CPL) to a called procedure. The called procedure then uses the RPL to determine if access to a segment is allowed. The RPL is said to "weaken" the privilege level of the called procedure to that of the RPL.

Operating-system procedures typically use the RPL to prevent less privileged application programs from accessing data located in more privileged segments. When an operating-system procedure (the called procedure) receives a segment selector from an application program (the calling procedure), it sets the segment selector's RPL to the privilege level of the calling procedure. Then, when the operating system uses the segment selector to access its associated segment, the processor performs privilege checks using the calling procedure's privilege level (stored in the RPL) rather than the numerically lower privilege level (the CPL) of the operating-system procedure. The RPL thus ensures that the operating system does not access a segment on behalf of an application program unless that program itself has access to the segment.

Figure 6-15 shows an example of how the processor uses the RPL field. In this example, an application program (located in code segment A) possesses a segment selector (segment selector D1) that points to a privileged data structure (that is, a data structure located in a data segment D at privilege level 0).

The application program cannot access data segment D, because it does not have sufficient privilege, but the operating system (located in code segment C) can. So, in an attempt to access data segment D, the application program executes a call to the operating system and passes segment selector D1 to the operating system as a parameter on the stack. Before passing the segment selector, the (well behaved) application program sets the RPL of the segment selector to its current privilege level (which in this example is 3). If the operating system attempts to access data segment D using segment selector D1, the processor compares the CPL (which is now 0 following the call), the RPL of segment selector D1, and the DPL of data segment D (which is 0). Since the RPL is greater than the DPL, access to data segment D is denied. The processor's protection mechanism thus protects data segment D from access by the operating system, because application program's privilege level (represented by the RPL of segment selector B) is greater than the DPL of data segment D.

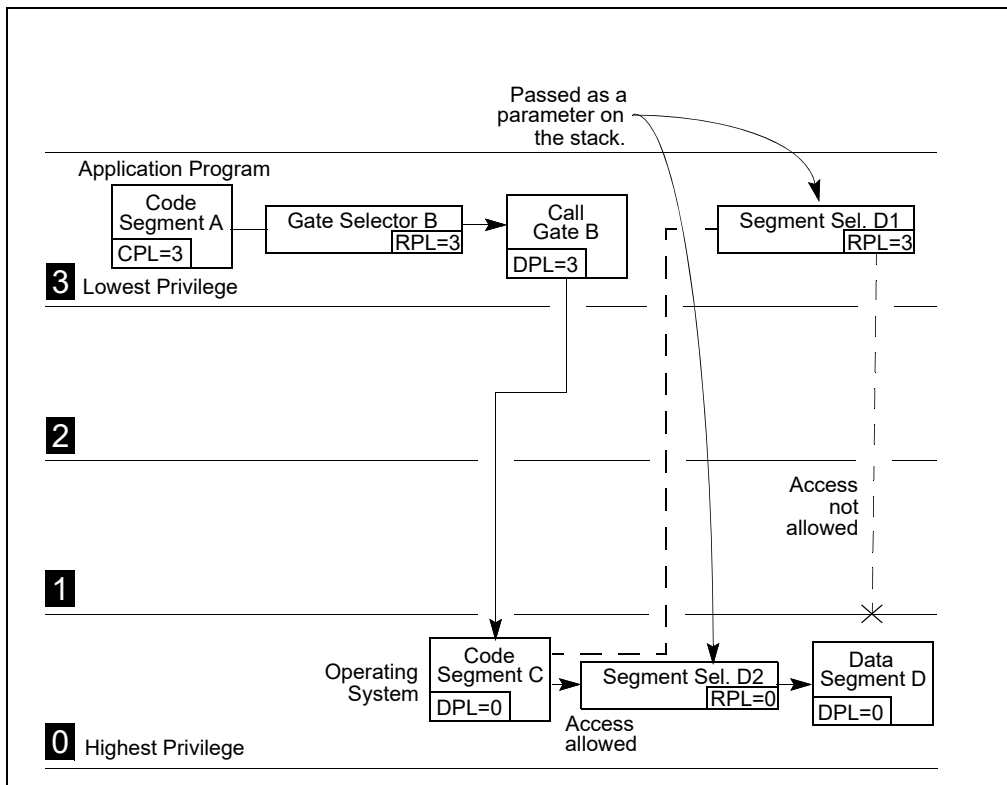


Figure 6-15. Use of RPL to Weaken Privilege Level of Called Procedure

Now assume that instead of setting the RPL of the segment selector to 3, the application program sets the RPL to 0 (segment selector D2). The operating system can now access data segment D, because its CPL and the RPL of segment selector D2 are both equal to the DPL of data segment D.

Because the application program is able to change the RPL of a segment selector to any value, it can potentially use a procedure operating at a numerically lower privilege level to access a protected data structure. This ability to lower the RPL of a segment selector breaches the processor's protection mechanism.

Because a called procedure cannot rely on the calling procedure to set the RPL correctly, operating-system procedures (executing at numerically lower privilege-levels) that receive segment selectors from numerically higher privilege-level procedures need to test the RPL of the segment selector to determine if it is at the appropriate level. The ARPL (adjust requested privilege level) instruction is provided for this purpose. This instruction adjusts the RPL of one segment selector to match that of another segment selector.

The example in Figure 6-15 demonstrates how the ARPL instruction is intended to be used. When the operating-system receives segment selector D2 from the application program, it uses the ARPL instruction to compare the RPL of the segment selector with the privilege level of the application program (represented by the code-segment selector pushed onto the stack). If the RPL is less than application program's privilege level, the ARPL instruction changes the RPL of the segment selector to match the privilege level of the application program (segment selector D1). Using this instruction thus prevents a procedure running at a numerically higher privilege level from accessing numerically lower privilege-level (more privileged) segments by lowering the RPL of a segment selector.

Note that the privilege level of the application program can be determined by reading the RPL field of the segment selector for the application-program's code segment. This segment selector is stored on the stack as part of the call to the operating system. The operating system can copy the segment selector from the stack into a register for use as an operand for the ARPL instruction.

6.10.5 Checking Alignment

When the CPL is 3, alignment of memory references can be checked by setting the AM flag in the CR0 register and the AC flag in the EFLAGS register. Unaligned memory references generate alignment exceptions (#AC). The processor does not generate alignment exceptions when operating at privilege level 0, 1, or 2. See Table 7-7 for a description of the alignment requirements when alignment checking is enabled.

6.11 PAGE-LEVEL PROTECTION

Page-level protection can be used alone or applied to segments. When page-level protection is used with the flat memory model, it allows supervisor code and data (the operating system or executive) to be protected from user code and data (application programs). It also allows pages containing code to be write protected. When the segment- and page-level protection are combined, page-level read/write protection allows more protection granularity within segments.

With page-level protection (as with segment-level protection) each memory reference is checked to verify that protection checks are satisfied. All checks are made before the memory cycle is started, and any violation prevents the cycle from starting and results in a page-fault exception being generated. Because checks are performed in parallel with address translation, there is no performance penalty.

The processor performs two page-level protection checks:

- Restriction of addressable domain (supervisor and user modes).
- Page type (read only or read/write).

Violations of either of these checks results in a page-fault exception being generated. See Chapter 7, "Interrupt 14—Page-Fault Exception (#PF)," for an explanation of the page-fault exception mechanism. This chapter describes the protection violations which lead to page-fault exceptions.

6.11.1 Page-Protection Flags

Protection information for pages is contained in two flags in a paging-structure entry (see Chapter 5): the read/write flag (bit 1) and the user/supervisor flag (bit 2). The protection checks use the flags in all paging structures.

6.11.2 Restricting Addressable Domain

The page-level protection mechanism allows restricting access to pages based on two privilege levels:

- Supervisor mode (U/S flag is 0)—(Most privileged) For the operating system or executive, other system software (such as device drivers), and protected system data (such as page tables).
- User mode (U/S flag is 1)—(Least privileged) For application code and data.

The segment privilege levels map to the page privilege levels as follows. If the processor is currently operating at a CPL of 0, 1, or 2, it is in supervisor mode; if it is operating at a CPL of 3, it is in user mode. When the processor is in supervisor mode, it can access all pages; when in user mode, it can access only user-level pages. (Note that the WP flag in control register CR0 modifies the supervisor permissions, as described in Section 6.11.3, "Page Type.")

Note that to use the page-level protection mechanism, code and data segments must be set up for at least two segment-based privilege levels: level 0 for supervisor code and data segments and level 3 for user code and data segments. (In this model, the stacks are placed in the data segments.) To minimize the use of segments, a flat memory model can be used (see Section 3.2.1, "Basic Flat Model").

Here, the user and supervisor code and data segments all begin at address zero in the linear address space and overlay each other. With this arrangement, operating-system code (running at the supervisor level) and application code (running at the user level) can execute as if there are no segments. Protection between operating-system and application code and data is provided by the processor's page-level protection mechanism.

6.11.3 Page Type

The page-level protection mechanism recognizes two page types:

- Read-only access (R/W flag is 0).
- Read/write access (R/W flag is 1).

When the processor is in supervisor mode and the WP flag in register CR0 is clear (its state following reset initialization), all pages are both readable and writable (write-protection is ignored). When the processor is in user mode, it can write only to user-mode pages that are read/write accessible. User-mode pages which are read/write or read-only are readable; supervisor-mode pages are neither readable nor writable from user mode. A page-fault exception is generated on any attempt to violate the protection rules.

Starting with the P6 family, Intel processors allow user-mode pages to be write-protected against supervisor-mode access. Setting CR0.WP = 1 enables supervisor-mode sensitivity to write protected pages. If CR0.WP = 1, read-only pages are not writable from any privilege level. This supervisor write-protect feature is useful for implementing a "copy-on-write" strategy used by some operating systems, such as UNIX*, for task creation (also called forking or spawning). When a new task is created, it is possible to copy the entire address space of the parent task. This gives the child task a complete, duplicate set of the parent's segments and pages. An alternative copy-on-write strategy saves memory space and time by mapping the child's segments and pages to the same segments and pages used by the parent task. A private copy of a page gets created only when one of the tasks writes to the page. By using the WP flag and marking the shared pages as read-only, the supervisor can detect an attempt to write to a page, and can copy the page at that time.

6.11.4 Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page-directory entry (first-level page table) may differ from those of its page-table entry (second-level page table). The processor checks the protection for a page in both its page-directory and the page-table entries. Table 6-3 shows the protection provided by the possible combinations of protection attributes when the WP flag is clear.

6.11.5 Overrides to Page Protection

The following types of memory accesses are checked as if they are privilege-level 0 accesses, regardless of the CPL at which the processor is currently operating:

- Access to segment descriptors in the GDT, LDT, or IDT.
- Access to an inner-privilege-level stack during an inter-privilege-level call or a call to an exception or interrupt handler, when a change of privilege level occurs.

6.12 COMBINING PAGE AND SEGMENT PROTECTION

When paging is enabled, the processor evaluates segment protection first, then evaluates page protection. If the processor detects a protection violation at either the segment level or the page level, the memory access is not carried out and an exception is generated. If an exception is generated by segmentation, no paging exception is generated.

Page-level protections cannot be used to override segment-level protection. For example, a code segment is by definition not writable. If a code segment is paged, setting the R/W flag for the pages to read-write does not make the pages writable. Attempts to write into the pages will be blocked by segment-level protection checks.

Page-level protection can be used to enhance segment-level protection. For example, if a large read-write data segment is paged, the page-protection mechanism can be used to write-protect individual pages.

Table 6-3. Combined Page-Directory and Page-Table Protection

Page-Directory Entry		Page-Table Entry		Combined Effect	
Privilege	Access Type	Privilege	Access Type	Privilege	Access Type
User	Read-Only	User	Read-Only	User	Read-Only
User	Read-Only	User	Read-Write	User	Read-Only
User	Read-Write	User	Read-Only	User	Read-Only
User	Read-Write	User	Read-Write	User	Read/Write
User	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	User	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write

NOTE:

- * If CRO.WP = 1, access type is determined by the R/W flags of the page-directory and page-table entries. If CRO.WP = 0, supervisor privilege permits read-write access.

6.13 PAGE-LEVEL PROTECTION AND EXECUTE-DISABLE BIT

In addition to page-level protection offered by the U/S and R/W flags, paging structures used with PAE paging, 4-level paging,¹ and 5-level paging provide the execute-disable bit (see Chapter 5, “Paging”). This bit offers additional protection for data pages.

An Intel 64 or IA-32 processor with the execute-disable bit capability can prevent data pages from being used by malicious software to execute code. This capability is provided in:

- 32-bit protected mode with PAE enabled.
- IA-32e mode.

While the execute-disable bit capability does not introduce new instructions, it does require operating systems to use a PAE-enabled environment and establish a page-granular protection policy for memory pages.

If the execute-disable bit of a memory page is set, that page can be used only as data. An attempt to execute code from a memory page with the execute-disable bit set causes a page-fault exception.

The execute-disable capability is not supported with 32-bit paging. Existing page-level protection mechanisms (see Section 6.11, “Page-Level Protection”) continue to apply to memory pages independent of the execute-disable setting.

6.13.1 Detecting and Enabling the Execute-Disable Capability

Software can detect the presence of the execute-disable capability using the CPUID instruction. CPUID.80000001H:EDX.NX[20] = 1 indicates the capability is available.

- If the capability is available, software can enable it by setting IA32_EFER.NXE[bit 11] to 1. IA32_EFER is available if CPUID.80000001H:EDX[20 or 29] = 1.

If the execute-disable capability is not available, a write to set IA32_EFER.NXE produces a #GP exception. See Table 6-4.

Table 6-4. Extended Feature Enable MSR (IA32_EFER)

63:12	11	10	9	8	7:1	0
Reserved	Execute-disable bit enable (NXE)	IA-32e mode active (LMA)	Reserved	IA-32e mode enable (LME)	Reserved	SysCall enable (SCE)

6.13.2 Execute-Disable Page Protection

- The execute-disable bit in the paging structures enhances page protection for data pages. Instructions cannot be fetched from a memory page if IA32_EFER.NXE = 1 and the execute-disable bit is set in any of the paging-structure entries used to map the page. Table 6-5 lists the valid usage of a page in relation to the value of execute-disable bit (bit 63) of the corresponding entry in each level of the paging structures. Execute-disable protection can be activated using the execute-disable bit at any level of the paging structure, irrespective of the corresponding entry in other levels. When execute-disable protection is not activated, the page can be used as code or data.

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

Table 6-5. Page Level Protection Matrix with Execute-Disable Bit Capability with 4-Level Paging

Execute Disable Bit Value (Bit 63)				Valid Usage
PML4	PDP	PDE	PTE	
Bit 63 = 1	*	*	*	Data
*	Bit 63 = 1	*	*	Data
*	*	Bit 63 = 1	*	Data
*	*	*	Bit 63 = 1	Data
Bit 63 = 0	Bit 63 = 0	Bit 63 = 0	Bit 63 = 0	Data/Code

NOTES:

* Value not checked.

In legacy PAE-enabled mode, Table 6-6 and Table 6-7 show the effect of setting the execute-disable bit for code and data pages.

Table 6-6. 4-KByte Page Level Protection Matrix with Execute-Disable Bit Capability with PAE Paging

Execute Disable Bit Value (Bit 63)		Valid Usage
PDE	PTE	
Bit 63 = 1	*	Data
*	Bit 63 = 1	Data
Bit 63 = 0	Bit 63 = 0	Data/Code

NOTE:

* Value not checked.

Table 6-7. 2-MByte Page Level Protection with Execute-Disable Bit Capability with PAE Paging

Execute Disable Bit Value (Bit 63)	Valid Usage
PDE	
Bit 63 = 1	Data
Bit 63 = 0	Data/Code

6.13.3 Reserved Bit Checking

The processor enforces reserved bit checking in paging data structure entries. The bits being checked varies with paging mode and may vary with the size of physical address space.

Table 6-8 shows the reserved bits that are checked when the execute disable bit capability is enabled (CR4.PAE = 1 and IA32_EFER.NXE = 1). Table 6-8 and Table 6-9 show the following paging modes:

- Non-PAE 4-KByte paging: 4-KByte-page only paging (CR4.PAE = 0, CR4.PSE = 0).
- PSE36: 4-KByte and 4-MByte pages (CR4.PAE = 0, CR4.PSE = 1).
- PAE: 4-KByte and 2-MByte pages (CR4.PAE = 1).

Table 6-8. Page Level Protection Matrix with Execute-Disable Bit Capability Enabled

Mode	Paging Mode	Check Bits
32-bit	4-KByte paging (non-PAE)	No reserved bits checked
	PSE36 - PDE, 4-MByte page	Bit 21
	PSE36 - PDE, 4-KByte page	No reserved bits checked
	PSE36 - PTE	No reserved bits checked
	PAE - PDP table entry	Bits 63:MAXPHYADDR, 8:5, and 2:1
	PAE - PDE, 2-MByte page	Bits 62:MAXPHYADDR and 20:13
	PAE - PDE, 4-KByte page	Bits 62:MAXPHYADDR
	PAE - PTE	Bits 62:MAXPHYADDR
64-bit	PML5E	Bits 51:MAXPHYADDR
	PML4E	Bits 51:MAXPHYADDR
	PDPTE	Bits 51:MAXPHYADDR
	PDE, 2-MByte page	Bits 51:MAXPHYADDR and 20:13
	PDE, 4-KByte page	Bits 51:MAXPHYADDR
	PTE	Bits 51:MAXPHYADDR

The reserved bit checking depends on the physical-address width supported by the processor. This width is denoted MAXPHYADDR and is enumerated in CPUID.80000008H:EAX[7:0].¹

If execute disable bit capability is not enabled or not available, reserved bit checking in 64-bit mode includes bit 63 and additional bits. This and reserved bit checking for legacy 32-bit paging modes are shown in Table 6-9.

Table 6-9. Reserved Bit Checking with Execute-Disable Bit Capability Not Enabled

Mode	Paging Mode	Check Bits
32-bit	KByte paging (non-PAE)	No reserved bits checked
	PSE36 - PDE, 4-MByte page	Bit 21
	PSE36 - PDE, 4-KByte page	No reserved bits checked
	PSE36 - PTE	No reserved bits checked
	PAE - PDP table entry	Bits 63:MAXPHYADDR, 8:5, and 2:1
	PAE - PDE, 2-MByte page	Bits 63:MAXPHYADDR and 20:13
	PAE - PDE, 4-KByte page	Bits 63:MAXPHYADDR
	PAE - PTE	Bits 63:MAXPHYADDR
64-bit	PML5E	Bit 63, bits 51:MAXPHYADDR
	PML4E	Bit 63, bits 51:MAXPHYADDR
	PDPTE	Bit 63, bits 51:MAXPHYADDR
	PDE, 2-MByte page	Bit 63, bits 51:MAXPHYADDR and 20:13
	PDE, 4-KByte page	Bit 63, bits 51:MAXPHYADDR
	PTE	Bit 63, bits 51:MAXPHYADDR

1. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.

6.13.4 Exception Handling

When execute disable bit capability is enabled (`IA32_EFER.NXE = 1`), conditions for a page fault to occur include the same conditions that apply to an Intel 64 or IA-32 processor without execute disable bit capability plus the following new condition: an instruction fetch to a linear address that translates to physical address in a memory page that has the execute-disable bit set.

An Execute Disable Bit page fault can occur at all privilege levels. It can occur on any instruction fetch, including (but not limited to): near branches, far branches, `CALL/RET/INT/IRET` execution, sequential instruction fetches, and task switches. The execute-disable bit in the page translation mechanism is checked only when:

- `IA32_EFER.NXE = 1`.
- The instruction translation look-aside buffer (ITLB) is loaded with a page that is not already present in the ITLB.

12. Updates to Chapter 7, Volume 3A

Change bars and violet text show changes to Chapter 7 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- In Section 7.15, "Exception and Interrupt Reference", added UDB instruction in Interrupt 6—Invalid Opcode Exception (#UD).

CHAPTER 7

INTERRUPT AND EXCEPTION HANDLING

This chapter describes the interrupt and exception-handling mechanism when operating in protected mode on an Intel 64 or IA-32 processor. Most of the information provided here also applies to interrupt and exception mechanisms used in real-address, virtual-8086 mode, and 64-bit mode.

Chapter 22, “8086 Emulation,” describes information specific to interrupt and exception mechanisms in real-address and virtual-8086 mode. Section 7.14, “Exception and Interrupt Handling in 64-bit Mode,” describes information specific to interrupt and exception mechanisms in IA-32e mode and 64-bit sub-mode.

7.1 INTERRUPT AND EXCEPTION OVERVIEW

Interrupts and exceptions are events that indicate that a condition exists somewhere in the system, the processor, or within the currently executing program or task that requires the attention of a processor. They typically result in a forced transfer of execution from the currently running program or task to a special software routine or task called an interrupt handler or an exception handler. The action taken by a processor in response to an interrupt or exception is referred to as servicing or handling the interrupt or exception.

Interrupts occur at random times during the execution of a program, in response to signals from hardware. System hardware uses interrupts to handle events external to the processor, such as requests to service peripheral devices. Software can also generate interrupts by executing the `INT n` instruction.

Exceptions occur when the processor detects an error condition while executing an instruction, such as division by zero. The processor detects a variety of error conditions including protection violations, page faults, and internal machine faults. The machine-check architecture of the Pentium 4, Intel Xeon, P6 family, and Pentium processors also permits a machine-check exception to be generated when internal hardware errors and bus errors are detected.

When an interrupt is received or an exception is detected, the currently running procedure or task is suspended while the processor executes an interrupt or exception handler. When execution of the handler is complete, the processor resumes execution of the interrupted procedure or task. The resumption of the interrupted procedure or task happens without loss of program continuity, unless recovery from an exception was not possible or an interrupt caused the currently running program to be terminated.

This chapter describes the processor’s interrupt and exception-handling mechanism, when operating in protected mode. A description of the exceptions and the conditions that cause them to be generated is given at the end of this chapter.

7.2 EXCEPTION AND INTERRUPT VECTORS

To aid in handling exceptions and interrupts, each architecturally defined exception and each interrupt condition requiring special handling by the processor is assigned a unique identification number, called a vector number. The processor uses the vector number assigned to an exception or interrupt as an index into the interrupt descriptor table (IDT). The table provides the entry point to an exception or interrupt handler (see Section 7.10, “Interrupt Descriptor Table (IDT)”).

The allowable range for vector numbers is 0 to 255. Vector numbers in the range 0 through 31 are reserved by the Intel 64 and IA-32 architectures for architecture-defined exceptions and interrupts. Not all of the vector numbers in this range have a currently defined function. The unassigned vector numbers in this range are reserved. Do not use the reserved vector numbers.

Vector numbers in the range 32 to 255 are designated as user-defined interrupts and are not reserved by the Intel 64 and IA-32 architecture. These interrupts are generally assigned to external I/O devices to enable those devices to send interrupts to the processor through one of the external hardware interrupt mechanisms (see Section 7.3, “Sources of Interrupts”).

Table 7-1 shows vector number assignments for architecturally defined exceptions and for the NMI interrupt. This table gives the exception type (see Section 7.5, “Exception Classifications”) and indicates whether an error code is saved on the stack for the exception. The source of each predefined exception and the NMI interrupt is also given.

7.3 SOURCES OF INTERRUPTS

The processor receives interrupts from two sources:

- External (hardware generated) interrupts.
- Software-generated interrupts.

7.3.1 External Interrupts

External interrupts are received through pins on the processor or through the local APIC. The primary interrupt pins on Pentium 4, Intel Xeon, P6 family, and Pentium processors are the LINT[1:0] pins, which are connected to the local APIC (see Chapter 12, “Advanced Programmable Interrupt Controller (APIC)”). When the local APIC is enabled, the LINT[1:0] pins can be programmed through the APIC’s local vector table (LVT) to be associated with any of the processor’s exception or interrupt vectors.

When the local APIC is global/hardware disabled, these pins are configured as INTR and NMI pins, respectively. Asserting the INTR pin signals the processor that an external interrupt has occurred. The processor reads from the system bus the interrupt vector number provided by an external interrupt controller, such as an 8259A (see Section 7.2, “Exception and Interrupt Vectors”). Asserting the NMI pin signals a non-maskable interrupt (NMI), which is assigned to interrupt vector 2.

Table 7-1. Protected-Mode Exceptions and Interrupts

Vector	Mnemonic	Description	Type	Error Code	Source
0	#DE	Divide Error	Fault	No	DIV and IDIV instructions.
1	#DB	Debug Exception	Fault/ Trap	No	Instruction, data, and I/O breakpoints; single-step; and others.
2	—	NMI Interrupt	Interrupt	No	Nonmaskable external interrupt.
3	#BP	Breakpoint	Trap	No	INT3 instruction.
4	#OF	Overflow	Trap	No	INTO instruction.
5	#BR	BOUND Range Exceeded	Fault	No	BOUND instruction.
6	#UD	Invalid Opcode (Undefined Opcode)	Fault	No	UD instruction or reserved opcode.
7	#NM	Device Not Available (No Math Coprocessor)	Fault	No	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Abort	Yes (zero)	Any instruction that can generate an exception, an NMI, or an INTR.
9		Coprocessor Segment Overrun (reserved)	Fault	No	Floating-point instruction. ¹
10	#TS	Invalid TSS	Fault	Yes	Task switch or TSS access.
11	#NP	Segment Not Present	Fault	Yes	Loading segment registers or accessing system segments.
12	#SS	Stack-Segment Fault	Fault	Yes	Stack operations and SS register loads.
13	#GP	General Protection	Fault	Yes	Any memory reference and other protection checks.
14	#PF	Page Fault	Fault	Yes	Any memory reference.

Table 7-1. Protected-Mode Exceptions and Interrupts (Contd.)

Vector	Mnemonic	Description	Type	Error Code	Source
15	—	(Intel reserved. Do not use.)		No	
16	#MF	x87 FPU Floating-Point Error (Math Fault)	Fault	No	x87 FPU floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Fault	Yes	Any data reference in memory. ²
18	#MC	Machine Check	Abort	No	Error codes (if any) and source are model dependent. ³
19	#XM	SIMD Floating-Point Exception	Fault	No	SSE/SSE2/SSE3 floating-point instructions ⁴
20	#VE	Virtualization Exception	Fault	No	EPT violations ⁵
21	#CP	Control Protection Exception	Fault	Yes	RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump.
22-31	—	Intel reserved. Do not use.			
32-255	—	User Defined (Non-reserved) Interrupts	Interrupt		External interrupt or INT <i>n</i> instruction.

NOTES:

1. Processors after the Intel386 processor do not generate this exception.
2. This exception was introduced in the Intel486 processor.
3. This exception was introduced in the Pentium processor and enhanced in the P6 family processors.
4. This exception was introduced in the Pentium III processor.
5. This exception can occur only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.

The processor’s local APIC is normally connected to a system-based I/O APIC. Here, external interrupts received at the I/O APIC’s pins can be directed to the local APIC through the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors). The I/O APIC determines the vector number of the interrupt and sends this number to the local APIC. When a system contains multiple processors, processors can also send interrupts to one another by means of the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors).

The LINT[1:0] pins are not available on the Intel486 processor and earlier Pentium processors that do not contain an on-chip local APIC. These processors have dedicated NMI and INTR pins. With these processors, external interrupts are typically generated by a system-based interrupt controller (8259A), with the interrupts being signaled through the INTR pin.

Note that several other pins on the processor can cause a processor interrupt to occur. However, these interrupts are not handled by the interrupt and exception mechanism described in this chapter. These pins include the RESET#, FLUSH#, STPCLK#, SMI#, R/S#, and INIT# pins. Whether they are included on a particular processor is implementation dependent. Pin functions are described in the data books for the individual processors. The SMI# pin is described in Chapter 33, “System Management Mode.”

7.3.2 Maskable Hardware Interrupts

Any external interrupt that is delivered to the processor by means of the INTR pin or through the local APIC is called a maskable hardware interrupt. Maskable hardware interrupts that can be delivered through the INTR pin include all IA-32 architecture defined interrupt vectors from 0 through 255; those that can be delivered through the local APIC include interrupt vectors 16 through 255.

The IF flag in the EFLAGS register permits all maskable hardware interrupts to be masked as a group (see Section 7.8.1, “Masking Maskable Hardware Interrupts”). Note that when interrupts 0 through 15 are delivered through the local APIC, the APIC indicates the receipt of an illegal vector.

7.3.3 Software-Generated Interrupts

The `INT n` instruction permits interrupts to be generated from within software by supplying an interrupt vector number as an operand. For example, the `INT 35` instruction forces an implicit call to the interrupt handler for interrupt 35.

Any of the interrupt vectors from 0 to 255 can be used as a parameter in this instruction. If the processor’s predefined NMI vector is used, however, the response of the processor will not be the same as it would be from an NMI interrupt generated in the normal manner. If vector number 2 (the NMI vector) is used in this instruction, the NMI interrupt handler is called, but the processor’s NMI-handling hardware is not activated.

Interrupts generated in software with the `INT n` instruction cannot be masked by the IF flag in the EFLAGS register.

7.4 SOURCES OF EXCEPTIONS

The processor receives exceptions from three sources:

- Processor-detected program-error exceptions.
- Software-generated exceptions.
- Machine-check exceptions.

7.4.1 Program-Error Exceptions

The processor generates one or more exceptions when it detects program errors during the execution in an application program or the operating system or executive. Intel 64 and IA-32 architectures define a vector number for each processor-detectable exception. Exceptions are classified as **faults**, **traps**, and **aborts** (see Section 7.5, “Exception Classifications”).

7.4.2 Software-Generated Exceptions

The `INTO`, `INT1`, `INT3`, and `BOUND` instructions permit exceptions to be generated in software. These instructions allow checks for exception conditions to be performed at points in the instruction stream. For example, `INT3` causes a breakpoint exception to be generated.

The `INT n` instruction can be used to emulate exceptions in software; but there is a limitation.¹ If `INT n` provides a vector for one of the architecturally-defined exceptions, the processor generates an interrupt to the correct vector (to access the exception handler) but does not push an error code on the stack. This is true even if the associated hardware-generated exception normally produces an error code. The exception handler will still attempt to pop an error code from the stack while handling the exception. Because no error code was pushed, the handler will pop off and discard the EIP instead (in place of the missing error code). This sends the return to the wrong location.

7.4.3 Machine-Check Exceptions

The P6 family and Pentium processors provide both internal and external machine-check mechanisms for checking the operation of the internal chip hardware and bus transactions. These mechanisms are implementation dependent. When a machine-check error is detected, the processor signals a machine-check exception (vector 18) and returns an error code.

1. The `INT n` instruction has opcode `CD` following by an immediate byte encoding the value of *n*. In contrast, `INT1` has opcode `F1` and `INT3` has opcode `CC`.

See Chapter 7, “Interrupt 18—Machine-Check Exception (#MC),” and Chapter 17, “Machine-Check Architecture,” for more information about the machine-check mechanism.

7.5 EXCEPTION CLASSIFICATIONS

Exceptions are classified as **faults**, **traps**, or **aborts** depending on the way they are reported and whether the instruction that caused the exception can be restarted without loss of program or task continuity.

- **Faults** — A fault is an exception that can generally be corrected and that, once corrected, allows the program to be restarted with no loss of continuity. When a fault is reported, the processor restores the machine state to the state prior to the beginning of execution of the faulting instruction. The return address (saved contents of the CS and EIP registers) for the fault handler points to the faulting instruction, rather than to the instruction following the faulting instruction.
- **Traps** — A trap is an exception that is reported immediately following the execution of the trapping instruction. Traps allow execution of a program or task to be continued without loss of program continuity. The return address for the trap handler points to the instruction to be executed after the trapping instruction.
- **Aborts** — An abort is an exception that does not always report the precise location of the instruction causing the exception and does not allow a restart of the program or task that caused the exception. Aborts are used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables.

NOTE

One exception subset normally reported as a fault is not restartable. Such exceptions result in loss of some processor state. For example, executing a POPAD instruction where the stack frame crosses over the end of the stack segment causes a fault to be reported. In this situation, the exception handler sees that the instruction pointer (CS:EIP) has been restored as if the POPAD instruction had not been executed. However, internal processor state (the general-purpose registers) will have been modified. Such cases are considered programming errors. An application causing this class of exceptions should be terminated by the operating system.

7.6 PROGRAM OR TASK RESTART

To allow the restarting of program or task following the handling of an exception or an interrupt, all exceptions (except aborts) are guaranteed to report exceptions on an instruction boundary. All interrupts are guaranteed to be taken on an instruction boundary.

For fault-class exceptions, the return instruction pointer (saved when the processor generates an exception) points to the faulting instruction. So, when a program or task is restarted following the handling of a fault, the faulting instruction is restarted (re-executed). Restarting the faulting instruction is commonly used to handle exceptions that are generated when access to an operand is blocked. The most common example of this type of fault is a page-fault exception (#PF) that occurs when a program or task references an operand located on a page that is not in memory. When a page-fault exception occurs, the exception handler can load the page into memory and resume execution of the program or task by restarting the faulting instruction. To ensure that the restart is handled transparently to the currently executing program or task, the processor saves the necessary registers and stack pointers to allow a restart to the state prior to the execution of the faulting instruction.

For trap-class exceptions, the return instruction pointer points to the instruction following the trapping instruction. If a trap is detected during an instruction which transfers execution, the return instruction pointer reflects the transfer. For example, if a trap is detected while executing a JMP instruction, the return instruction pointer points to the destination of the JMP instruction, not to the next address past the JMP instruction. All trap exceptions allow program or task restart with no loss of continuity. For example, the overflow exception is a trap exception. Here, the return instruction pointer points to the instruction following the INTO instruction that tested EFLAGS.OF (overflow) flag. The trap handler for this exception resolves the overflow condition. Upon return from the trap handler, program or task execution continues at the instruction following the INTO instruction.

The abort-class exceptions do not support reliable restarting of the program or task. Abort handlers are designed to collect diagnostic information about the state of the processor when the abort exception occurred and then shut down the application and system as gracefully as possible.

Interrupts rigorously support restarting of interrupted programs and tasks without loss of continuity. The return instruction pointer saved for an interrupt points to the next instruction to be executed at the instruction boundary where the processor took the interrupt. If the instruction just executed has a repeat prefix, the interrupt is taken at the end of the current iteration with the registers set to execute the next iteration.

The ability of a P6 family processor to speculatively execute instructions does not affect the taking of interrupts by the processor. Interrupts are taken at instruction boundaries located during the retirement phase of instruction execution; so they are always taken in the “in-order” instruction stream. See Chapter 2, “Intel® 64 and IA-32 Architectures,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about the P6 family processors’ microarchitecture and its support for out-of-order instruction execution.

Note that the Pentium processor and earlier IA-32 processors also perform varying amounts of prefetching and preliminary decoding. With these processors as well, exceptions and interrupts are not signaled until actual “in-order” execution of the instructions. For a given code sample, the signaling of exceptions occurs uniformly when the code is executed on any family of IA-32 processors (except where new exceptions or new opcodes have been defined).

7.7 NONMASKABLE INTERRUPT (NMI)

The nonmaskable interrupt (NMI) can be generated in either of two ways:

- External hardware asserts the NMI pin.
- The processor receives a message on the system bus (Pentium 4, Intel Core Duo, Intel Core 2, Intel Atom, and Intel Xeon processors) or the APIC serial bus (P6 family and Pentium processors) with a delivery mode NMI.

When the processor receives a NMI from either of these sources, the processor handles it immediately by calling the NMI handler pointed to by interrupt vector number 2. The processor also invokes certain hardware conditions to ensure that no other interrupts, including NMI interrupts, are received until the NMI handler has completed executing (see Section 7.7.1, “Handling Multiple NMIs”).

Also, when an NMI is received from either of the above sources, it cannot be masked by the IF flag in the EFLAGS register.

It is possible to issue a maskable hardware interrupt (through the INTR pin) to vector 2 to invoke the NMI interrupt handler; however, this interrupt will not truly be an NMI interrupt. A true NMI interrupt that activates the processor’s NMI-handling hardware can only be delivered through one of the mechanisms listed above.

7.7.1 Handling Multiple NMIs

While an NMI interrupt handler is executing, the processor blocks delivery of subsequent NMIs until the next execution of the IRET instruction. This blocking of NMIs prevents nested execution of the NMI handler. It is recommended that the NMI interrupt handler be accessed through an interrupt gate to disable maskable hardware interrupts (see Section 7.8.1, “Masking Maskable Hardware Interrupts”).

An execution of the IRET instruction unblocks NMIs even if the instruction causes a fault. For example, if the IRET instruction executes with EFLAGS.VM = 1 and IOPL of less than 3, a general-protection exception is generated (see Section 22.2.7, “Sensitive Instructions”). In such a case, NMIs are unmasked before the exception handler is invoked.

7.8 ENABLING AND DISABLING INTERRUPTS

The processor inhibits the generation of some interrupts, depending on the state of the processor and of the IF and RF flags in the EFLAGS register, as described in the following sections.

7.8.1 Masking Maskable Hardware Interrupts

The IF flag can disable the servicing of maskable hardware interrupts received on the processor's INTR pin or through the local APIC (see Section 7.3.2, "Maskable Hardware Interrupts"). When the IF flag is clear, the processor inhibits interrupts delivered to the INTR pin or through the local APIC from generating an internal interrupt request; when the IF flag is set, interrupts delivered to the INTR or through the local APIC pin are processed as normal external interrupts.

The IF flag does not affect non-maskable interrupts (NMIs) delivered to the NMI pin or delivery mode NMI messages delivered through the local APIC, nor does it affect processor generated exceptions. As with the other flags in the EFLAGS register, the processor clears the IF flag in response to a hardware reset.

The fact that the group of maskable hardware interrupts includes the reserved interrupt and exception vectors 0 through 32 can potentially cause confusion. Architecturally, when the IF flag is set, an interrupt for any of the vectors from 0 through 32 can be delivered to the processor through the INTR pin and any of the vectors from 16 through 32 can be delivered through the local APIC. The processor will then generate an interrupt and call the interrupt or exception handler pointed to by the vector number. So for example, it is possible to invoke the page-fault handler through the INTR pin (by means of vector 14); however, this is not a true page-fault exception. It is an interrupt. As with the *INT n* instruction (see Section 7.4.2, "Software-Generated Exceptions"), when an interrupt is generated through the INTR pin to an exception vector, the processor does not push an error code on the stack, so the exception handler may not operate correctly.

The IF flag can be set or cleared with the STI (set interrupt-enable flag) and CLI (clear interrupt-enable flag) instructions, respectively. These instructions may be executed only if the CPL is equal to or less than the IOPL. A general-protection exception (#GP) is generated if they are executed when the CPL is greater than the IOPL.¹ If IF = 0, maskable hardware interrupts remain inhibited on the instruction boundary following an execution of STI.² The inhibition ends after delivery of another event (e.g., exception) or the execution of the next instruction.

The IF flag is also affected by the following operations:

- The PUSHF instruction stores all flags on the stack, where they can be examined and modified. The POPF instruction can be used to load the modified flags back into the EFLAGS register.
- Task switches and the POPF and IRET instructions load the EFLAGS register; therefore, they can be used to modify the setting of the IF flag.
- When an interrupt is handled through an interrupt gate, the IF flag is automatically cleared, which disables maskable hardware interrupts. (If an interrupt is handled through a trap gate, the IF flag is not cleared.)

See the descriptions of the CLI, STI, PUSHF, POPF, and IRET instructions in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, and Chapter 4, "Instruction Set Reference, M-U," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B, for a detailed description of the operations these instructions are allowed to perform on the IF flag.

7.8.2 Masking Instruction Breakpoints

The RF (resume) flag in the EFLAGS register controls the response of the processor to instruction-breakpoint conditions (see the description of the RF flag in Section 2.3, "System Flags and Fields in the EFLAGS Register").

When set, it prevents an instruction breakpoint from generating a debug exception (#DB); when clear, instruction breakpoints will generate debug exceptions. The primary function of the RF flag is to prevent the processor from going into a debug exception loop on an instruction-breakpoint. See Section 19.3.1.1, "Instruction-Breakpoint Exception Condition," for more information on the use of this flag.

As noted in Section 7.8.3, execution of the MOV or POP instruction to load the SS register suppresses any instruction breakpoint on the next instruction (just as if EFLAGS.RF were 1).

1. The effect of the IOPL on these instructions is modified slightly when the virtual mode extension is enabled by setting the VME flag in control register CR4: see Section 22.3, "Interrupt and Exception Handling in Virtual-8086 Mode." Behavior is also impacted by the PVI flag: see Section 22.4, "Protected-Mode Virtual Interrupts."

2. Nonmaskable interrupts and system-management interrupts may also be inhibited on the instruction boundary following such an execution of STI.

7.8.3 Masking Exceptions and Interrupts When Switching Stacks

To switch to a different stack segment, software often uses a pair of instructions, for example:

```
MOV SS, AX
MOV ESP, StackTop
```

(Software might also use the POP instruction to load SS and ESP.)

If an interrupt or exception occurs after the new SS segment descriptor has been loaded but before the ESP register has been loaded, these two parts of the logical address into the stack space are inconsistent for the duration of the interrupt or exception handler (assuming that delivery of the interrupt or exception does not itself load a new stack pointer).

To account for this situation, the processor prevents certain events from being delivered after execution of a MOV to SS instruction or a POP to SS instruction. The following items provide details:

- Any instruction breakpoint on the next instruction is suppressed (as if EFLAGS.RF were 1).
- Any data breakpoint on the MOV to SS instruction or POP to SS instruction is inhibited until the instruction boundary following the next instruction.
- Any single-step trap that would be delivered following the MOV to SS instruction or POP to SS instruction (because EFLAGS.TF is 1) is suppressed.
- The suppression and inhibition ends after delivery of an exception or the execution of the next instruction.
- If a sequence of consecutive instructions each loads the SS register (using MOV or POP), only the first is guaranteed to inhibit or suppress events in this way.

Intel recommends that software use the LSS instruction to load the SS register and ESP together. The problem identified earlier does not apply to LSS, and the LSS instruction does not inhibit events as detailed above.

7.9 PRIORITIZATION OF CONCURRENT EVENTS

If more than one event is pending at an instruction boundary (between execution of instructions), the processor services them in a predictable order. Table 7-2 shows the priority among classes of event sources.

Table 7-2. Priority Among Concurrent Events

Priority	Description
1 (Highest)	Hardware Reset and Machine Checks - RESET - Machine Check (#MC)
2	Trap on Task Switch - T flag in TSS is set (#DB)
3	External Hardware Interventions - FLUSH - STOPCLK - SMI - INIT
4	Traps on the Previous Instruction - Trap-class Debug Exceptions (#DB due to TF flag set or data/I-O breakpoint)
5	Nonmaskable Interrupts (NMI) ¹
6	Maskable Hardware Interrupts ¹
7	Fault-class Debug Exceptions (#DB due to instruction breakpoint)

Table 7-2. Priority Among Concurrent Events (Contd.)

Priority	Description
8	Faults from Fetching Next Instruction - Code-Segment Limit Violation (#GP) - Code Page Fault (#PF)
9 (Lowest)	Faults from Decoding the Next Instruction - Control protection exception due to missing ENDBRANCH at target of an indirect call or jump (#CP) - Instruction length > 15 bytes (#GP) - Invalid Opcode (#UD) - Coprocessor Not Available (#NM)

NOTE

1. The Intel® 486 processor and earlier processors group nonmaskable and maskable interrupts in the same priority class.

The processor first services a pending event from the class which has the highest priority, transferring execution to the first instruction of the handler. Lower priority exceptions are discarded; lower priority interrupts are held pending. Discarded exceptions may be re-generated when the event handler returns execution to the point in the program or task where the original event occurred. While the priority among the classes listed in Table 7-2 is consistent across processor implementations, the priority of events within a class is implementation-dependent and may vary from processor to processor.

Table 7-2 specifies the prioritization of events that may be pending at an instruction boundary. It does not specify the prioritization of faults that arise during instruction execution or event delivery (these include #BR, #TS, #NP, #SS, #GP, #PF, #AC, #MF, #XM, #VE, or #CP). It also does not apply to the events generated by the "Call to Interrupt Procedure" instructions (INT n, INTO, INT3, and INT1), as these events are integral to the execution of those instructions and do not occur between instructions.

7.10 INTERRUPT DESCRIPTOR TABLE (IDT)

The interrupt descriptor table (IDT) associates each exception or interrupt vector with a gate descriptor for the procedure or task used to service the associated exception or interrupt. Like the GDT and LDTs, the IDT is an array of 8-byte descriptors (in protected mode). Unlike the GDT, the first entry of the IDT may contain a descriptor. To form an index into the IDT, the processor scales the exception or interrupt vector by eight (the number of bytes in a gate descriptor). Because there are only 256 interrupt or exception vectors, the IDT need not contain more than 256 descriptors. It can contain fewer than 256 descriptors, because descriptors are required only for the interrupt and exception vectors that may occur. All empty descriptor slots in the IDT should have the present flag for the descriptor set to 0.

The base addresses of the IDT should be aligned on an 8-byte boundary to maximize performance of cache line fills. The limit value is expressed in bytes and is added to the base address to get the address of the last valid byte. A limit value of 0 results in exactly 1 valid byte. Because IDT entries are always eight bytes long, the limit should always be one less than an integral multiple of eight (that is, $8N - 1$).

The IDT may reside anywhere in the linear address space. As shown in Figure 7-1, the processor locates the IDT using the IDTR register. This register holds both a 32-bit base address and 16-bit limit for the IDT.

The LIDT (load IDT register) and SIDT (store IDT register) instructions load and store the contents of the IDTR register, respectively. The LIDT instruction loads the IDTR register with the base address and limit held in a memory operand. This instruction can be executed only when the CPL is 0. It normally is used by the initialization code of an operating system when creating an IDT. An operating system also may use it to change from one IDT to another. The SIDT instruction copies the base and limit value stored in IDTR to memory. This instruction can be executed at any privilege level.

If a vector references a descriptor beyond the limit of the IDT, a general-protection exception (#GP) is generated.

NOTE

Because interrupts are delivered to the processor core only once, an incorrectly configured IDT could result in incomplete interrupt handling and/or the blocking of interrupt delivery.

IA-32 architecture rules need to be followed for setting up IDTR base/limit/access fields and each field in the gate descriptors. The same apply for the Intel 64 architecture. This includes implicit referencing of the destination code segment through the GDT or LDT and accessing the stack.

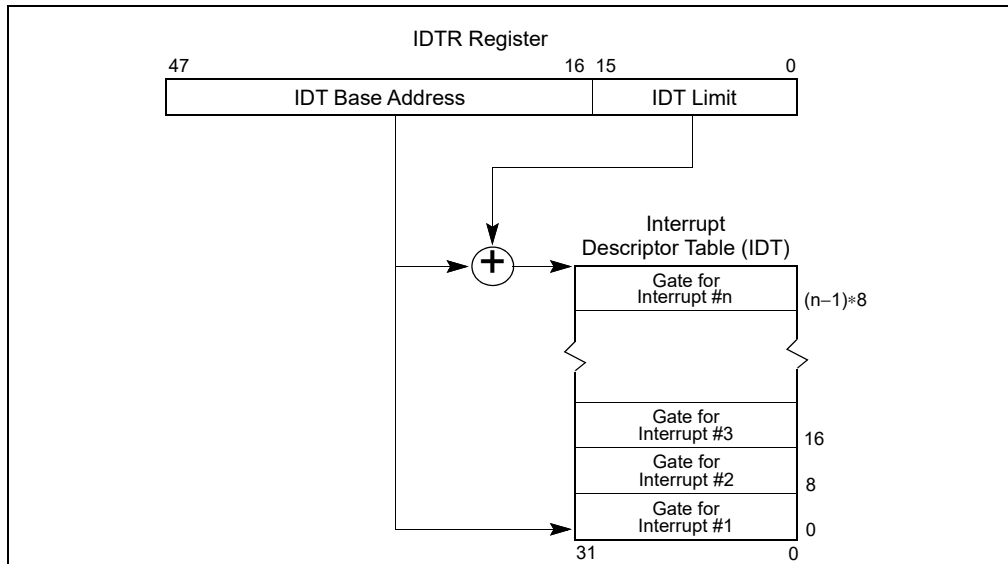


Figure 7-1. Relationship of the IDTR and IDT

7.11 IDT DESCRIPTORS

The IDT may contain any of three kinds of gate descriptors:

- Task-gate descriptor
- Interrupt-gate descriptor
- Trap-gate descriptor

Figure 7-2 shows the formats for the task-gate, interrupt-gate, and trap-gate descriptors. The format of a task gate used in an IDT is the same as that of a task gate used in the GDT or an LDT (see Section 9.2.5, "Task-Gate Descriptor"). The task gate contains the segment selector for a TSS for an exception and/or interrupt handler task.

Interrupt and trap gates are very similar to call gates (see Section 6.8.3, "Call Gates"). They contain a far pointer (segment selector and offset) that the processor uses to transfer program execution to a handler procedure in an exception- or interrupt-handler code segment. These gates differ in the way the processor handles the IF flag in the EFLAGS register (see Section 7.12.1.3, "Flag Usage By Exception- or Interrupt-Handler Procedure").

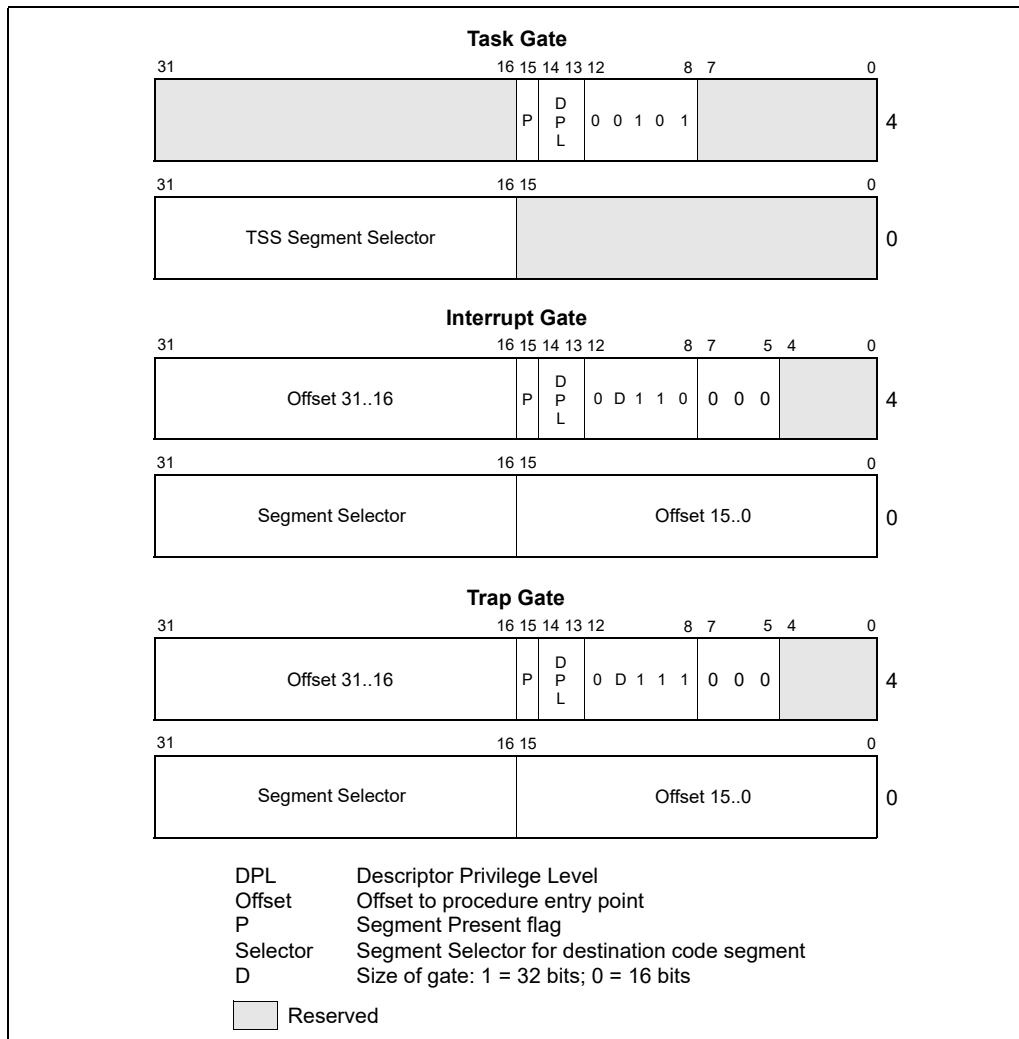


Figure 7-2. IDT Gate Descriptors

7.12 EXCEPTION AND INTERRUPT HANDLING

The processor handles calls to exception- and interrupt-handlers similar to the way it handles calls with a CALL instruction to a procedure or a task. When responding to an exception or interrupt, the processor uses the exception or interrupt vector as an index to a descriptor in the IDT. If the index points to an interrupt gate or trap gate, the processor calls the exception or interrupt handler in a manner similar to a CALL to a call gate (see Section 6.8.2, “Gate Descriptors,” through Section 6.8.6, “Returning from a Called Procedure”). If index points to a task gate, the processor executes a task switch to the exception- or interrupt-handler task in a manner similar to a CALL to a task gate (see Section 9.3, “Task Switching”).

7.12.1 Exception- or Interrupt-Handler Procedures

An interrupt gate or trap gate references an exception- or interrupt-handler procedure that runs in the context of the currently executing task (see Figure 7-3). The segment selector for the gate points to a segment descriptor for an executable code segment in either the GDT or the current LDT. The offset field of the gate descriptor points to the beginning of the exception- or interrupt-handling procedure.

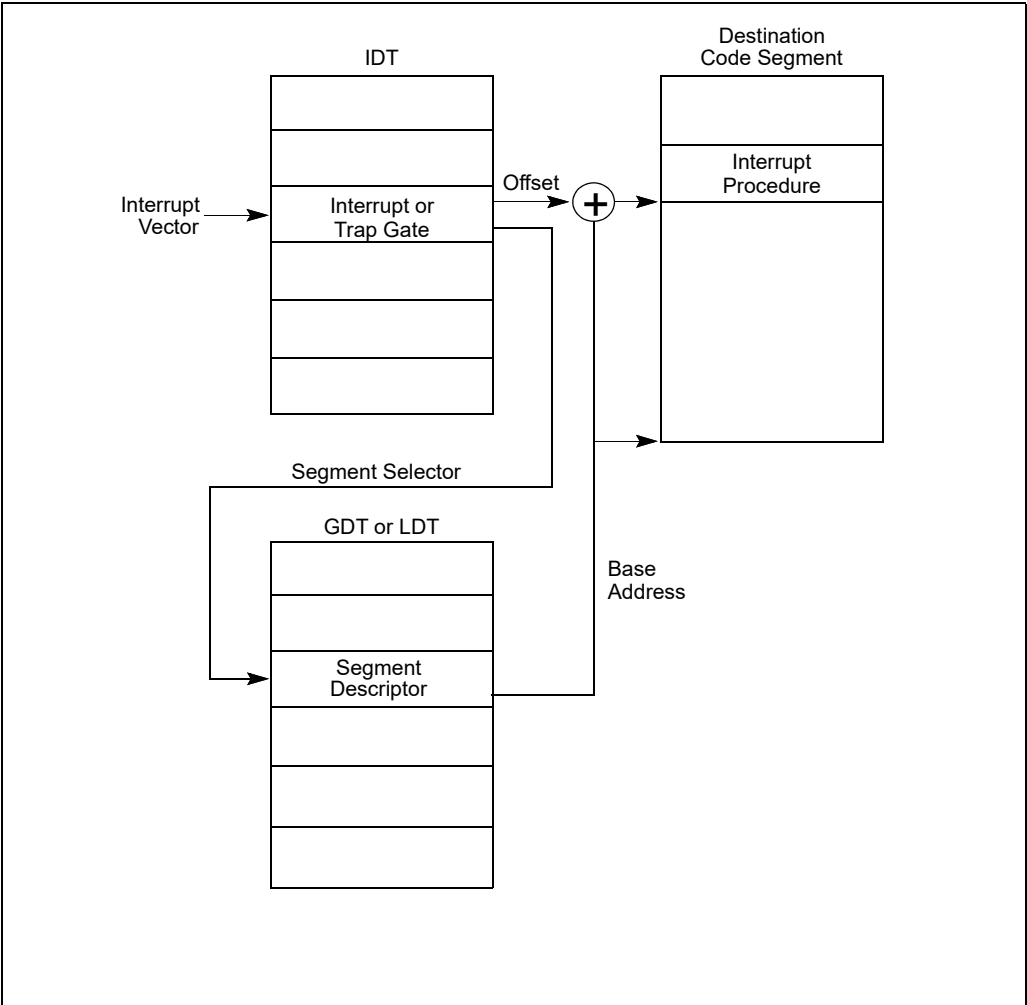


Figure 7-3. Interrupt Procedure Call

When the processor performs a call to the exception- or interrupt-handler procedure:

- If the handler procedure is going to be executed at a numerically lower privilege level, a stack switch occurs. When the stack switch occurs:
 - a. The segment selector and stack pointer for the stack to be used by the handler are obtained from the TSS for the currently executing task. On this new stack, the processor pushes the stack segment selector and stack pointer of the interrupted procedure.
 - b. The processor then saves the current state of the EFLAGS, CS, and EIP registers on the new stack (see Figure 7-4).
 - c. If an exception causes an error code to be saved, it is pushed on the new stack after the EIP value.
- If the handler procedure is going to be executed at the same privilege level as the interrupted procedure:
 - a. The processor saves the current state of the EFLAGS, CS, and EIP registers on the current stack (see Figure 7-4).
 - b. If an exception causes an error code to be saved, it is pushed on the current stack after the EIP value.

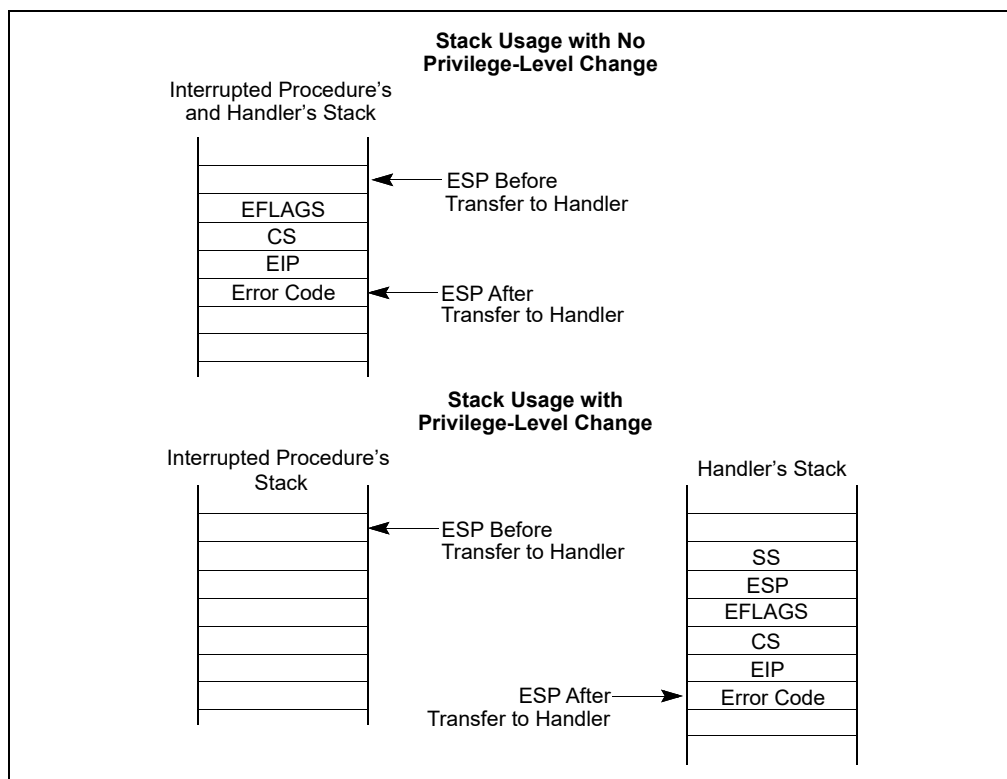


Figure 7-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines

To return from an exception- or interrupt-handler procedure, the handler must use the IRET (or IRETD) instruction. The IRET instruction is similar to the RET instruction except that it restores the saved flags into the EFLAGS register. The IOPL field of the EFLAGS register is restored only if the CPL is 0. The IF flag is changed only if the CPL is less than or equal to the IOPL. See Chapter 3, "Instruction Set Reference, A-L," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for a description of the complete operation performed by the IRET instruction.

If a stack switch occurred when calling the handler procedure, the IRET instruction switches back to the interrupted procedure's stack on the return.

7.12.1.1 Shadow Stack Usage on Transfers to Interrupt and Exception Handling Routines

When the processor performs a call to the exception- or interrupt-handler procedure:

- If the handler procedure is going to be executed at a numerically lower privilege level, a shadow stack switch occurs. When the shadow stack switch occurs:
 - a. On a transfer from privilege level 3, if shadow stacks are enabled at privilege level 3 then the SSP is saved to the IA32_PL3_SSP MSR.
 - b. If shadow stacks are enabled at the privilege level where the handler will execute then the shadow stack for the handler is obtained from one of the following MSRs based on the privilege level at which the handler executes.
 - IA32_PL2_SSP if handler executes at privilege level 2.
 - IA32_PL1_SSP if handler executes at privilege level 1.
 - IA32_PL0_SSP if handler executes at privilege level 0.
 - c. The SSP obtained is then verified to ensure it points to a valid supervisory shadow stack that is not currently active by verifying a supervisor shadow stack token at the address pointed to by the SSP. The operations performed to verify and acquire the supervisor shadow stack token by making it busy are as described in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
 - d. On this new shadow stack, the processor pushes the CS, LIP (CS.base + EIP), and SSP of the interrupted procedure if the interrupted procedure was executing at privilege level less than 3; see Figure 7-5.¹
- If the handler procedure is going to be executed at the same privilege level as the interrupted procedure and shadow stacks are enabled at current privilege level:
 - a. The processor saves the current state of the CS, LIP (CS.base + EIP), and SSP registers on the current shadow stack; see Figure 7-5.

1. If any of these pushes leads to an exception or a VM exit, the supervisor shadow-stack token remains busy.

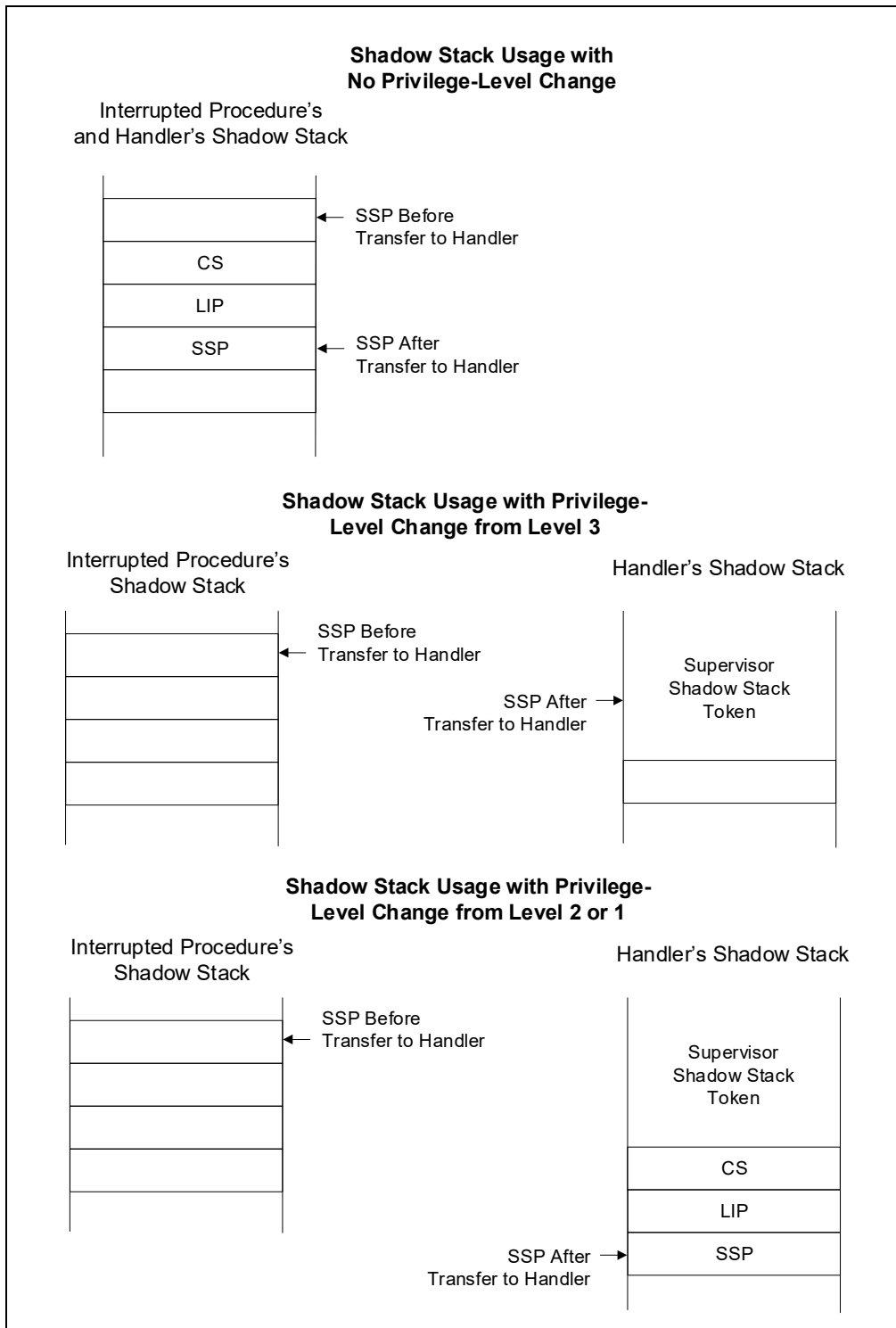


Figure 7-5. Shadow Stack Usage on Transfers to Interrupt and Exception-Handling Routines

To return from an exception- or interrupt-handler procedure, the handler must use the IRET (or IRETD) instruction. When executing a return from an interrupt or exception handler from the same privilege level as the interrupted procedure, the processor performs these actions to enforce return address protection:

- Restores the CS and EIP registers to their values prior to the interrupt or exception.

If shadow stack is enabled:

- Compares the values on shadow stack at address $SSP+8$ (the LIP) and $SSP+16$ (the CS) to the CS and $(CS.base + EIP)$ popped from the stack and causes a control protection exception ($\#CP(FAR-RET/IRET)$) if they do not match.
- Pops the top-of-stack value (the SSP prior to the interrupt or exception) from shadow stack into SSP register.

When executing a return from an interrupt or exception handler from a different privilege level than the interrupted procedure, the processor performs the actions below.

- If shadow stack is enabled at current privilege level:
 - If SSP is not aligned to 8 bytes then causes a control protection exception ($\#CP(FAR-RET/IRET)$).
 - If privilege level of the procedure being returned to is less than 3 (returning to supervisor mode):
 - Compares the values on shadow stack at address $SSP+8$ (the LIP) and $SSP+16$ (the CS) to the CS and $(CS.base + EIP)$ popped from the stack and causes a control protection exception ($\#CP(FAR-RET/IRET)$) if they do not match.
 - Temporarily saves the top-of-stack value (the SSP of the procedure being returned to) internally.
 - If a busy supervisor shadow stack token is present at address $SSP+24$, then marks the token free using operations described in section Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.
 - If the privilege level of the procedure being returned to is less than 3 (returning to supervisor mode), restores the SSP register from the internally saved value.
 - If the privilege level of the procedure being returned to is 3 (returning to user mode) and shadow stack is enabled at privilege level 3, then restores the SSP register with value of $IA32\_PL3\_SSP$ MSR.

7.12.1.2 Protection of Exception- and Interrupt-Handler Procedures

The privilege-level protection for exception- and interrupt-handler procedures is similar to that used for ordinary procedure calls when called through a call gate (see Section 6.8.4, "Accessing a Code Segment Through a Call Gate"). The processor does not permit transfer of execution to an exception- or interrupt-handler procedure in a less privileged code segment (numerically greater privilege level) than the CPL.

An attempt to violate this rule results in a general-protection exception ($\#GP$). The protection mechanism for exception- and interrupt-handler procedures is different in the following ways:

- Because interrupt and exception vectors have no RPL, the RPL is not checked on implicit calls to exception and interrupt handlers.
- The processor checks the DPL of the interrupt or trap gate only if an exception or interrupt is generated with an $INT\ n$, $INT3$, or $INTO$ instruction.¹ Here, the CPL must be less than or equal to the DPL of the gate. This restriction prevents application programs or procedures running at privilege level 3 from using a software interrupt to access critical exception handlers, such as the page-fault handler, providing that those handlers are placed in more privileged code segments (numerically lower privilege level). For hardware-generated interrupts and processor-detected exceptions, the processor ignores the DPL of interrupt and trap gates.

Because exceptions and interrupts generally do not occur at predictable times, these privilege rules effectively impose restrictions on the privilege levels at which exception and interrupt- handling procedures can run. Either of the following techniques can be used to avoid privilege-level violations.

- The exception or interrupt handler can be placed in a conforming code segment. This technique can be used for handlers that only need to access data available on the stack (for example, divide error exceptions). If the handler needs data from a data segment, the data segment needs to be accessible from privilege level 3, which would make it unprotected.
- The handler can be placed in a nonconforming code segment with privilege level 0. This handler would always run, regardless of the CPL that the interrupted program or task is running at.

1. This check is not performed by execution of the $INT1$ instruction (opcode $F1$); it would be performed by execution of $INT\ 1$ (opcode $CD\ 01$).

7.12.1.3 Flag Usage By Exception- or Interrupt-Handler Procedure

When accessing an exception or interrupt handler through either an interrupt gate or a trap gate, the processor clears the TF flag in the EFLAGS register after it saves the contents of the EFLAGS register on the stack. (On calls to exception and interrupt handlers, the processor also clears the VM, RF, and NT flags in the EFLAGS register, after they are saved on the stack.) Clearing the TF flag prevents instruction tracing from affecting interrupt response and ensures that no single-step exception will be delivered after delivery to the handler. A subsequent IRET instruction restores the TF (and VM, RF, and NT) flags to the values in the saved contents of the EFLAGS register on the stack.

The only difference between an interrupt gate and a trap gate is the way the processor handles the IF flag in the EFLAGS register. When accessing an exception- or interrupt-handling procedure through an interrupt gate, the processor clears the IF flag to prevent other interrupts from interfering with the current interrupt handler. A subsequent IRET instruction restores the IF flag to its value in the saved contents of the EFLAGS register on the stack. Accessing a handler procedure through a trap gate does not affect the IF flag.

7.12.2 Interrupt Tasks

When an exception or interrupt handler is accessed through a task gate in the IDT, a task switch results. Handling an exception or interrupt with a separate task offers several advantages:

- The entire context of the interrupted program or task is saved automatically.
- A new TSS permits the handler to use a new privilege level 0 stack when handling the exception or interrupt. If an exception or interrupt occurs when the current privilege level 0 stack is corrupted, accessing the handler through a task gate can prevent a system crash by providing the handler with a new privilege level 0 stack.
- The handler can be further isolated from other tasks by giving it a separate address space. This is done by giving it a separate LDT.

The disadvantage of handling an interrupt with a separate task is that the amount of machine state that must be saved on a task switch makes it slower than using an interrupt gate, resulting in increased interrupt latency.

A task gate in the IDT references a TSS descriptor in the GDT (see Figure 7-6). A switch to the handler task is handled in the same manner as an ordinary task switch (see Section 9.3, "Task Switching"). The link back to the interrupted task is stored in the previous task link field of the handler task's TSS. If an exception caused an error code to be generated, this error code is copied to the stack of the new task.

When exception- or interrupt-handler tasks are used in an operating system, there are actually two mechanisms that can be used to dispatch tasks: the software scheduler (part of the operating system) and the hardware scheduler (part of the processor's interrupt mechanism). The software scheduler needs to accommodate interrupt tasks that may be dispatched when interrupts are enabled.

NOTE

Because IA-32 architecture tasks are not re-entrant, an interrupt-handler task must disable interrupts between the time it completes handling the interrupt and the time it executes the IRET instruction. This action prevents another interrupt from occurring while the interrupt task's TSS is still marked busy, which would cause a general-protection (#GP) exception.

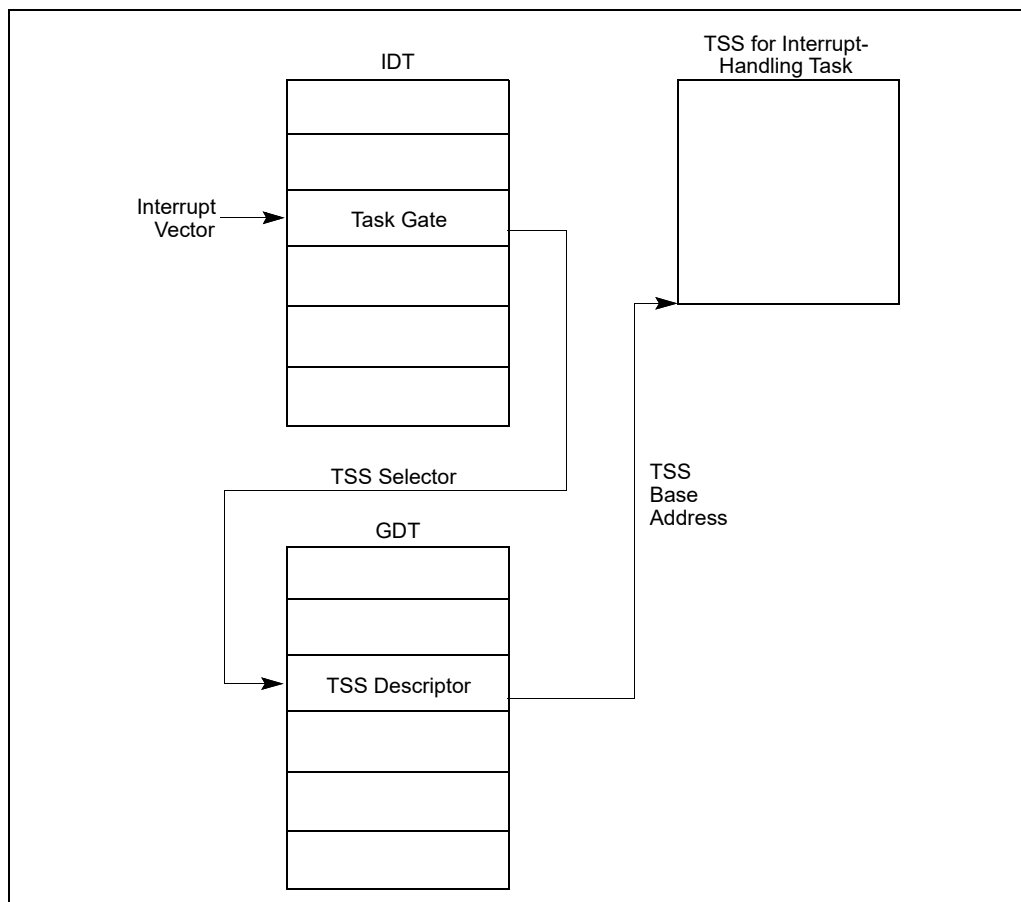


Figure 7-6. Interrupt Task Switch

7.13 ERROR CODE

When an exception condition is related to a specific segment selector or IDT vector, the processor pushes an error code onto the stack of the exception handler (whether it is a procedure or task). The error code has the format shown in Figure 7-7. The error code resembles a segment selector; however, instead of a TI flag and RPL field, the error code contains 3 flags:

- EXT** **External event (bit 0)** — When set, indicates that the exception occurred during delivery of an event external to the program, such as an interrupt or an earlier exception.¹ The bit is cleared if the exception occurred during delivery of a software interrupt (INT *n*, INT3, or INTO).
- IDT** **Descriptor location (bit 1)** — When set, indicates that the index portion of the error code refers to a gate descriptor in the IDT; when clear, indicates that the index refers to a descriptor in the GDT or the current LDT.
- TI** **GDT/LDT (bit 2)** — Only used when the IDT flag is clear. When set, the TI flag indicates that the index portion of the error code refers to a segment or gate descriptor in the LDT; when clear, it indicates that the index refers to a descriptor in the current GDT.

1. The bit is also set if the exception occurred during delivery of INT1.

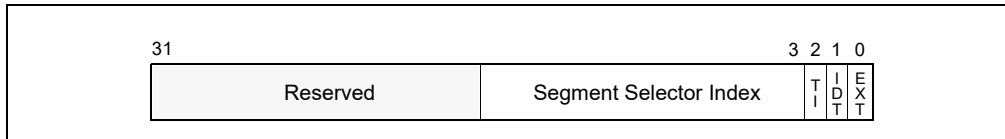


Figure 7-7. Error Code

The segment selector index field provides an index into the IDT, GDT, or current LDT to the segment or gate selector being referenced by the error code. In some cases the error code is null (all bits are clear except possibly EXT). A null error code indicates that the error was not caused by a reference to a specific segment or that a null segment selector was referenced in an operation.

The format of the error code is different for page-fault exceptions (#PF). See the “Interrupt 14—Page-Fault Exception (#PF)” section in this chapter.

The format of the error code is different for control protection exceptions (#CP). See the “Interrupt 21—Control Protection Exception (#CP)” section in this chapter.

The error code is pushed on the stack as a doubleword or word (depending on the default interrupt, trap, or task gate size). To keep the stack aligned for doubleword pushes, the upper half of the error code is reserved. Note that the error code is not popped when the IRET instruction is executed to return from an exception handler, so the handler must remove the error code before executing a return.

Error codes are not pushed on the stack for exceptions that are generated externally (with the INTR or LINT[1:0] pins) or the INT *n* instruction, even if an error code is normally produced for those exceptions.

7.14 EXCEPTION AND INTERRUPT HANDLING IN 64-BIT MODE

In 64-bit mode, interrupt and exception handling is similar to what has been described for non-64-bit modes. The following are the exceptions:

- All interrupt handlers pointed by the IDT are in 64-bit code (this does not apply to the SMI handler).
- The size of interrupt-stack pushes is fixed at 64 bits; and the processor uses 8-byte, zero extended stores.
- The stack pointer (SS:RSP) is pushed unconditionally on interrupts. In legacy modes, this push is conditional and based on a change in current privilege level (CPL).
- The new SS is set to NULL if there is a change in CPL.
- IRET behavior changes.
- There is a new interrupt stack-switch mechanism and a new interrupt shadow stack-switch mechanism.
- The alignment of interrupt stack frame is different.

7.14.1 64-Bit Mode IDT

Interrupt and trap gates are 16 bytes in length to provide a 64-bit offset for the instruction pointer (RIP). The 64-bit RIP referenced by interrupt-gate descriptors allows an interrupt service routine to be located anywhere in the linear-address space. See Figure 7-8.

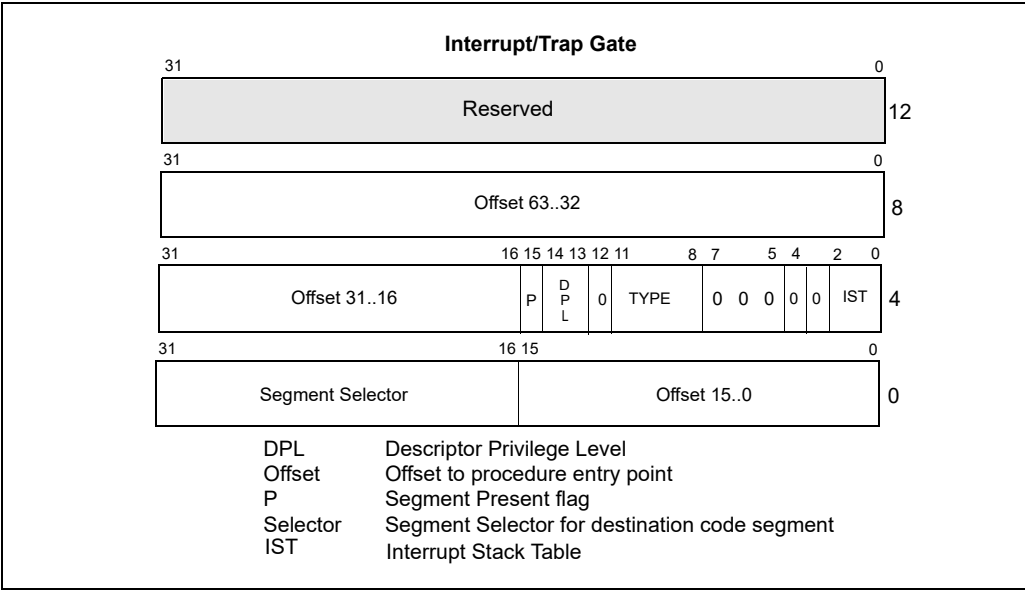


Figure 7-8. 64-Bit IDT Gate Descriptors

In 64-bit mode, the IDT index is formed by scaling the interrupt vector by 16. The first eight bytes (bytes 7:0) of a 64-bit mode interrupt gate are similar but not identical to legacy 32-bit interrupt gates. The type field (bits 11:8 in bytes 7:4) is described in Table 3-2. The Interrupt Stack Table (IST) field (bits 4:0 in bytes 7:4) is used by the stack switching mechanisms described in Section 7.14.5, “Interrupt Stack Table.” Bytes 11:8 hold the upper 32 bits of the target RIP (interrupt segment offset) in canonical form. A general-protection exception (#GP) is generated if software attempts to reference an interrupt gate with a target RIP that is not in canonical form.

The target code segment referenced by the interrupt gate must be a 64-bit code segment (CS.L = 1, CS.D = 0). If the target is not a 64-bit code segment, a general-protection exception (#GP) is generated with the IDT vector number reported as the error code.

Only 64-bit interrupt and trap gates can be referenced in IA-32e mode (64-bit mode and compatibility mode). Legacy 32-bit interrupt or trap gate types (0EH or 0FH) are redefined in IA-32e mode as 64-bit interrupt and trap gate types. No 32-bit interrupt or trap gate type exists in IA-32e mode. If a reference is made to a 16-bit interrupt or trap gate (06H or 07H), a general-protection exception (#GP(0)) is generated.

7.14.2 64-Bit Mode Stack Frame

In legacy mode, the size of an IDT entry (16 bits or 32 bits) determines the size of interrupt-stack-frame pushes. SS:ESP is pushed only on a CPL change. In 64-bit mode, the size of interrupt stack-frame pushes is fixed at eight bytes. This is because only 64-bit mode gates can be referenced. 64-bit mode also pushes SS:RSP unconditionally, rather than only on a CPL change.

When shadow stacks are enabled at the interrupt handler’s privilege level and the interrupted procedure was not executing at a privilege level 3, then the processor pushes the CS:LIP:SSP of the interrupted procedure on the shadow stack of the interrupt handler (where LIP is the linear address of the return address).

Aside from error codes, pushing SS:RSP unconditionally presents operating systems with a consistent interrupt-stackframe size across all interrupts. Interrupt service-routine entry points that handle interrupts generated by the INTn instruction or external INTR# signal can push an additional error code place-holder to maintain consistency.

In legacy mode, the stack pointer may be at any alignment when an interrupt or exception causes a stack frame to be pushed. This causes the stack frame and succeeding pushes done by an interrupt handler to be at arbitrary alignments. In IA-32e mode, the RSP is aligned to a 16-byte boundary before pushing the stack frame. The stack frame itself is aligned on a 16-byte boundary when the interrupt handler is called. The processor can arbitrarily realign the new RSP on interrupts because the previous (possibly unaligned) RSP is unconditionally saved on the newly aligned stack. The previous RSP will be automatically restored by a subsequent IRET.

Aligning the stack permits exception and interrupt frames to be aligned on a 16-byte boundary before interrupts are re-enabled. This allows the stack to be formatted for optimal storage of 16-byte XMM registers, which enables the interrupt handler to use faster 16-byte aligned loads and stores (MOVAPS rather than MOVUPS) to save and restore XMM registers.

Although the RSP alignment is always performed when LMA = 1, it is only of consequence for the kernel-mode case where there is no stack switch or IST used. For a stack switch or IST, the OS would have presumably put suitably aligned RSP values in the TSS.

7.14.3 IRET in IA-32e Mode

In IA-32e mode, IRET executes with an 8-byte operand size. There is nothing that forces this requirement. The stack is formatted in such a way that for actions where IRET is required, the 8-byte IRET operand size works correctly.

Because interrupt stack-frame pushes are always eight bytes in IA-32e mode, an IRET must pop eight byte items off the stack. This is accomplished by preceding the IRET with a 64-bit operand-size prefix. The size of the pop is determined by the address size of the instruction. The SS/ESP/RSP size adjustment is determined by the stack size.

IRET pops SS:RSP unconditionally off the interrupt stack frame only when it is executed in 64-bit mode. In compatibility mode, IRET pops SS:RSP off the stack only if there is a CPL change. This allows legacy applications to execute properly in compatibility mode when using the IRET instruction. 64-bit interrupt service routines that exit with an IRET unconditionally pop SS:RSP off of the interrupt stack frame, even if the target code segment is running in 64-bit mode or at CPL = 0. This is because the original interrupt always pushes SS:RSP.

When shadow stacks are enabled and the target privilege level is not 3, the CS:LIP from the shadow stack frame is compared to the return linear address formed by CS:EIP from the stack. If they do not match then the processor caused a control protection exception (#CP(FAR-RET/IRET)), else the processor pops the SSP of the interrupted procedure from the shadow stack. If the target privilege level is 3 and shadow stacks are enabled at privilege level 3, then the SSP for the interrupted procedure is restored from the IA32_PL3_SSP MSR.

In IA-32e mode, IRET is allowed to load a NULL SS under certain conditions. If the target mode is 64-bit mode and the target CPL $\geq$ 3, IRET allows SS to be loaded with a NULL selector. As part of the stack switch mechanism, an interrupt or exception sets the new SS to NULL, instead of fetching a new SS selector from the TSS and loading the corresponding descriptor from the GDT or LDT. The new SS selector is set to NULL in order to properly handle returns from subsequent nested far transfers. If the called procedure itself is interrupted, the NULL SS is pushed on the stack frame. On the subsequent IRET, the NULL SS on the stack acts as a flag to tell the processor not to load a new SS descriptor.

7.14.4 Stack Switching in IA-32e Mode

The IA-32 architecture provides a mechanism to automatically switch stack frames in response to an interrupt. The 64-bit extensions of Intel 64 architecture implement a modified version of the legacy stack-switching mechanism and an alternative stack-switching mechanism called the interrupt stack table (IST).

In IA-32 modes, the legacy IA-32 stack-switch mechanism is unchanged. In IA-32e mode, the legacy stack-switch mechanism is modified. When stacks are switched as part of a 64-bit mode privilege-level change (resulting from an interrupt), a new SS descriptor is not loaded. IA-32e mode loads only an inner-level RSP from the TSS. The new SS selector is forced to NULL and the SS selector's RPL field is set to the new CPL. The new SS is set to NULL in order to handle nested far transfers (far CALL, INT, interrupts, and exceptions). The old SS and RSP are saved on the new stack (Figure 7-9). On the subsequent IRET, the old SS is popped from the stack and loaded into the SS register.

In summary, a stack switch in IA-32e mode works like the legacy stack switch, except that a new SS selector is not loaded from the TSS. Instead, the new SS is forced to NULL.

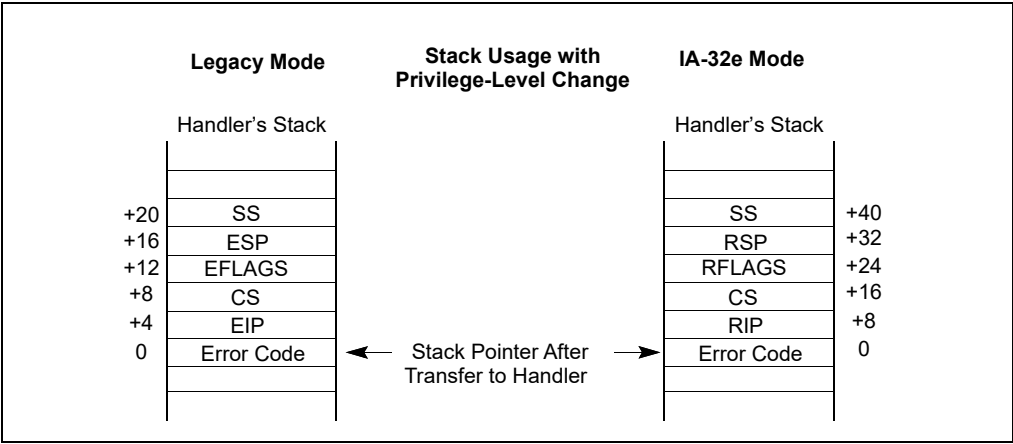


Figure 7-9. IA-32e Mode Stack Usage After Privilege Level Change

7.14.5 Interrupt Stack Table

In IA-32e mode, a new interrupt stack table (IST) mechanism is available as an alternative to the modified legacy stack-switching mechanism described above. This mechanism unconditionally switches stacks when it is enabled. It can be enabled on an individual interrupt-vector basis using a field in the IDT entry. This means that some interrupt vectors can use the modified legacy mechanism and others can use the IST mechanism.

The IST mechanism is only available in IA-32e mode. It is part of the 64-bit mode TSS. The motivation for the IST mechanism is to provide a method for specific interrupts (such as NMI, double-fault, and machine-check) to always execute on a known good stack. In legacy mode, interrupts can use the task-switch mechanism to set up a known-good stack by accessing the interrupt service routine through a task gate located in the IDT. However, the legacy task-switch mechanism is not supported in IA-32e mode.

The IST mechanism provides up to seven IST pointers in the TSS. The pointers are referenced by an interrupt-gate descriptor in the interrupt-descriptor table (IDT); see Figure 7-8. The gate descriptor contains a 3-bit IST index field that provides an offset into the IST section of the TSS. Using the IST mechanism, the processor loads the value pointed by an IST pointer into the RSP.

When an interrupt occurs, the new SS selector is forced to NULL and the SS selector’s RPL field is set to the new CPL. The old SS, RSP, RFLAGS, CS, and RIP are pushed onto the new stack. Interrupt processing then proceeds as normal. If the IST index is zero, the modified legacy stack-switching mechanism described above is used.

To support this stack-switching mechanism with shadow stacks enabled, the processor provides an MSR, IA32_INTERRUPT_SSP_TABLE, to program the linear address of a table of seven shadow stack pointers that are selected using the IST index from the gate descriptor. To switch to a shadow stack selected from the interrupt shadow stack table pointed to by the IA32_INTERRUPT_SSP_TABLE, the processor requires that the shadow stack addresses programmed into this table point to a supervisor shadow stack token; see Figure 7-10.

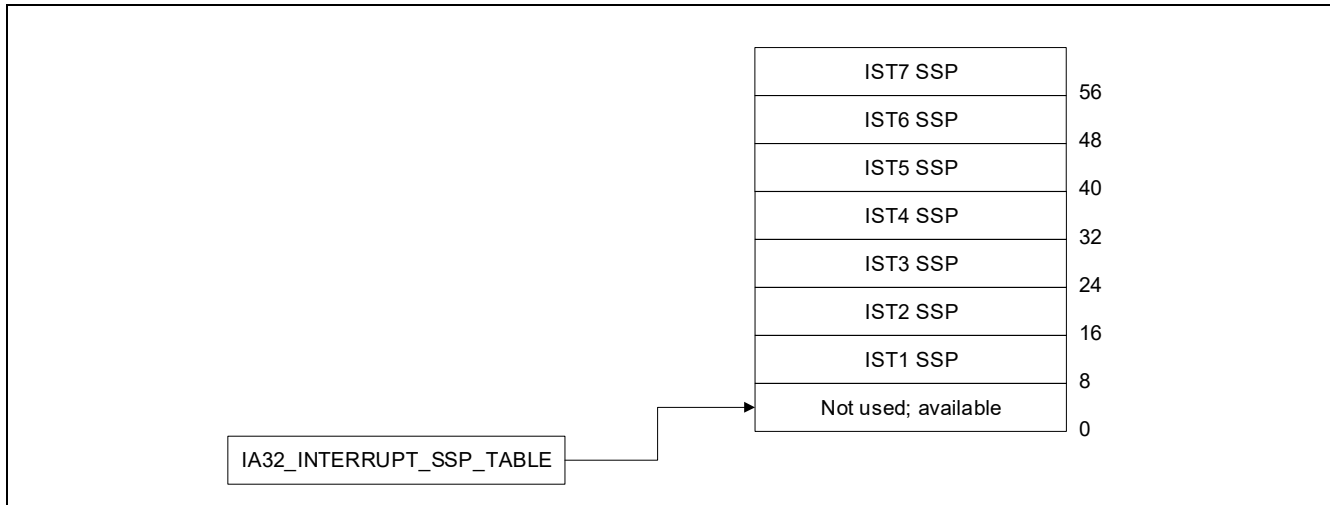


Figure 7-10. Interrupt Shadow Stack Table

7.15 EXCEPTION AND INTERRUPT REFERENCE

The following sections describe conditions which generate exceptions and interrupts. They are arranged in the order of vector numbers. The information contained in these sections are as follows:

- **Exception Class** — Indicates whether the exception class is a fault, trap, or abort type. Some exceptions can be either a fault or trap type, depending on when the error condition is detected. (This section is not applicable to interrupts.)
- **Description** — Gives a general description of the purpose of the exception or interrupt type. It also describes how the processor handles the exception or interrupt.
- **Exception Error Code** — Indicates whether an error code is saved for the exception. If one is saved, the contents of the error code are described. (This section is not applicable to interrupts.)
- **Saved Instruction Pointer** — Describes which instruction the saved (or return) instruction pointer points to. It also indicates whether the pointer can be used to restart a faulting instruction.
- **Program State Change** — Describes the effects of the exception or interrupt on the state of the currently running program or task and the possibilities of restarting the program or task without loss of continuity.

Interrupt 0—Divide Error Exception (#DE)

Exception Class **Fault.**

Description

Indicates the divisor operand for a DIV or IDIV instruction is 0 or that the result cannot be represented in the number of bits specified for the destination operand.

Exception Error Code

None.

Saved Instruction Pointer

Saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany the divide error, because the exception occurs before the faulting instruction is executed.

Interrupt 1—Debug Exception (#DB)

Exception Class **Trap or Fault.** The exception handler can distinguish between traps or faults by examining the contents of DR6 and the other debug registers.

Description

Indicates that one or more of several debug-exception conditions has been detected. Whether the exception is a fault or a trap depends on the condition (see Table 7-3). See Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” for detailed information about the debug exceptions.

Table 7-3. Debug Exception Conditions and Corresponding Exception Classes

Exception Condition	Exception Class
Instruction fetch breakpoint	Fault
Data read or write breakpoint	Trap
I/O read or write breakpoint	Trap
General detect condition (in conjunction with in-circuit emulation)	Fault
Single-step	Trap
Task-switch	Trap
Execution of INT1 ¹	Trap

NOTES:

1. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

Exception Error Code

None. An exception handler can examine the debug registers to determine which condition caused the exception.

Saved Instruction Pointer

Fault — Saved contents of CS and EIP registers point to the instruction that generated the exception.

Trap — Saved contents of CS and EIP registers point to the instruction following the instruction that generated the exception.

Program State Change

Fault — A program-state change does not accompany the debug exception, because the exception occurs before the faulting instruction is executed. The program can resume normal execution upon returning from the debug exception handler.

Trap — A program-state change does accompany the debug exception, because the instruction or task switch being executed is allowed to complete before the exception is generated. However, the new state of the program is not corrupted and execution of the program can continue reliably.

The following items detail the treatment of debug exceptions on the instruction boundary following execution of the MOV or the POP instruction that loads the SS register:

- If EFLAGS.TF is 1, no single-step trap is generated.
- If the instruction encounters a data breakpoint, the resulting debug exception is delivered after completion of the instruction after the MOV or POP. This occurs even if the next instruction is INT *n*, INT3, or INTO.
- Any instruction breakpoint on the instruction after the MOV or POP is suppressed (as if EFLAGS.RF were 1).

Any debug exception inside an RTM region causes a transactional abort and, by default, redirects control flow to the fallback instruction address. If advanced debugging of RTM transactional regions has been enabled, any transactional abort due to a debug exception instead causes execution to roll back to just before the XBEGIN instruction

and then delivers a #DB. See Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

Interrupt 2—NMI Interrupt

Exception Class **Not applicable.**

Description

The nonmaskable interrupt (NMI) is generated externally by asserting the processor's NMI pin or through an NMI request set by the I/O APIC to the local APIC. This interrupt causes the NMI interrupt handler to be called.

Exception Error Code

Not applicable.

Saved Instruction Pointer

The processor always takes an NMI interrupt on an instruction boundary. The saved contents of CS and EIP registers point to the next instruction to be executed at the point the interrupt is taken. See Section 7.5, "Exception Classifications," for more information about when the processor takes NMI interrupts.

Program State Change

The instruction executing when an NMI interrupt is received is completed before the NMI is generated. A program or task can thus be restarted upon returning from an interrupt handler without loss of continuity, provided the interrupt handler saves the state of the processor before handling the interrupt and restores the processor's state prior to a return.

Interrupt 3—Breakpoint Exception (#BP)

Exception Class **Trap.**

Description

Indicates that a breakpoint instruction (INT3, opcode CC) was executed, causing a breakpoint trap to be generated. Typically, a debugger sets a breakpoint by replacing the first opcode byte of an instruction with the opcode for the INT3 instruction. (The INT3 instruction is one byte long, which makes it easy to replace an opcode in a code segment in RAM with the breakpoint opcode.) The operating system or a debugging tool can use a data segment mapped to the same physical address space as the code segment to place an INT3 instruction in places where it is desired to call the debugger.

With the P6 family, Pentium, Intel486, and Intel386 processors, it is more convenient to set breakpoints with the debug registers. (See Section 19.3.2, “Breakpoint Exception (#BP)—Interrupt Vector 3,” for information about the breakpoint exception.) If more breakpoints are needed beyond what the debug registers allow, the INT3 instruction can be used.

Any breakpoint exception inside an RTM region causes a transactional abort and, by default, redirects control flow to the fallback instruction address. If advanced debugging of RTM transactional regions has been enabled, any transactional abort due to a break exception instead causes execution to roll back to just before the XBEGIN instruction and then delivers a **debug exception (#DB)** — **not** a breakpoint exception. See Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

A breakpoint exception can also be generated by executing the INT *n* instruction with an operand of 3. The action of this instruction (INT 3) is slightly different than that of the INT3 instruction (see “INT *n*/INTO/INT3/INT1—Call to Interrupt Procedure” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A).

Exception Error Code

None.

Saved Instruction Pointer

Saved contents of CS and EIP registers point to the instruction following the INT3 instruction.

Program State Change

Even though the EIP points to the instruction following the breakpoint instruction, the state of the program is essentially unchanged because the INT3 instruction does not affect any register or memory locations. The debugger can thus resume the suspended program by replacing the INT3 instruction that caused the breakpoint with the original opcode and decrementing the saved contents of the EIP register. Upon returning from the debugger, program execution resumes with the replaced instruction.

Interrupt 4—Overflow Exception (#OF)

Exception Class **Trap.**

Description

Indicates that an overflow trap occurred when an INTO instruction was executed. The INTO instruction checks the state of the OF flag in the EFLAGS register. If the OF flag is set, an overflow trap is generated.

Some arithmetic instructions (such as the ADD and SUB) perform both signed and unsigned arithmetic. These instructions set the OF and CF flags in the EFLAGS register to indicate signed overflow and unsigned overflow, respectively. When performing arithmetic on signed operands, the OF flag can be tested directly or the INTO instruction can be used. The benefit of using the INTO instruction is that if the overflow exception is detected, an exception handler can be called automatically to handle the overflow condition.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction following the INTO instruction.

Program State Change

Even though the EIP points to the instruction following the INTO instruction, the state of the program is essentially unchanged because the INTO instruction does not affect any register or memory locations. The program can thus resume normal execution upon returning from the overflow exception handler.

Interrupt 5—BOUND Range Exceeded Exception (#BR)

Exception Class **Fault.**

Description

Indicates that a BOUND-range-exceeded fault occurred when a BOUND instruction was executed. The BOUND instruction checks that a signed array index is within the upper and lower bounds of an array located in memory. If the array index is not within the bounds of the array, a BOUND-range-exceeded fault is generated.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the BOUND instruction that generated the exception.

Program State Change

A program-state change does not accompany the bounds-check fault, because the operands for the BOUND instruction are not modified. Returning from the BOUND-range-exceeded exception handler causes the BOUND instruction to be restarted.

Interrupt 6—Invalid Opcode Exception (#UD)

Exception Class **Fault.**

Description

Indicates that the processor did one of the following things:

- Attempted to execute an invalid or reserved opcode.
- Attempted to execute an instruction with an operand type that is invalid for its accompanying opcode; for example, the source operand for a LES instruction is not a memory location.
- Attempted to execute an MMX or SSE/SSE2/SSE3 instruction on an Intel 64 or IA-32 processor that does not support the MMX technology or SSE/SSE2/SSE3/SSSE3 extensions, respectively. CPUID feature flags MMX (bit 23), SSE (bit 25), SSE2 (bit 26), SSE3 (ECX, bit 0), SSSE3 (ECX, bit 9) indicate support for these extensions.
- Attempted to execute an MMX instruction or SSE/SSE2/SSE3/SSSE3 SIMD instruction (with the exception of the MOVNTI, PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, CLFLUSH, MONITOR, and MWAIT instructions) when the EM flag in control register CR0 is set (1).
- Attempted to execute an SSE/SE2/SSE3/SSSE3 instruction when the OSFXSR bit in control register CR4 is clear (0). Note this does not include the following SSE/SSE2/SSE3 instructions: MASKMOVQ, MOVNTQ, MOVNTI, PREFETCHh, SFENCE, LFENCE, MFENCE, and CLFLUSH; or the 64-bit versions of the PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADB, PSHUFW, PADDQ, PSUBQ, PALIGNR, PABSB, PABSD, PABSW, PHADDD, PHADDSW, PHADDW, PHSUBD, PHSUBSW, PHSUBW, PMADDUSB, PMULHRW, PSHUFB, PSIGNB, PSIGND, and PSIGNW.
- Attempted to execute an SSE/SSE2/SSE3/SSSE3 instruction on an Intel 64 or IA-32 processor that caused a SIMD floating-point exception when the OSXMMEXCPT bit in control register CR4 is clear (0).
- Executed a UD0, UD1, UD2, or UDB instruction. Note that even though it is the execution of the UD0, UD1, UD2, or UDB instruction that causes the invalid opcode exception, the saved instruction pointer will reference the UD0, UD1, UD2, or UDB instruction.
- Detected a LOCK prefix that precedes an instruction that may not be locked or one that may be locked but the destination operand is not a memory location.
- Attempted to execute an LLDT, SLDT, LTR, STR, LSL, LAR, VERR, VERW, or ARPL instruction while in real-address or virtual-8086 mode.
- Attempted to execute the RSM instruction when not in SMM mode.

In Intel 64 and IA-32 processors that implement out-of-order execution microarchitectures, this exception is not generated until an attempt is made to retire the result of executing an invalid instruction; that is, decoding and speculatively attempting to execute an invalid opcode does not generate this exception. Likewise, in the Pentium processor and earlier IA-32 processors, this exception is not generated as the result of prefetching and preliminary decoding of an invalid instruction. (See Section 7.5, “Exception Classifications,” for general rules for taking of interrupts and exceptions.)

The opcodes D6 and F1 are undefined opcodes reserved by the Intel 64 and IA-32 architectures. These opcodes, even though undefined, do not generate an invalid opcode exception.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany an invalid-opcode fault, because the invalid instruction is not executed.

Interrupt 7—Device Not Available Exception (#NM)

Exception Class **Fault.**

Description

Indicates one of the following things:

The device-not-available exception is generated by either of three conditions:

- The processor executed an x87 FPU floating-point instruction while the EM flag in control register CR0 was set (1). See the paragraph below for the special case of the WAIT/FWAIT instruction.
- The processor executed a WAIT/FWAIT instruction while the MP and TS flags of register CR0 were set, regardless of the setting of the EM flag.
- The processor executed an x87 FPU, MMX, or SSE/SSE2/SSE3 instruction (with the exception of MOVNTI, PAUSE, PREFETCHh, SFENCE, LFENCE, MFENCE, and CLFLUSH) while the TS flag in control register CR0 was set and the EM flag is clear.

The EM flag is set when the processor does not have an internal x87 FPU floating-point unit. A device-not-available exception is then generated each time an x87 FPU floating-point instruction is encountered, allowing an exception handler to call floating-point instruction emulation routines.

The TS flag indicates that a context switch (task switch) has occurred since the last time an x87 floating-point, MMX, or SSE/SSE2/SSE3 instruction was executed; but that the context of the x87 FPU, XMM, and MXCSR registers were not saved. When the TS flag is set and the EM flag is clear, the processor generates a device-not-available exception each time an x87 floating-point, MMX, or SSE/SSE2/SSE3 instruction is encountered (with the exception of the instructions listed above). The exception handler can then save the context of the x87 FPU, XMM, and MXCSR registers before it executes the instruction. See Section 2.5, “Control Registers,” for more information about the TS flag.

The MP flag in control register CR0 is used along with the TS flag to determine if WAIT or FWAIT instructions should generate a device-not-available exception. It extends the function of the TS flag to the WAIT and FWAIT instructions, giving the exception handler an opportunity to save the context of the x87 FPU before the WAIT or FWAIT instruction is executed. The MP flag is provided primarily for use with the Intel 286 and Intel386 DX processors. For programs running on the Pentium 4, Intel Xeon, P6 family, Pentium, or Intel486 DX processors, or the Intel 487 SX coprocessors, the MP flag should always be set; for programs running on the Intel486 SX processor, the MP flag should be clear.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the floating-point instruction or the WAIT/FWAIT instruction that generated the exception.

Program State Change

A program-state change does not accompany a device-not-available fault, because the instruction that generated the exception is not executed.

If the EM flag is set, the exception handler can then read the floating-point instruction pointed to by the EIP and call the appropriate emulation routine.

If the MP and TS flags are set or the TS flag alone is set, the exception handler can save the context of the x87 FPU, clear the TS flag, and continue execution at the interrupted floating-point or WAIT/FWAIT instruction.

Interrupt 8—Double Fault Exception (#DF)

Exception Class **Abort.**

Description

Indicates that the processor detected a second exception while calling an exception handler for a prior exception. Normally, when the processor detects another exception while trying to call an exception handler, the two exceptions can be handled serially. If, however, the processor cannot handle them serially, it signals the double-fault exception. To determine when two faults need to be signalled as a double fault, the processor divides the exceptions into three classes: benign exceptions, contributory exceptions, and page faults (see Table 7-4).

Table 7-4. Interrupt and Exception Classes

Class	Vector Number	Description
Benign Exceptions and Interrupts	1	Debug
	2	NMI Interrupt
	3	Breakpoint
	4	Overflow
	5	BOUND Range Exceeded
	6	Invalid Opcode
	7	Device Not Available
	9	Coprocessor Segment Overrun
	16	Floating-Point Error
	17	Alignment Check
	18	Machine Check
	19	SIMD floating-point
	All	INT <i>n</i>
	All	INTR
Contributory Exceptions	0	Divide Error
	10	Invalid TSS
	11	Segment Not Present
	12	Stack Fault
	13	General Protection
Page Faults	21	Control Protection
	14	Page Fault
	20	Virtualization Exception

Table 7-5 shows the various combinations of exception classes that cause a double fault to be generated. A double-fault exception falls in the abort class of exceptions. The program or task cannot be restarted or resumed. The double-fault handler can be used to collect diagnostic information about the state of the machine and/or, when possible, to shut the application and/or system down gracefully or restart the system.

A segment or page fault may be encountered while prefetching instructions; however, this behavior is outside the domain of Table 7-5. Any further faults generated while the processor is attempting to transfer control to the appropriate fault handler could still lead to a double-fault sequence.

Table 7-5. Conditions for Generating a Double Fault

First Exception	Second Exception		
	Benign	Contributory	Page Fault
Benign	Handle Exceptions Serially	Handle Exceptions Serially	Handle Exceptions Serially
Contributory	Handle Exceptions Serially	Generate a Double Fault	Handle Exceptions Serially
Page Fault	Handle Exceptions Serially	Generate a Double Fault	Generate a Double Fault
Double Fault	Handle Exceptions Serially	Enter Shutdown Mode	Enter Shutdown Mode

If another contributory or page fault exception occurs while attempting to call the double-fault handler, the processor enters shutdown mode. This mode is similar to the state following execution of an HLT instruction. In this mode, the processor stops executing instructions until an NMI interrupt, SMI interrupt, hardware reset, or INIT# is received. The processor generates a special bus cycle to indicate that it has entered shutdown mode. Software designers may need to be aware of the response of hardware when it goes into shutdown mode. For example, hardware may turn on an indicator light on the front panel, generate an NMI interrupt to record diagnostic information, invoke reset initialization, generate an INIT initialization, or generate an SMI. If any events are pending during shutdown, they will be handled after an wake event from shutdown is processed (for example, A20M# interrupts).

If a shutdown occurs while the processor is executing an NMI interrupt handler, then only a hardware reset can restart the processor. Likewise, if the shutdown occurs while executing in SMM, a hardware reset must be used to restart the processor.

Exception Error Code

Zero. The processor always pushes an error code of 0 onto the stack of the double-fault handler.

Saved Instruction Pointer

The saved contents of CS and EIP registers are undefined.

Program State Change

A program-state following a double-fault exception is undefined. The program or task cannot be resumed or restarted. The only available action of the double-fault exception handler is to collect all possible context information for use in diagnostics and then close the application and/or shut down or reset the processor.

If the double fault occurs when any portion of the exception handling machine state is corrupted, the handler cannot be invoked and the processor must be reset.

Interrupt 9—Coprocesor Segment Overrun

Exception Class **Abort. (Intel reserved; do not use. Recent IA-32 processors do not generate this exception.)**

Description

Indicates that an Intel386 CPU-based systems with an Intel 387 math coprocessor detected a page or segment violation while transferring the middle portion of an Intel 387 math coprocessor operand. The P6 family, Pentium, and Intel486 processors do not generate this exception; instead, this condition is detected with a general protection exception (#GP), interrupt 13.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state following a coprocessor segment-overrun exception is undefined. The program or task cannot be resumed or restarted. The only available action of the exception handler is to save the instruction pointer and reinitialize the x87 FPU using the FNINIT instruction.

Interrupt 10—Invalid TSS Exception (#TS)

Exception Class **Fault.**

Description

Indicates that there was an error related to a TSS. Such an error might be detected during a task switch or during the execution of instructions that use information from a TSS. Table 7-6 shows the conditions that cause an invalid TSS exception to be generated.

Table 7-6. Invalid TSS Conditions

Error Code Index	Invalid Condition
TSS segment selector index	The TSS segment limit is less than 67H for 32-bit TSS or less than 2CH for 16-bit TSS.
TSS segment selector index	During an IRET task switch, the TI flag in the TSS segment selector indicates the LDT.
TSS segment selector index	During an IRET task switch, the TSS segment selector exceeds descriptor table limit.
TSS segment selector index	During an IRET task switch, the busy flag in the TSS descriptor indicates an inactive task.
TSS segment selector index	During a task switch, an attempt to access data in a TSS results in a limit violation or canonical fault.
TSS segment selector index	During an IRET task switch, the backlink is a NULL selector.
TSS segment selector index	During an IRET task switch, the backlink points to a descriptor which is not a busy TSS.
TSS segment selector index	The new TSS descriptor is beyond the GDT limit.
TSS segment selector index	The new TSS selector is null on an attempt to lock the new TSS.
TSS segment selector index	The new TSS selector has the TI bit set on an attempt to lock the new TSS.
TSS segment selector index	The new TSS descriptor is not an available TSS descriptor on an attempt to lock the new TSS.
LDT segment selector index	LDT not valid or not present.
Stack segment selector index	The stack segment selector exceeds descriptor table limit.
Stack segment selector index	The stack segment selector is NULL.
Stack segment selector index	The stack segment descriptor is a non-data segment.
Stack segment selector index	The stack segment is not writable.
Stack segment selector index	The stack segment DPL ? CPL.
Stack segment selector index	The stack segment selector RPL ? CPL.
Code segment selector index	The code segment selector exceeds descriptor table limit.
Code segment selector index	The code segment selector is NULL.
Code segment selector index	The code segment descriptor is not a code segment type.
Code segment selector index	The nonconforming code segment DPL ? CPL.
Code segment selector index	The conforming code segment DPL is greater than CPL.
Data segment selector index	The data segment selector exceeds the descriptor table limit.
Data segment selector index	The data segment descriptor is not a readable code or data type.
Data segment selector index	The data segment descriptor is a nonconforming code type and RPL > DPL.
Data segment selector index	The data segment descriptor is a nonconforming code type and CPL > DPL.
TSS segment selector index	The TSS segment descriptor/upper descriptor is beyond the GDT segment limit.
TSS segment selector index	The TSS segment descriptor is not an available TSS type.
TSS segment selector index	The TSS segment descriptor is an available 286 TSS type in IA-32e mode.

Table 7-6. Invalid TSS Conditions (Contd.)

Error Code Index	Invalid Condition
TSS segment selector index	The TSS segment upper descriptor is not the correct type.
TSS segment selector index	The TSS segment descriptor contains a non-canonical base.

This exception can be generated either in the context of the original task or in the context of the new task (see Section 9.3, “Task Switching”). Until the processor has completely verified the presence of the new TSS, the exception is generated in the context of the original task. Once the existence of the new TSS is verified, the task switch is considered complete. Any invalid-TSS conditions detected after this point are handled in the context of the new task. (A task switch is considered complete when the task register is loaded with the segment selector for the new TSS and, if the switch is due to a procedure call or interrupt, the previous task link field of the new TSS references the old TSS.)

The invalid-TSS handler must be a task called using a task gate. Handling this exception inside the faulting TSS context is not recommended because the processor state may not be consistent.

Exception Error Code

An error code containing the segment selector index for the segment descriptor that caused the violation is pushed onto the stack of the exception handler. If the EXT flag is set, it indicates that the exception was caused by an event external to the currently running program (for example, if an external interrupt handler using a task gate attempted a task switch to an invalid TSS).

Saved Instruction Pointer

If the exception condition was detected before the task switch was carried out, the saved contents of CS and EIP registers point to the instruction that invoked the task switch. If the exception condition was detected after the task switch was carried out, the saved contents of CS and EIP registers point to the first instruction of the new task.

Program State Change

The ability of the invalid-TSS handler to recover from the fault depends on the error condition that causes the fault. See Section 9.3, “Task Switching,” for more information on the task switch process and the possible recovery actions that can be taken.

If an invalid TSS exception occurs during a task switch, it can occur before or after the commit-to-new-task point. If it occurs before the commit point, no program state change occurs. If it occurs after the commit point (when the segment descriptor information for the new segment selectors have been loaded in the segment registers), the processor will load all the state information from the new TSS before it generates the exception. During a task switch, the processor first loads all the segment registers with segment selectors from the TSS, then checks their contents for validity. If an invalid TSS exception is discovered, the remaining segment registers are loaded but not checked for validity and therefore may not be usable for referencing memory. The invalid TSS handler should not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. The exception handler should load all segment registers before trying to resume the new task; otherwise, general-protection exceptions (#GP) may result later under conditions that make diagnosis more difficult. The Intel recommended way of dealing with this situation is to use a task for the invalid TSS exception handler. The task switch back to the interrupted task from the invalid-TSS exception-handler task will then cause the processor to check the registers as it loads them from the TSS.

Interrupt 11—Segment Not Present (#NP)

Exception Class **Fault.**

Description

Indicates that the present flag of a segment or gate descriptor is clear. The processor can generate this exception during any of the following operations:

- While attempting to load CS, DS, ES, FS, or GS registers. [Detection of a not-present segment while loading the SS register causes a stack fault exception (#SS) to be generated.] This situation can occur while performing a task switch.
- While attempting to load the LDTR using an LLDT instruction. Detection of a not-present LDT while loading the LDTR during a task switch operation causes an invalid-TSS exception (#TS) to be generated.
- When executing the LTR instruction and the TSS is marked not present.
- While attempting to use a gate descriptor or TSS that is marked segment-not-present, but is otherwise valid.

An operating system typically uses the segment-not-present exception to implement virtual memory at the segment level. If the exception handler loads the segment and returns, the interrupted program or task resumes execution.

A not-present indication in a gate descriptor, however, does not indicate that a segment is not present (because gates do not correspond to segments). The operating system may use the present flag for gate descriptors to trigger exceptions of special significance to the operating system.

A contributory exception or page fault that subsequently referenced a not-present segment would cause a double fault (#DF) to be generated instead of #NP.

Exception Error Code

An error code containing the segment selector index for the segment descriptor that caused the violation is pushed onto the stack of the exception handler. If the EXT flag is set, it indicates that the exception resulted from either:

- an external event (NMI or INTR) that caused an interrupt, which subsequently referenced a not-present segment
- a benign exception that subsequently referenced a not-present segment

The IDT flag is set if the error code refers to an IDT entry. This occurs when the IDT entry for an interrupt being serviced references a not-present gate. Such an event could be generated by an INT instruction or a hardware interrupt.

Saved Instruction Pointer

The saved contents of CS and EIP registers normally point to the instruction that generated the exception. If the exception occurred while loading segment descriptors for the segment selectors in a new TSS, the CS and EIP registers point to the first instruction in the new task. If the exception occurred while accessing a gate descriptor, the CS and EIP registers point to the instruction that invoked the access (for example a CALL instruction that references a call gate).

Program State Change

If the segment-not-present exception occurs as the result of loading a register (CS, DS, SS, ES, FS, GS, or LDTR), a program-state change does accompany the exception because the register is not loaded. Recovery from this exception is possible by simply loading the missing segment into memory and setting the present flag in the segment descriptor.

If the segment-not-present exception occurs while accessing a gate descriptor, a program-state change does not accompany the exception. Recovery from this exception is possible merely by setting the present flag in the gate descriptor.

If a segment-not-present exception occurs during a task switch, it can occur before or after the commit-to-new-task point (see Section 9.3, “Task Switching”). If it occurs before the commit point, no program state change

occurs. If it occurs after the commit point, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The segment-not-present exception handler should not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for “Interrupt 10—Invalid TSS Exception (#TS)” in this chapter for additional information on how to handle this situation.)

Interrupt 12—Stack Fault Exception (#SS)

Exception Class **Fault.**

Description

Indicates that one of the following stack related conditions was detected:

- A limit violation is detected during an operation that refers to the SS register. Operations that can cause a limit violation include stack-oriented instructions such as POP, PUSH, CALL, RET, IRET, ENTER, and LEAVE, as well as other memory references which implicitly or explicitly use the SS register (for example, MOV AX, [BP+6] or MOV AX, SS:[EAX+6]). The ENTER instruction generates this exception when there is not enough stack space for allocating local variables.
- A not-present stack segment is detected when attempting to load the SS register. This violation can occur during the execution of a task switch, a CALL instruction to a different privilege level, a return to a different privilege level, an LSS instruction, or a MOV or POP instruction to the SS register.
- A canonical violation is detected in 64-bit mode during an operation that reference memory using the stack pointer register containing a non-canonical memory address.

Recovery from this fault is possible by either extending the limit of the stack segment (in the case of a limit violation) or loading the missing stack segment into memory (in the case of a not-present violation).

In the case of a canonical violation that was caused intentionally by software, recovery is possible by loading the correct canonical value into RSP. Otherwise, a canonical violation of the address in RSP likely reflects some register corruption in the software.

Exception Error Code

If the exception is caused by a not-present stack segment or by overflow of the new stack during an inter-privilege-level call, the error code contains a segment selector for the segment that caused the exception. Here, the exception handler can test the present flag in the segment descriptor pointed to by the segment selector to determine the cause of the exception. For a normal limit violation (on a stack segment already in use) the error code is set to 0.

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception. However, when the exception results from attempting to load a not-present stack segment during a task switch, the CS and EIP registers point to the first instruction of the new task.

Program State Change

A program-state change does not generally accompany a stack-fault exception, because the instruction that generated the fault is not executed. Here, the instruction can be restarted after the exception handler has corrected the stack fault condition.

If a stack fault occurs during a task switch, it occurs after the commit-to-new-task point (see Section 9.3, “Task Switching”). Here, the processor loads all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The stack fault handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. The exception handler should check all segment registers before trying to resume the new task; otherwise, general protection faults may result later under conditions that are more difficult to diagnose. (See the Program State Change description for “Interrupt 10—Invalid TSS Exception (#TS)” in this chapter for additional information on how to handle this situation.)

Interrupt 13—General Protection Exception (#GP)

Exception Class **Fault.**

Description

Indicates that the processor detected one of a class of protection violations called “general-protection violations.” The conditions that cause this exception to be generated comprise all the protection violations that do not cause other exceptions to be generated (such as, invalid-TSS, segment-not-present, stack-fault, or page-fault exceptions). The following conditions cause general-protection exceptions to be generated:

- Exceeding the segment limit when accessing the CS, DS, ES, FS, or GS segments.
- Exceeding the segment limit when referencing a descriptor table (except during a task switch or a stack switch).
- Transferring execution to a segment that is not executable.
- Writing to a code segment or a read-only data segment.
- Reading from an execute-only code segment.
- Loading the SS register with a segment selector for a read-only segment (unless the selector comes from a TSS during a task switch, in which case an invalid-TSS exception occurs).
- Loading the SS, DS, ES, FS, or GS register with a segment selector for a system segment.
- Loading the DS, ES, FS, or GS register with a segment selector for an execute-only code segment.
- Loading the SS register with the segment selector of an executable segment or a null segment selector.
- Loading the CS register with a segment selector for a data segment or a null segment selector.
- Accessing memory using the DS, ES, FS, or GS register when it contains a null segment selector.
- Switching to a busy task during a call or jump to a TSS.
- Using a segment selector on a non-IRET task switch that points to a TSS descriptor in the current LDT. TSS descriptors can only reside in the GDT. This condition causes a #TS exception during an IRET task switch.
- Violating any of the privilege rules described in Chapter 6, “Protection.”
- Exceeding the instruction length limit of 15 bytes (this only can occur when redundant prefixes are placed before an instruction).
- Loading the CR0 register with a set PG flag (paging enabled) and a clear PE flag (protection disabled).
- Loading the CR0 register with a set NW flag and a clear CD flag.
- Referencing an entry in the IDT (following an interrupt or exception) that is not an interrupt, trap, or task gate.
- Attempting to access an interrupt or exception handler through an interrupt or trap gate from virtual-8086 mode when the handler’s code segment DPL is greater than 0.
- Attempting to write a 1 into a reserved bit of CR4.
- Attempting to execute a privileged instruction when the CPL is not equal to 0 (see Section 6.9, “Privileged Instructions,” for a list of privileged instructions).
- Attempting to execute SGDT, SIDT, SLDT, SMSW, or STR when CR4.UMIP = 1 and the CPL is not equal to 0.
- Writing to a reserved bit in an MSR.
- Accessing a gate that contains a null segment selector.
- Executing the INT *n* instruction when the CPL is greater than the DPL of the referenced interrupt, trap, or task gate.
- The segment selector in a call, interrupt, or trap gate does not point to a code segment.
- The segment selector operand in the LLDT instruction is a local type (TI flag is set) or does not point to a segment descriptor of the LDT type.
- The segment selector operand in the LTR instruction is local or points to a TSS that is not available.
- The target code-segment selector for a call, jump, or return is null.

- If the PAE and/or PSE flag in control register CR4 is set and the processor detects any reserved bits in a page-directory-pointer-table entry set to 1. These bits are checked during a write to control registers CR0, CR3, or CR4 that causes a reloading of the page-directory-pointer-table entry.
- Attempting to write a non-zero value into the reserved bits of the MXCSR register.
- Executing an SSE/SSE2/SSE3 instruction that attempts to access a 128-bit memory location that is not aligned on a 16-byte boundary when the instruction requires 16-byte alignment. This condition also applies to the stack segment.

A program or task can be restarted following any general-protection exception. If the exception occurs while attempting to call an interrupt handler, the interrupted program can be restartable, but the interrupt may be lost.

Exception Error Code

The processor pushes an error code onto the exception handler's stack. If the fault condition was detected while loading a segment descriptor, the error code contains a segment selector to or IDT vector number for the descriptor; otherwise, the error code is 0. The source of the selector in an error code may be any of the following:

- An operand of the instruction.
- A selector from a gate which is the operand of the instruction.
- A selector from a TSS involved in a task switch.
- IDT vector number.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

In general, a program-state change does not accompany a general-protection exception, because the invalid instruction or operation is not executed. An exception handler can be designed to correct all of the conditions that cause general-protection exceptions and restart the program or task without any loss of program continuity.

If a general-protection exception occurs during a task switch, it can occur before or after the commit-to-new-task point (see Section 9.3, "Task Switching"). If it occurs before the commit point, no program state change occurs. If it occurs after the commit point, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The general-protection exception handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for "Interrupt 10—Invalid TSS Exception (#TS)" in this chapter for additional information on how to handle this situation.)

General Protection Exception in 64-bit Mode

The following conditions cause general-protection exceptions in 64-bit mode:

- If the memory address is in a non-canonical form.
- If a segment descriptor memory address is in non-canonical form.
- If the target offset in a destination operand of a call or jmp is in a non-canonical form.
- If a code segment or 64-bit call gate overlaps non-canonical space.
- If the code segment descriptor pointed to by the selector in the 64-bit gate doesn't have the L-bit set and the D-bit clear.
- If the EFLAGS.NT bit is set in IRET.
- If the stack segment selector of IRET is null when going back to compatibility mode.
- If the stack segment selector of IRET is null going back to CPL3 and 64-bit mode.
- If a null stack segment selector RPL of IRET is not equal to CPL going back to non-CPL3 and 64-bit mode.
- If the proposed new code segment descriptor of IRET has both the D-bit and the L-bit set.

- If the segment descriptor pointed to by the segment selector in the destination operand is a code segment and it has both the D-bit and the L-bit set.
- If the segment descriptor from a 64-bit call gate is in non-canonical space.
- If the DPL from a 64-bit call-gate is less than the CPL or than the RPL of the 64-bit call-gate.
- If the type field of the upper 64 bits of a 64-bit call gate is not 0.
- If an attempt is made to load a null selector in the SS register in compatibility mode.
- If an attempt is made to load null selector in the SS register in CPL3 and 64-bit mode.
- If an attempt is made to load a null selector in the SS register in non-CPL3 and 64-bit mode where RPL is not equal to CPL.
- If an attempt is made to clear CR0.PG while IA-32e mode is enabled.
- If an attempt is made to set a reserved bit in CR3, CR4 or CR8.

Interrupt 14—Page-Fault Exception (#PF)

Exception Class **Fault.**

Description

Indicates that, with paging enabled (the PG flag in the CR0 register is set), the processor detected one of the following conditions while using the page-translation mechanism to translate a linear address to a physical address:

- The P (present) flag in a page-directory or page-table entry needed for the address translation is clear, indicating that a page table or the page containing the operand is not present in physical memory.
- The procedure does not have sufficient privilege to access the indicated page (that is, a procedure running in user mode attempts to access a supervisor-mode page). If the SMAP flag is set in CR4, a page fault may also be triggered by code running in supervisor mode that tries to access data at a user-mode address. If either the PKE flag or the PKS flag is set in CR4, the protection-key rights registers may cause page faults on data accesses to linear addresses with certain protection keys.
- Code running in user mode attempts to write to a read-only page. If the WP flag is set in CR0, the page fault will also be triggered by code running in supervisor mode that tries to write to a read-only page.
- An instruction fetch to a linear address that translates to a physical address in a memory page with the execute-disable bit set (for information about the execute-disable bit, see Chapter 5, “Paging”). If the SMEP flag is set in CR4, a page fault will also be triggered by code running in supervisor mode that tries to fetch an instruction from a user-mode address.
- One or more reserved bits in paging-structure entry are set to 1. See description below of RSVD error code flag.
- A shadow-stack access is made to a page that is not a shadow-stack page. See Section 18.2, “Shadow Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, and Section 5.6, “Access Rights.”
- An enclave access violates one of the specified access-control requirements. See Section 37.3, “Access-control Requirements,” and Section 37.20, “Enclave Page Cache Map (EPCM),” in Chapter 37, “Enclave Access Control and Data Structures.” In this case, the exception is called an **SGX-induced page fault**. The processor uses the error code (below) to distinguish SGX-induced page faults from ordinary page faults.

The exception handler can recover from page-not-present conditions and restart the program or task without any loss of program continuity. It can also restart the program or task after a privilege violation, but the problem that caused the privilege violation may be uncorrectable.

See also: Section 5.7, “Page-Fault Exceptions.”

Exception Error Code

Yes (special format). The processor provides the page-fault handler with two items of information to aid in diagnosing the exception and recovering from it:

- An error code on the stack. The error code for a page fault has a format different from that for other exceptions (see Figure 7-11). The processor establishes the bits in the error code as follows:
 - P flag (bit 0).
This flag is 0 if there is no translation for the linear address because the P flag was 0 in one of the paging-structure entries used to translate that address.
 - W/R (bit 1).
If the access causing the page-fault exception was a write, this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
 - U/S (bit 2).
If a user-mode access caused the page-fault exception, this flag is 1; it is 0 if a supervisor-mode access did so. This flag describes the access causing the page-fault exception, not the access rights specified by paging.

- RSVD flag (bit 3).
This flag is 1 if there is no translation for the linear address because a reserved bit was set in one of the paging-structure entries used to translate that address.
- I/D flag (bit 4).
This flag is 1 if the access causing the page-fault exception was an instruction fetch. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- PK flag (bit 5).
This flag is 1 if the access causing the page-fault exception was a data access to a linear address with a protection key for which the protection-key rights registers disallow access.
- SS (bit 6).
If the access causing the page-fault exception was a shadow-stack access (including shadow-stack accesses in enclave mode), this flag is 1; otherwise, it is 0. This flag describes the access causing the page-fault exception, not the access rights specified by paging.
- HLAT (bit 7).
This flag is 1 if there is no translation for the linear address using HLAT paging because, in one of the paging-structure entries used to translate that address, either the P flag was 0 or a reserved bit was set. An error code will set this flag only if it clears bit 0 or sets bit 3. This flag will not be set by a page fault resulting from a violation of access rights, nor for one encountered during ordinary paging, including the case in which there has been a restart of HLAT paging.
- SGX flag (bit 15).
This flag is 1 if the exception is unrelated to paging and resulted from violation of SGX-specific access-control requirements. Because such a violation can occur only if there is no ordinary page fault, this flag is set only if the P flag (bit 0) is 1 and the RSVD flag (bit 3) and the PK flag (bit 5) are both 0.

See Section 5.6, “Access Rights,” and Section 5.7, “Page-Fault Exceptions,” for more information about page-fault exceptions and the error codes that they produce.

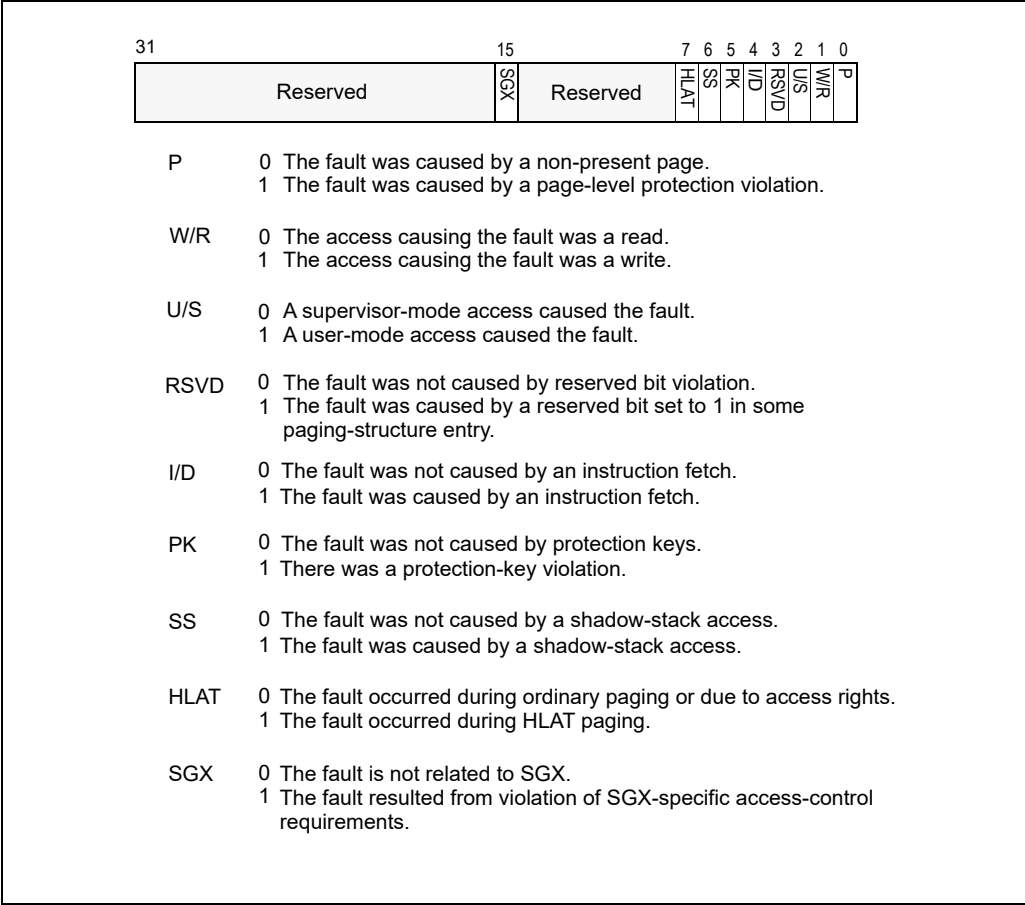


Figure 7-11. Page-Fault Error Code

- The contents of the CR2 register. The processor loads the CR2 register with the linear address that generated the exception. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata. If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the address are cleared.
The page-fault handler can use this address to locate the corresponding paging-structure entries. Another page fault can potentially occur during execution of the page-fault handler; the handler should save the contents of the CR2 register before a second page fault can occur.¹ If a page fault is caused by a page-level protection violation, the accessed flags in paging-structure entries may be set when the fault occurs (behavior is model-specific and not architecturally defined).

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception. If the page-fault exception occurred during a task switch, the CS and EIP registers may point to the first instruction of the new task (as described in the following “Program State Change” section).

1. Processors update CR2 whenever a page fault is detected. If a second page fault occurs while an earlier page fault is being delivered, the faulting linear address of the second fault will overwrite the contents of CR2 (replacing the previous address). These updates to CR2 occur even if the page fault results in a double fault or occurs during the delivery of a double fault.

Program State Change

A program-state change does not normally accompany a page-fault exception, because the instruction that causes the exception to be generated is not executed. After the page-fault exception handler has corrected the violation (for example, loaded the missing page into memory), execution of the program or task can be resumed.

When a page-fault exception is generated during a task switch, the program-state may change, as follows. During a task switch, a page-fault exception can occur during any of following operations:

- While writing the state of the original task into the TSS of that task.
- While reading the GDT to locate the TSS descriptor of the new task.
- While reading the TSS of the new task.
- While reading segment descriptors associated with segment selectors from the new task.
- While reading the LDT of the new task to verify the segment registers stored in the new TSS.

In the last two cases the exception occurs in the context of the new task. The instruction pointer refers to the first instruction of the new task, not to the instruction which caused the task switch (or the last instruction to be executed, in the case of an interrupt). If the design of the operating system permits page faults to occur during task-switches, the page-fault handler should be called through a task gate.

If a page fault occurs during a task switch, the processor will load all the state information from the new TSS (without performing any additional limit, present, or type checks) before it generates the exception. The page-fault handler should thus not rely on being able to use the segment selectors found in the CS, SS, DS, ES, FS, and GS registers without causing another exception. (See the Program State Change description for “Interrupt 10—Invalid TSS Exception (#TS)” in this chapter for additional information on how to handle this situation.)

Additional Exception-Handling Information

Special care should be taken to ensure that an exception that occurs during an explicit stack switch does not cause the processor to use an invalid stack pointer (SS:ESP). Software written for 16-bit IA-32 processors often use a pair of instructions to change to a new stack, for example:

```
MOV SS, AX
MOV SP, StackTop
```

When executing this code on one of the 32-bit IA-32 processors, it is possible to get a page fault, general-protection fault (#GP), or alignment check fault (#AC) after the segment selector has been loaded into the SS register but before the ESP register has been loaded. At this point, the two parts of the stack pointer (SS and ESP) are inconsistent. The new stack segment is being used with the old stack pointer.

The processor does not use the inconsistent stack pointer if the exception handler switches to a well defined stack (that is, the handler is a task or a more privileged procedure). However, if the exception handler is called at the same privilege level and from the same task, the processor will attempt to use the inconsistent stack pointer.

In systems that handle page-fault, general-protection, or alignment check exceptions within the faulting task (with trap or interrupt gates), software executing at the same privilege level as the exception handler should initialize a new stack by using the LSS instruction rather than a pair of MOV instructions, as described earlier in this note. When the exception handler is running at privilege level 0 (the normal case), the problem is limited to procedures or tasks that run at privilege level 0, typically the kernel of the operating system.

Interrupt 16—x87 FPU Floating-Point Error (#MF)

Exception Class **Fault.**

Description

Indicates that the x87 FPU has detected a floating-point error. The NE flag in the register CR0 must be set for an interrupt 16 (floating-point error exception) to be generated. (See Section 2.5, “Control Registers,” for a detailed description of the NE flag.)

NOTE

SIMD floating-point exceptions (#XM) are signaled through interrupt 19.

While executing x87 FPU instructions, the x87 FPU detects and reports six types of floating-point error conditions:

- Invalid operation (#I)
 - Stack overflow or underflow (#IS)
 - Invalid arithmetic operation (#IA)
- Divide-by-zero (#Z)
- Denormalized operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (precision) (#P)

Each of these error conditions represents an x87 FPU exception type, and for each of exception type, the x87 FPU provides a flag in the x87 FPU status register and a mask bit in the x87 FPU control register. If the x87 FPU detects a floating-point error and the mask bit for the exception type is set, the x87 FPU handles the exception automatically by generating a predefined (default) response and continuing program execution. The default responses have been designed to provide a reasonable result for most floating-point applications.

If the mask for the exception is clear and the NE flag in register CR0 is set, the x87 FPU does the following:

1. Sets the necessary flag in the FPU status register.
2. Waits until the next “waiting” x87 FPU instruction or WAIT/FWAIT instruction is encountered in the program’s instruction stream.
3. Generates an internal error signal that cause the processor to generate a floating-point exception (#MF).

Prior to executing a waiting x87 FPU instruction or the WAIT/FWAIT instruction, the x87 FPU checks for pending x87 FPU floating-point exceptions (as described in step 2 above). Pending x87 FPU floating-point exceptions are ignored for “non-waiting” x87 FPU instructions, which include the FNINIT, FNCLEX, FNSTSW, FNSTSW AX, FNSTCW, FNSTENV, and FNSAVE instructions. Pending x87 FPU exceptions are also ignored when executing the state management instructions FXSAVE and FXRSTOR.

All of the x87 FPU floating-point error conditions can be recovered from. The x87 FPU floating-point-error exception handler can determine the error condition that caused the exception from the settings of the flags in the x87 FPU status word. See “Software Exception Handling” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information on handling x87 FPU floating-point exceptions.

Exception Error Code

None. The x87 FPU provides its own error information.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the floating-point or WAIT/FWAIT instruction that was about to be executed when the floating-point-error exception was generated. This is not the faulting instruction in which the error condition was detected. The address of the faulting instruction is contained in the x87 FPU instruction pointer

register. See Section 8.1.8, “x87 FPU Instruction and Data (Operand) Pointers,” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about information the FPU saves for use in handling floating-point-error exceptions.

Program State Change

A program-state change generally accompanies an x87 FPU floating-point exception because the handling of the exception is delayed until the next waiting x87 FPU floating-point or WAIT/FWAIT instruction following the faulting instruction. The x87 FPU, however, saves sufficient information about the error condition to allow recovery from the error and re-execution of the faulting instruction if needed.

In situations where non- x87 FPU floating-point instructions depend on the results of an x87 FPU floating-point instruction, a WAIT or FWAIT instruction can be inserted in front of a dependent instruction to force a pending x87 FPU floating-point exception to be handled before the dependent instruction is executed. See “x87 FPU Exception Synchronization” in Chapter 8 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information about synchronization of x87 floating-point-error exceptions.

Interrupt 17—Alignment Check Exception (#AC)

Exception Class **Fault.**

Description

There are two causes of alignment-check exceptions: alignment violations and bus-lock violations.

Alignment Violations

An **alignment violation** occurs when the processor detects an unaligned memory operand when alignment checking was enabled. Alignment checks are only carried out in data (or stack) accesses (not in code fetches or system segment accesses). An example of an alignment violation is a word stored at an odd byte address, or a doubleword stored at an address that is not an integer multiple of 4. Table 7-7 lists the alignment requirements various data types recognized by the processor.

Table 7-7. Alignment Requirements by Data Type

Data Type	Address Must Be Divisible By
Word	2
Doubleword	4
Single precision floating-point (32-bits)	4
Double precision floating-point (64-bits)	8
Double extended precision floating-point (80-bits)	8
Quadword	8
Double quadword	16
Segment Selector	2
32-bit Far Pointer	2
48-bit Far Pointer	4
32-bit Pointer	4
GDTR, IDTR, LDTR, or Task Register Contents	4
FSTENV/FLDENV Save Area	4 or 2, depending on operand size
FSAVE/FRSTOR Save Area	4 or 2, depending on operand size
Bit String	2 or 4 depending on the operand-size attribute.

Note that an alignment violation occurs only for data types that must be aligned on word, doubleword, and quadword boundaries. A general-protection exception (#GP) is generated 128-bit data types that are not aligned on a 16-byte boundary.

To enable alignment checking, the following conditions must be true:

- AM flag in CR0 register is set.
- AC flag in the EFLAGS register is set.
- The CPL is 3 (including virtual-8086 mode).

Alignment violations are generated only when operating at privilege level 3 (user mode). Memory references that default to privilege level 0, such as segment descriptor loads, do not generate alignment violations, even when caused by a memory reference made from privilege level 3.

Storing the contents of the GDTR, IDTR, LDTR, or task register in memory while at privilege level 3 can generate an alignment violation. Although application programs do not normally store these registers, the fault can be avoided by aligning the information stored on an even word-address.

The FXSAVE/XSAVE and FXRSTOR/XRSTOR instructions save and restore a 512-byte data structure, the first byte of which must be aligned on a 16-byte boundary. If alignment violations are enabled when executing these instruc-

tions (and CPL is 3), a misaligned memory operand can cause either an alignment violation or a general-protection exception (#GP) depending on the processor implementation (see “FXSAVE—Save x87 FPU, MMX, SSE, and SSE2 State” and “FXRSTOR—Restore x87 FPU, MMX, SSE, and SSE2 State” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A; see “XSAVE—Save Processor Extended States” and “XRSTOR—Restore Processor Extended States” in Chapter 6 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D).

The MOVDQU, MOVUPS, and MOVUPD instructions perform 128-bit unaligned loads or stores. The LDDQU instructions loads 128-bit unaligned data. They do not generate general-protection exceptions (#GP) when operands are not aligned on a 16-byte boundary. If alignment checking is enabled, alignment violations may or may not be generated depending on processor implementation when data addresses are not aligned on an 8-byte boundary.

FSAVE and FRSTOR instructions can generate unaligned references, which can cause alignment violations. These instructions are rarely needed by application programs.

Bus -Lock Violations

Some processors include features that disable bus locks. Section 10.1.2.3 provides details. When these features are enabled, occurrence of a bus lock causes a bus-lock violation, leading to a fault. In some cases, the fault is delivered as an alignment-check exception (#AC). The following are the cases in which bus-lock violation leads to an #AC:

- Split-lock disable is enabled (because MSR_MEMORY_CTRL[29] = 1) and locked access to multiple cache lines occurs (a split lock).
- Bus-lock disabled is enabled (because MSR_MEMORY_CTRL[28] = 1), CPUID.07H.02H:EDX[6] is enumerated as 1 (indicating support for the architectural form of bus-lock disable), and a locked access using a memory type other than WB occurs (a UC lock).¹

Exception Error Code

Yes. For alignment violations and bus-lock violations due to split locks, the error code is null, meaning that bits 31:2 of the error code are clear. For bus-lock violations due to UC locks, the error code has value 4, meaning that bit 2 is set.

In either case, bit 0 (EXT) is set if the violation is recognized during delivery of an event other than a software interrupt. (See Section 7.13, “Error Code” and “INT n/INTO/INT3/INT1—Call to Interrupt Procedure” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.) In such cases, the actual error code delivered will have value 1 or value 5.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that generated the exception.

Program State Change

A program-state change does not accompany an alignment-check fault, because the instruction is not executed.

1. If CPUID.07H.02H:EDX[6] is enumerated as 0, the processor may support an older model-specific form of bus-lock disable. That form generates a general-protection exception (#GP) and not an alignment-check exception.

Interrupt 18—Machine-Check Exception (#MC)

Exception Class **Abort.**

Description

Indicates that the processor detected an internal machine error or a bus error, or that an external agent detected a bus error. The machine-check exception is model-specific, available on the Pentium and later generations of processors. The implementation of the machine-check exception is different between different processor families, and these implementations may not be compatible with future Intel 64 or IA-32 processors. (Use the CPUID instruction to determine whether this feature is present.)

Bus errors detected by external agents are signaled to the processor on dedicated pins: the BINIT# and MCERR# pins on the Pentium 4, Intel Xeon, and P6 family processors and the BUSCHK# pin on the Pentium processor. When one of these pins is enabled, asserting the pin causes error information to be loaded into machine-check registers and a machine-check exception is generated.

The machine-check exception and machine-check architecture are discussed in detail in Chapter 17, “Machine-Check Architecture.” Also, see the data books for the individual processors for processor-specific hardware information.

Exception Error Code

None. Error information is provided by machine-check MSRs.

Saved Instruction Pointer

For the Pentium 4 and Intel Xeon processors, the saved contents of extended machine-check state registers are directly associated with the error that caused the machine-check exception to be generated (see Section 17.3.1.2, “IA32_MCG_STATUS MSR,” and Section 17.3.2.6, “IA32_MCG Extended Machine Check State MSRs”).

For the P6 family processors, if the EIPV flag in the MCG_STATUS MSR is set, the saved contents of CS and EIP registers are directly associated with the error that caused the machine-check exception to be generated; if the flag is clear, the saved instruction pointer may not be associated with the error (see Section 17.3.1.2, “IA32_MCG_STATUS MSR”).

For the Pentium processor, contents of the CS and EIP registers may not be associated with the error.

Program State Change

The machine-check mechanism is enabled by setting the MCE flag in control register CR4.

For the Pentium 4, Intel Xeon, P6 family, and Pentium processors, a program-state change always accompanies a machine-check exception, and an abort class exception is generated. For abort exceptions, information about the exception can be collected from the machine-check MSRs, but the program cannot generally be restarted.

If the machine-check mechanism is not enabled (the MCE flag in control register CR4 is clear), a machine-check exception causes the processor to enter the shutdown state.

Interrupt 19—SIMD Floating-Point Exception (#XM)

Exception Class **Fault.**

Description

Indicates the processor has detected an SSE/SSE2/SSE3 SIMD floating-point exception. The appropriate status flag in the MXCSR register must be set and the particular exception unmasked for this interrupt to be generated.

There are six classes of numeric exception conditions that can occur while executing an SSE/ SSE2/SSE3 SIMD floating-point instruction:

- Invalid operation (#I)
- Divide-by-zero (#Z)
- Denormal operand (#D)
- Numeric overflow (#O)
- Numeric underflow (#U)
- Inexact result (Precision) (#P)

The invalid operation, divide-by-zero, and denormal-operand exceptions are pre-computation exceptions; that is, they are detected before any arithmetic operation occurs. The numeric underflow, numeric overflow, and inexact result exceptions are post-computational exceptions.

See “SIMD Floating-Point Exceptions” in Chapter 11 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for additional information about the SIMD floating-point exception classes.

When a SIMD floating-point exception occurs, the processor does either of the following things:

- It handles the exception automatically by producing the most reasonable result and allowing program execution to continue undisturbed. This is the response to masked exceptions.
- It generates a SIMD floating-point exception, which in turn invokes a software exception handler. This is the response to unmasked exceptions.

Each of the six SIMD floating-point exception conditions has a corresponding flag bit and mask bit in the MXCSR register. If an exception is masked (the corresponding mask bit in the MXCSR register is set), the processor takes an appropriate automatic default action and continues with the computation. If the exception is unmasked (the corresponding mask bit is clear) and the operating system supports SIMD floating-point exceptions (the OSXM-MEXCPT flag in control register CR4 is set), a software exception handler is invoked through a SIMD floating-point exception. If the exception is unmasked and the OSXMMEXCPT bit is clear (indicating that the operating system does not support unmasked SIMD floating-point exceptions), an invalid opcode exception (#UD) is signaled instead of a SIMD floating-point exception.

Note that because SIMD floating-point exceptions are precise and occur immediately, the situation does not arise where an x87 FPU instruction, a WAIT/FWAIT instruction, or another SSE/SSE2/SSE3 instruction will catch a pending unmasked SIMD floating-point exception.

In situations where a SIMD floating-point exception occurred while the SIMD floating-point exceptions were masked (causing the corresponding exception flag to be set) and the SIMD floating-point exception was subsequently unmasked, then no exception is generated when the exception is unmasked.

When SSE/SSE2/SSE3 SIMD floating-point instructions operate on packed operands (made up of two or four sub-operands), multiple SIMD floating-point exception conditions may be detected. If no more than one exception condition is detected for one or more sets of sub-operands, the exception flags are set for each exception condition detected. For example, an invalid exception detected for one sub-operand will not prevent the reporting of a divide-by-zero exception for another sub-operand. However, when two or more exceptions conditions are generated for one sub-operand, only one exception condition is reported, according to the precedences shown in Table 7-8. This exception precedence sometimes results in the higher priority exception condition being reported and the lower priority exception conditions being ignored.

Table 7-8. SIMD Floating-Point Exceptions Priority

Priority	Description
1 (Highest)	Invalid operation exception due to SNaN operand (or any NaN operand for maximum, minimum, or certain compare and convert operations).
2	QNaN operand ¹ .
3	Any other invalid operation exception not mentioned above or a divide-by-zero exception ² .
4	Denormal operand exception ² .
5	Numeric overflow and underflow exceptions possibly in conjunction with the inexact result exception ² .
6 (Lowest)	Inexact result exception.

NOTES:

1. Though a QNaN this is not an exception, the handling of a QNaN operand has precedence over lower priority exceptions. For example, a QNaN divided by zero results in a QNaN, not a divide-by-zero- exception.
2. If masked, then instruction execution continues, and a lower priority exception can occur as well.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the SSE/SSE2/SSE3 instruction that was executed when the SIMD floating-point exception was generated. This is the faulting instruction in which the error condition was detected.

Program State Change

A program-state change does not accompany a SIMD floating-point exception because the handling of the exception is immediate unless the particular exception is masked. The available state information is often sufficient to allow recovery from the error and re-execution of the faulting instruction if needed.

Interrupt 20—Virtualization Exception (#VE)

Exception Class **Fault.**

Description

Indicates that the processor detected an EPT violation in VMX non-root operation. Not all EPT violations cause virtualization exceptions. See Section 27.5.7.2 for details.

The exception handler can recover from EPT violations and restart the program or task without any loss of program continuity. In some cases, however, the problem that caused the EPT violation may be uncorrectable.

Exception Error Code

None.

Saved Instruction Pointer

The saved contents of CS and EIP registers generally point to the instruction that generated the exception.

Program State Change

A program-state change does not normally accompany a virtualization exception, because the instruction that causes the exception to be generated is not executed. After the virtualization exception handler has corrected the violation (for example, by executing the EPTP-switching VM function), execution of the program or task can be resumed.

Additional Exception-Handling Information

The processor saves information about virtualization exceptions in the virtualization-exception information area. See Section 27.5.7.2 for details.

Interrupt 21—Control Protection Exception (#CP)

Exception Class **Fault.**

Description

Indicates a control flow transfer attempt violated the control flow enforcement technology constraints.

Exception Error Code

Yes (special format). The processor provides the control protection exception handler with following information through the error code on the stack.

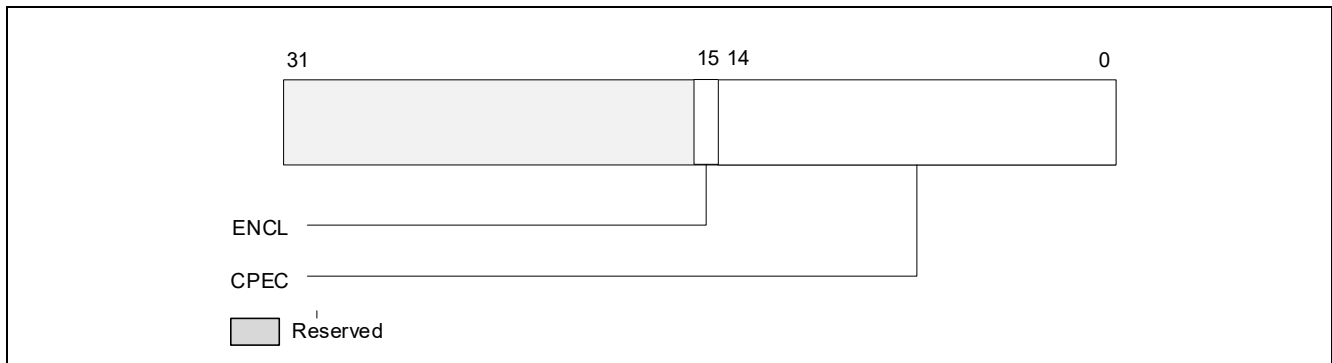


Figure 7-12. Exception Error Code Information

- Bit 14:0 - CPEC
 - 1 - NEAR-RET: Indicates the #CP was caused by a near RET instruction.
 - 2 - FAR-RET/IRET: Indicates the #CP was caused by a FAR RET or IRET instruction.
 - 3 - ENDBRANCH: indicates the #CP was due to missing ENDBRANCH at target of an indirect call or jump instruction.
 - 4 - RSTORSSP: Indicates the #CP was caused by a shadow-stack-restore token check failure in the RSTORSSP instruction.
 - 5- SETSSBSY: Indicates #CP was caused by a supervisor shadow stack token check failure in the SETSSBSY instruction.
- Bit 15 (ENCL) of the error code, if set to 1, indicates the #CP occurred during enclave execution.

Saved Instruction Pointer

The saved contents of the CS and EIP registers generally point to the instruction that generated the exception.

Program State Change

A program-state change does not normally accompany a control protection exception, because the instruction that causes the exception to be generated is not executed.

When a control protection exception is generated during a task switch, the program-state may change as follows. During a task switch, a control protection exception can occur during any of following operations:

- If task switch is initiated by IRET, CS and LIP stored on old task shadow stack do not match CS and LIP of new task (where LIP is the linear address of the return address).
- If task switch is initiated by IRET and SSP of new task loaded from shadow stack of old task (if new task CPL is < 3), OR the SSP from IA32_PL3_SSP (if new task CPL = 3) is not aligned to 4 bytes or is a value beyond 4GB.

In these cases the exception occurs in the context of the new task. The instruction pointer refers to the first instruction of the new task, not to the instruction which caused the task switch (or the last instruction to be executed, in the case of an interrupt). If the design of the operating system permits control protection faults to occur during task-switches, the control protection fault handler should be called through a task gate.

Interrupts 32 to 255—User Defined Interrupts

Exception Class **Not applicable.**

Description

Indicates that the processor did one of the following things:

- Executed an INT *n* instruction where the instruction operand is one of the vector numbers from 32 through 255.
- Responded to an interrupt request at the INTR pin or from the local APIC when the interrupt vector number associated with the request is from 32 through 255.

Exception Error Code

Not applicable.

Saved Instruction Pointer

The saved contents of CS and EIP registers point to the instruction that follows the INT *n* instruction or instruction following the instruction on which the INTR signal occurred.

Program State Change

A program-state change does not accompany interrupts generated by the INT *n* instruction or the INTR signal. The INT *n* instruction generates the interrupt within the instruction stream. When the processor receives an INTR signal, it commits all state changes for all previous instructions before it responds to the interrupt; so, program execution can resume upon returning from the interrupt handler.

13. Updates to Chapter 12, Volume 3A

Change bars and violet text show changes to Chapter 12 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- Updated APIC Base field with MAXAPICADDR in Section 12.4.4, "Local APIC Status and Location".

CHAPTER 12

ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (APIC)

The Advanced Programmable Interrupt Controller (APIC), referred to in the following sections as the local APIC, was introduced into the IA-32 processors with the Pentium processor (see Section 24.27, “Advanced Programmable Interrupt Controller (APIC)”) and is included in the P6 family, Pentium 4, Intel Xeon processors, and other more recent Intel 64 and IA-32 processor families (see Section 12.4.2, “Presence of the Local APIC”). The local APIC performs two primary functions for the processor:

- It receives interrupts from the processor’s interrupt pins, from internal sources and from an external I/O APIC (or other external interrupt controller). It sends these to the processor core for handling.
- In multiple processor (MP) systems, it sends and receives interprocessor interrupt (IPI) messages to and from other logical processors on the system bus. IPI messages can be used to distribute interrupts among the processors in the system or to execute system wide functions (such as, booting up processors or distributing work among a group of processors).

The external **I/O APIC** is part of Intel’s system chipset. Its primary function is to receive external interrupt events from the system and its associated I/O devices and relay them to the local APIC as interrupt messages. In MP systems, the I/O APIC also provides a mechanism for distributing external interrupts to the local APICs of selected processors or groups of processors on the system bus.

This chapter provides a description of the local APIC and its programming interface. It also provides an overview of the interface between the local APIC and the I/O APIC. Contact Intel for detailed information about the I/O APIC.

When a local APIC has sent an interrupt to its processor core for handling, the processor uses the interrupt and exception handling mechanism described in Chapter 7, “Interrupt and Exception Handling.” See Section 7.1, “Interrupt and Exception Overview,” for an introduction to interrupt and exception handling.

12.1 LOCAL AND I/O APIC OVERVIEW

Each local APIC consists of a set of APIC registers (see Table 12-1) and associated hardware that control the delivery of interrupts to the processor core and the generation of IPI messages. The APIC registers are memory mapped and can be read and written to using the MOV instruction.

Local APICs can receive interrupts from the following sources:

- **Locally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected directly to the processor’s local interrupt pins (LINT0 and LINT1). The I/O devices may also be connected to an 8259-type interrupt controller that is in turn connected to the processor through one of the local interrupt pins.
- **Externally connected I/O devices** — These interrupts originate as an edge or level asserted by an I/O device that is connected to the interrupt input pins of an I/O APIC. Interrupts are sent as I/O interrupt messages from the I/O APIC to one or more of the processors in the system.
- **Inter-processor interrupts (IPIs)** — An Intel 64 or IA-32 processor can use the IPI mechanism to interrupt another processor or group of processors on the system bus. IPIs are used for software self-interrupts, interrupt forwarding, or preemptive scheduling.
- **APIC timer generated interrupts** — The local APIC timer can be programmed to send a local interrupt to its associated processor when a programmed count is reached (see Section 12.5.4, “APIC Timer”).
- **Performance monitoring counter interrupts** — P6 family, Pentium 4, and Intel Xeon processors provide the ability to send an interrupt to its associated processor when a performance-monitoring counter overflows (see Section 21.6.3.5.8, “Generating an Interrupt on Overflow”).
- **Thermal Sensor interrupts** — Pentium 4 and Intel Xeon processors provide the ability to send an interrupt to themselves when the internal thermal sensor has been tripped (see Section 16.8.2, “Thermal Monitor”).
- **APIC internal error interrupts** — When an error condition is recognized within the local APIC (such as an attempt to access an unimplemented register), the APIC can be programmed to send an interrupt to its associated processor (see Section 12.5.3, “Error Handling”).

Of these interrupt sources: the processor's LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, the thermal sensor, and the internal APIC error detector are referred to as **local interrupt sources**. Upon receiving a signal from a local interrupt source, the local APIC delivers the interrupt to the processor core using an interrupt delivery protocol that has been set up through a group of APIC registers called the **local vector table** or **LVT** (see Section 12.5.1, "Local Vector Table"). A separate entry is provided in the local vector table for each local interrupt source, which allows a specific interrupt delivery protocol to be set up for each source. For example, if the LINT1 pin is going to be used as an NMI pin, the LINT1 entry in the local vector table can be set up to deliver an interrupt with vector number 2 (NMI interrupt) to the processor core.

The local APIC handles interrupts from the other two interrupt sources (externally connected I/O devices and IPIs) through its IPI message handling facilities.

A processor can generate IPIs by programming the interrupt command register (ICR) in its local APIC (see Section 12.6.1, "Interrupt Command Register (ICR)"). The act of writing to the ICR causes an IPI message to be generated and issued on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for Pentium and P6 family processors). See Section 12.2, "System Bus Vs. APIC Bus."

IPIs can be sent to other processors in the system or to the originating processor (self-interrupts). When the target processor receives an IPI message, its local APIC handles the message automatically (using information included in the message such as vector number and trigger mode). See Section 12.6, "Issuing Interprocessor Interrupts," for a detailed explanation of the local APIC's IPI message delivery and acceptance mechanism.

The local APIC can also receive interrupts from externally connected devices through the I/O APIC (see Figure 12-1). The I/O APIC is responsible for receiving interrupts generated by system hardware and I/O devices and forwarding them to the local APIC as interrupt messages.

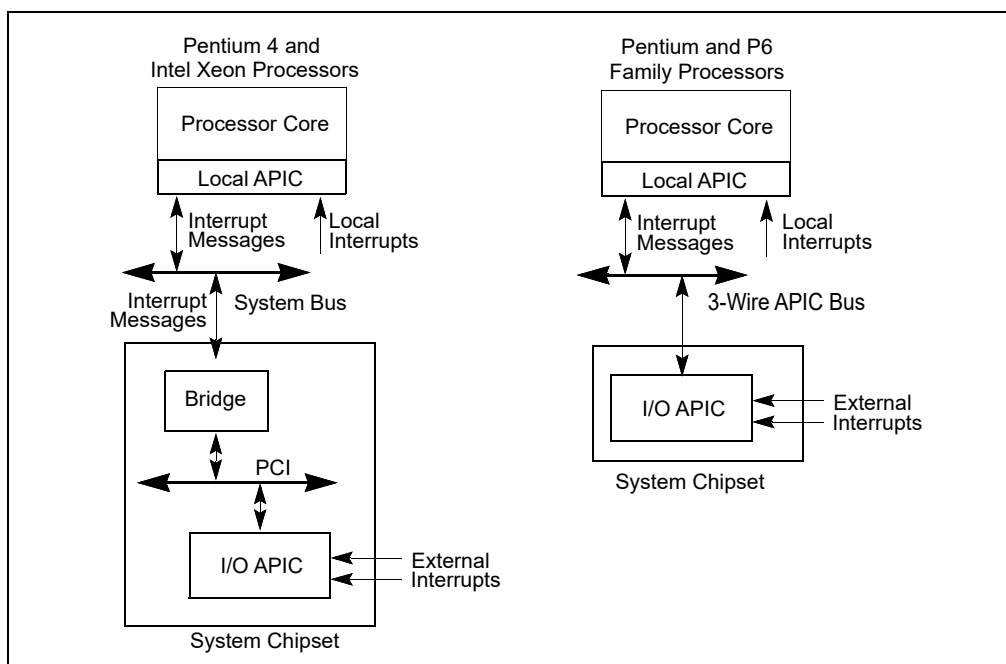


Figure 12-1. Relationship of Local APIC and I/O APIC In Single-Processor Systems

Individual pins on the I/O APIC can be programmed to generate a specific interrupt vector when asserted. The I/O APIC also has a "virtual wire mode" that allows it to communicate with a standard 8259A-style external interrupt controller. Note that the local APIC can be disabled (see Section 12.4.3, "Enabling or Disabling the Local APIC"). This allows an associated processor core to receive interrupts directly from an 8259A interrupt controller.

Both the local APIC and the I/O APIC are designed to operate in MP systems (see Figures 12-2 and 12-3). Each local APIC handles interrupts from the I/O APIC, IPIs from processors on the system bus, and self-generated interrupts. Interrupts can also be delivered to the individual processors through the local interrupt pins; however, this mechanism is commonly not used in MP systems.

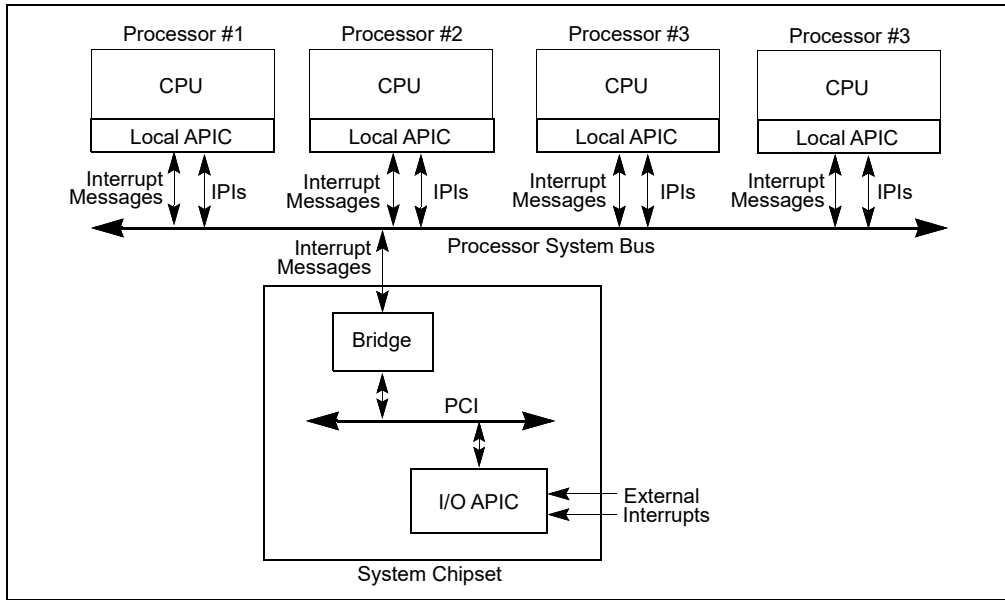


Figure 12-2. Local APICs and I/O APIC When Intel Xeon Processors Are Used in Multiple-Processor Systems

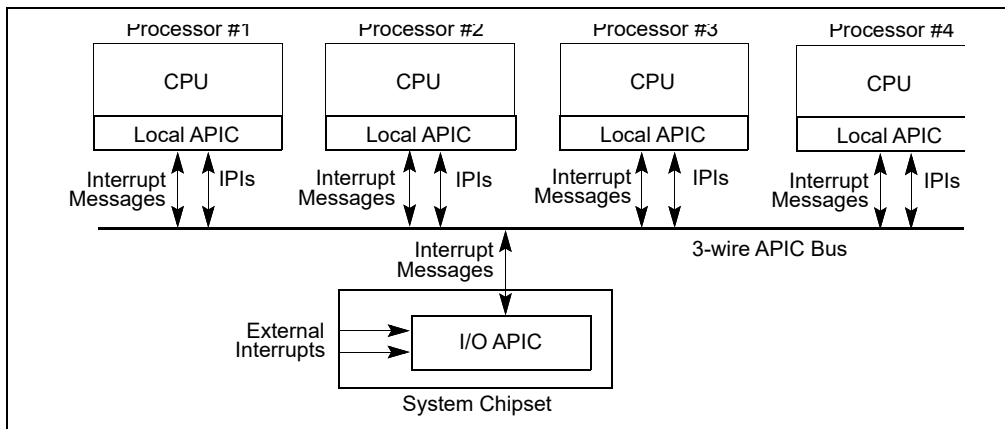


Figure 12-3. Local APICs and I/O APIC When P6 Family Processors Are Used in Multiple-Processor Systems

The IPI mechanism is typically used in MP systems to send fixed interrupts (interrupts for a specific vector number) and special-purpose interrupts to processors on the system bus. For example, a local APIC can use an IPI to forward a fixed interrupt to another processor for servicing. Special-purpose IPIs (including NMI, INIT, SMI, and SIPI IPIs) allow one or more processors on the system bus to perform system-wide boot-up and control functions.

The following sections focus on the local APIC and its implementation in the Pentium 4, Intel Xeon, and P6 family processors. In these sections, the terms “local APIC” and “I/O APIC” refer to local and I/O APICs used with the P6 family processors and to local and I/O xAPICs used with the Pentium 4 and Intel Xeon processors (see Section 12.3, “The Intel® 82489DX External APIC, the APIC, the xAPIC, and the X2APIC”).

12.2 SYSTEM BUS VS. APIC BUS

For the P6 family and Pentium processors, the I/O APIC and local APICs communicate through the 3-wire inter-APIC bus (see Figure 12-3). Local APICs also use the APIC bus to send and receive IPIs. The APIC bus and its messages are invisible to software and are not classed as architectural.

Beginning with the Pentium 4 and Intel Xeon processors, the I/O APIC and local APICs (using the xAPIC architecture) communicate through the system bus (see Figure 12-2). The I/O APIC sends interrupt requests to the processors on the system bus through bridge hardware that is part of the Intel chipset. The bridge hardware generates the interrupt messages that go to the local APICs. IPIs between local APICs are transmitted directly on the system bus.

12.3 THE INTEL® 82489DX EXTERNAL APIC, THE APIC, THE XAPIC, AND THE X2APIC

The local APIC in the P6 family and Pentium processors is an architectural subset of the Intel® 82489DX external APIC. See Section 24.27.1, “Software Visible Differences Between the Local APIC and the 82489DX.”

The APIC architecture used in the Pentium 4 and Intel Xeon processors (called the xAPIC architecture) is an extension of the APIC architecture found in the P6 family processors. The primary difference between the APIC and xAPIC architectures is that with the xAPIC architecture, the local APICs and the I/O APIC communicate through the system bus. With the APIC architecture, they communicate through the APIC bus (see Section 12.2, “System Bus Vs. APIC Bus”). Also, some APIC architectural features have been extended and/or modified in the xAPIC architecture. These extensions and modifications are described in Section 12.4 through Section 12.10.

The basic operating mode of the xAPIC is **xAPIC mode**. The x2APIC architecture is an extension of the xAPIC architecture, primarily to increase processor addressability. The x2APIC architecture provides backward compatibility to the xAPIC architecture and forward extendability for future Intel platform innovations. These extensions and modifications are supported by a new mode of execution (**x2APIC mode**) are detailed in Section 12.12.

12.4 LOCAL APIC

The following sections describe the architecture of the local APIC and how to detect it, identify it, and determine its status. Descriptions of how to program the local APIC are given in Section 12.5.1, “Local Vector Table,” and Section 12.6.1, “Interrupt Command Register (ICR).”

12.4.1 The Local APIC Block Diagram

Figure 12-4 gives a functional block diagram for the local APIC. Software interacts with the local APIC by reading and writing its registers. APIC registers are memory-mapped to a 4-KByte region of the processor’s physical address space with an initial starting address of FEE00000H. For correct APIC operation, this address space must be mapped to an area of memory that has been designated as strong uncacheable (UC). See Section 13.3, “Methods of Caching Available.”

In MP system configurations, the APIC registers for Intel 64 or IA-32 processors on the system bus are initially mapped to the same 4-KByte region of the physical address space. Software has the option of changing initial mapping to a different 4-KByte region for all the local APICs or of mapping the APIC registers for each local APIC to its own 4-KByte region. Section 12.4.5, “Relocating the Local APIC Registers,” describes how to relocate the base address for APIC registers.

On processors supporting x2APIC architecture (indicated by CPUID.01H:ECX[21] = 1), the local APIC supports operation both in xAPIC mode and (if enabled by software) in x2APIC mode. x2APIC mode provides extended processor addressability (see Section 12.12).

NOTE

For P6 family, Pentium 4, and Intel Xeon processors, the APIC handles all memory accesses to addresses within the 4-KByte APIC register space internally and no external bus cycles are produced. For the Pentium processors with an on-chip APIC, bus cycles are produced for accesses to the APIC register space. Thus, for software intended to run on Pentium processors, system software should explicitly not map the APIC register space to regular system memory. Doing so can result in an invalid opcode exception (#UD) being generated or unpredictable execution.

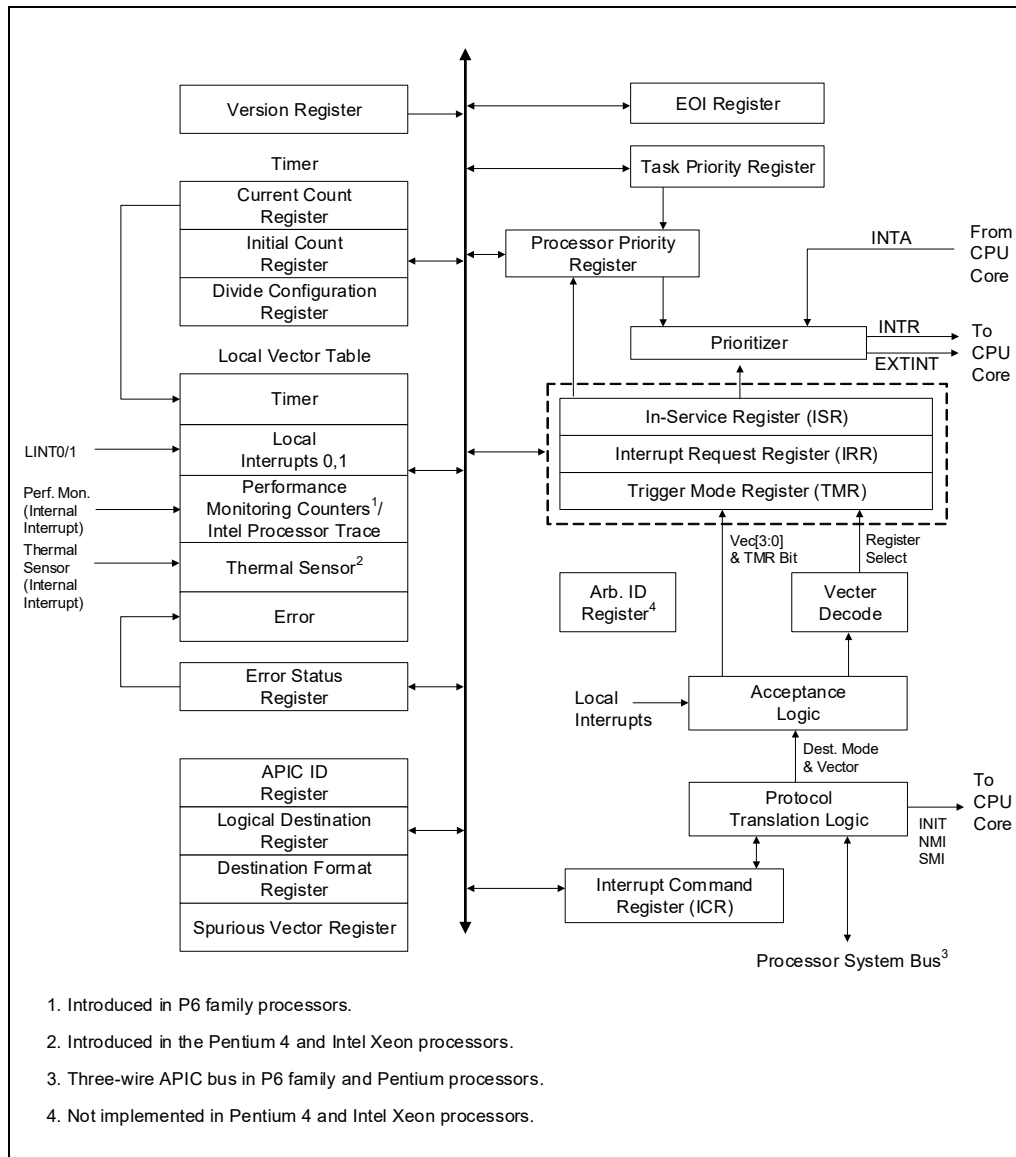


Figure 12-4. Local APIC Structure

Table 12-1 shows how the APIC registers are mapped into the 4-KByte APIC register space. Registers are 32 bits, 64 bits, or 256 bits in width; all are aligned on 128-bit boundaries. All 32-bit registers should be accessed using 128-bit aligned 32-bit loads or stores. Some processors may support loads and stores of less than 32 bits to some of the APIC registers. This is model specific behavior and is not guaranteed to work on all processors. Any FP/MMX/SSE access to an APIC register, or any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results or unexpected exceptions, including machine checks, and may vary between implementations. Wider registers (64-bit or 256-bit) must be accessed using multiple 32-bit loads or stores, with all accesses being 128-bit aligned.

The local APIC registers listed in Table 12-1 are not MSRs. The only MSR associated with the programming of the local APIC is the IA32_APIC_BASE MSR (see Section 12.4.3, "Enabling or Disabling the Local APIC").

NOTE

In processors based on Nehalem¹ microarchitecture, the Local APIC ID Register is no longer Read/Write; it is Read Only.

Table 12-1. Local APIC Register Address Map

Address	Register Name	Software Read/Write
FEE0 0000H	Reserved	
FEE0 0010H	Reserved	
FEE0 0020H	Local APIC ID Register	Read/Write.
FEE0 0030H	Local APIC Version Register	Read Only.
FEE0 0040H	Reserved	
FEE0 0050H	Reserved	
FEE0 0060H	Reserved	
FEE0 0070H	Reserved	
FEE0 0080H	Task Priority Register (TPR)	Read/Write.
FEE0 0090H	Arbitration Priority Register ¹ (APR)	Read Only.
FEE0 00A0H	Processor Priority Register (PPR)	Read Only.
FEE0 00B0H	EOI Register	Write Only.
FEE0 00C0H	Remote Read Register ¹ (RRD)	Read Only
FEE0 00D0H	Logical Destination Register	Read/Write.
FEE0 00E0H	Destination Format Register	Read/Write (see Section 12.6.2.2).
FEE0 00F0H	Spurious Interrupt Vector Register	Read/Write (see Section 12.9.
FEE0 0100H	In-Service Register (ISR); bits 31:0	Read Only.
FEE0 0110H	In-Service Register (ISR); bits 63:32	Read Only.
FEE0 0120H	In-Service Register (ISR); bits 95:64	Read Only.
FEE0 0130H	In-Service Register (ISR); bits 127:96	Read Only.
FEE0 0140H	In-Service Register (ISR); bits 159:128	Read Only.
FEE0 0150H	In-Service Register (ISR); bits 191:160	Read Only.
FEE0 0160H	In-Service Register (ISR); bits 223:192	Read Only.
FEE0 0170H	In-Service Register (ISR); bits 255:224	Read Only.
FEE0 0180H	Trigger Mode Register (TMR); bits 31:0	Read Only.
FEE0 0190H	Trigger Mode Register (TMR); bits 63:32	Read Only.
FEE0 01A0H	Trigger Mode Register (TMR); bits 95:64	Read Only.
FEE0 01B0H	Trigger Mode Register (TMR); bits 127:96	Read Only.
FEE0 01C0H	Trigger Mode Register (TMR); bits 159:128	Read Only.
FEE0 01D0H	Trigger Mode Register (TMR); bits 191:160	Read Only.
FEE0 01E0H	Trigger Mode Register (TMR); bits 223:192	Read Only.
FEE0 01F0H	Trigger Mode Register (TMR); bits 255:224	Read Only.
FEE0 0200H	Interrupt Request Register (IRR); bits 31:0	Read Only.
FEE0 0210H	Interrupt Request Register (IRR); bits 63:32	Read Only.
FEE0 0220H	Interrupt Request Register (IRR); bits 95:64	Read Only.

1. See Table 2-1, "CPUID Signature Values of DisplayFamily_DisplayModel," on page 1, and Section 2.8, "MSRs In Processors Based on Nehalem Microarchitecture," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, to determine which processors are based on Nehalem microarchitecture.

Table 12-1. Local APIC Register Address Map (Contd.)

Address	Register Name	Software Read/Write
FEE0 0230H	Interrupt Request Register (IRR); bits 127:96	Read Only.
FEE0 0240H	Interrupt Request Register (IRR); bits 159:128	Read Only.
FEE0 0250H	Interrupt Request Register (IRR); bits 191:160	Read Only.
FEE0 0260H	Interrupt Request Register (IRR); bits 223:192	Read Only.
FEE0 0270H	Interrupt Request Register (IRR); bits 255:224	Read Only.
FEE0 0280H	Error Status Register	Write/Read; see Section 12.5.3.
FEE0 0290H through FEE0 02E0H	Reserved	
FEE0 02F0H	LVT Corrected Machine Check Interrupt (CMCI) Register	Read/Write.
FEE0 0300H	Interrupt Command Register (ICR); bits 0-31	Read/Write.
FEE0 0310H	Interrupt Command Register (ICR); bits 32-63	Read/Write.
FEE0 0320H	LVT Timer Register	Read/Write.
FEE0 0330H	LVT Thermal Sensor Register ²	Read/Write.
FEE0 0340H	LVT Performance Monitoring Counters Register ³	Read/Write.
FEE0 0350H	LVT LINT0 Register	Read/Write.
FEE0 0360H	LVT LINT1 Register	Read/Write.
FEE0 0370H	LVT Error Register	Read/Write.
FEE0 0380H	Initial Count Register (for Timer)	Read/Write.
FEE0 0390H	Current Count Register (for Timer)	Read Only.
FEE0 03A0H through FEE0 03D0H	Reserved	
FEE0 03E0H	Divide Configuration Register (for Timer)	Read/Write.
FEE0 03F0H	Reserved	

NOTES:

1. Not supported in the Pentium 4 and Intel Xeon processors. The Illegal Register Access bit (7) of the ESR will not be set when writing to these registers.
2. Introduced in the Pentium 4 and Intel Xeon processors. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.
3. Introduced in the Pentium Pro processor. This APIC register and its associated function are implementation dependent and may not be present in future IA-32 or Intel 64 processors.

12.4.2 Presence of the Local APIC

Beginning with the P6 family processors, the presence or absence of an on-chip local APIC can be detected using the CPUID instruction. When the CPUID instruction is executed with a source operand of 1 in the EAX register, bit 9 of the CPUID feature flags returned in the EDX register indicates the presence (set) or absence (clear) of a local APIC.

12.4.3 Enabling or Disabling the Local APIC

The local APIC can be enabled or disabled in either of two ways:

1. Using the APIC global enable/disable flag in the IA32_APIC_BASE MSR (MSR address 1BH; see Figure 12-5):

- When IA32_APIC_BASE[11] is 0, the processor is functionally equivalent to an IA-32 processor without an on-chip APIC. The CPUID feature flag for the APIC (see Section 12.4.2, “Presence of the Local APIC”) is also set to 0.
 - When IA32_APIC_BASE[11] is set to 0, processor APICs based on the 3-wire APIC bus cannot be generally re-enabled until a system hardware reset. The 3-wire bus loses track of arbitration that would be necessary for complete re-enabling. Certain APIC functionality can be enabled (for example: performance and thermal monitoring interrupt generation).
 - For processors that use Front Side Bus (FSB) delivery of interrupts, software may disable or enable the APIC by setting and resetting IA32_APIC_BASE[11]. A hardware reset is not required to re-start APIC functionality, if software guarantees no interrupt will be sent to the APIC as IA32_APIC_BASE[11] is cleared.
 - When IA32_APIC_BASE[11] is set to 0, prior initialization to the APIC may be lost and the APIC may return to the state described in Section 12.4.7.1, “Local APIC State After Power-Up or Reset.”
2. Using the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 12-23):
- If IA32_APIC_BASE[11] is 1, software can temporarily disable a local APIC at any time by clearing the APIC software enable/disable flag in the spurious-interrupt vector register (see Figure 12-23). The state of the local APIC when in this software-disabled state is described in Section 12.4.7.2, “Local APIC State After It Has Been Software Disabled.”
 - When the local APIC is in the software-disabled state, it can be re-enabled at any time by setting the APIC software enable/disable flag to 1.

For the Pentium processor, the APICEN pin (which is shared with the PICD1 pin) is used during power-up or reset to disable the local APIC.

Note that each entry in the LVT has a mask bit that can be used to inhibit interrupts from being delivered to the processor from selected local interrupt sources (the LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, Intel® Processor Trace, the thermal sensor, and/or the internal APIC error detector).

12.4.4 Local APIC Status and Location

The status and location of the local APIC are contained in the IA32_APIC_BASE MSR (see Figure 12-5). MSR bit functions are described below:

- **BSP flag, bit 8** — Indicates if the processor is the bootstrap processor (BSP). See Section 10.4, “Multiple-Processor (MP) Initialization.” Following a power-up or reset, this flag is set to 1 for the processor selected as the BSP and set to 0 for the remaining processors (APs).
- **APIC Global Enable flag, bit 11** — Enables or disables the local APIC (see Section 12.4.3, “Enabling or Disabling the Local APIC”). This flag is available in the Pentium 4, Intel Xeon, and P6 family processors. It is not guaranteed to be available or available at the same location in future Intel 64 or IA-32 processors.
- **APIC Base field, bits MAXAPICADDR-1:12** — Specifies the base address of the APIC registers.¹ This value is extended by 12 bits at the low end to form the base address. This automatically aligns the address on a 4-KByte boundary. Following a power-up or reset, the field is set to FEE0 0000H.
- Bits 7:0, bits 10:9, and bits 63:MAXAPICADDR in the IA32_APIC_BASE MSR are reserved.

1. MAXAPICADDR is normally CPUID.80000008H:EAX[7:0] for processors that support CPUID.80000008H and 36 otherwise. If IA32_TME_ACTIVATE[39:36] > 0 (indicating that SEAM-private KeyIDs have been configured), MAXAPICADDR is reduced by the value of IA32_TME_ACTIVATE[35:32] (the KeyID bits in the address must be 0).

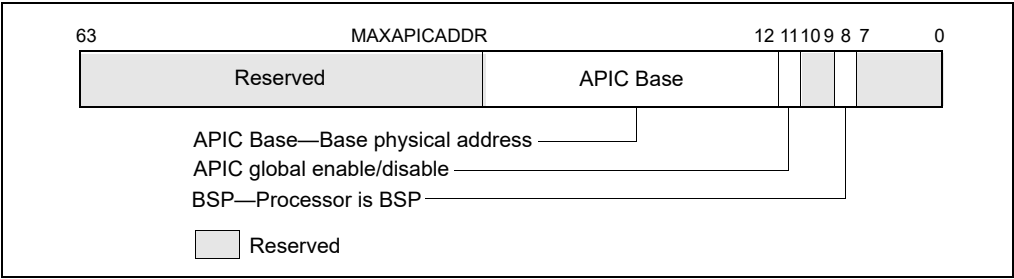


Figure 12-5. IA32_APIC_BASE MSR (APIC_BASE_MSR in P6 Family)

12.4.5 Relocating the Local APIC Registers

The Pentium 4, Intel Xeon, and P6 family processors permit the starting address of the APIC registers to be relocated from FEE00000H to another physical address by modifying the value in the base address field of the IA32_APIC_BASE MSR. This extension of the APIC architecture is provided to help resolve conflicts with memory maps of existing systems and to allow individual processors in an MP system to map their APIC registers to different locations in physical memory.

12.4.6 Local APIC ID

At power up, system hardware assigns a unique APIC ID to each local APIC on the system bus (for Pentium 4 and Intel Xeon processors) or on the APIC bus (for P6 family and Pentium processors). The hardware assigned APIC ID is based on system topology and includes encoding for socket position and cluster information (see Figure 10-2 and Section 10.9.1, “Hierarchical Mapping of Shared Resources”).

In MP systems, the local APIC ID is also used as a processor ID by the BIOS and the operating system. Some processors permit software to modify the APIC ID. However, the ability of software to modify the APIC ID is processor model specific. Because of this, operating system software should avoid writing to the local APIC ID register. The value returned by bits 31-24 of the EBX register (when the CUID instruction is executed with a source operand value of 1 in the EAX register) is always the Initial APIC ID (determined by the platform initialization). This is true even if software has changed the value in the Local APIC ID register.

The processor receives the hardware assigned APIC ID (or Initial APIC ID) by sampling pins A11# and A12# and pins BR0# through BR3# (for the Pentium 4, Intel Xeon, and P6 family processors) and pins BE0# through BE3# (for the Pentium processor). The APIC ID latched from these pins is stored in the APIC ID field of the local APIC ID register (see Figure 12-6), and is used as the Initial APIC ID for the processor.

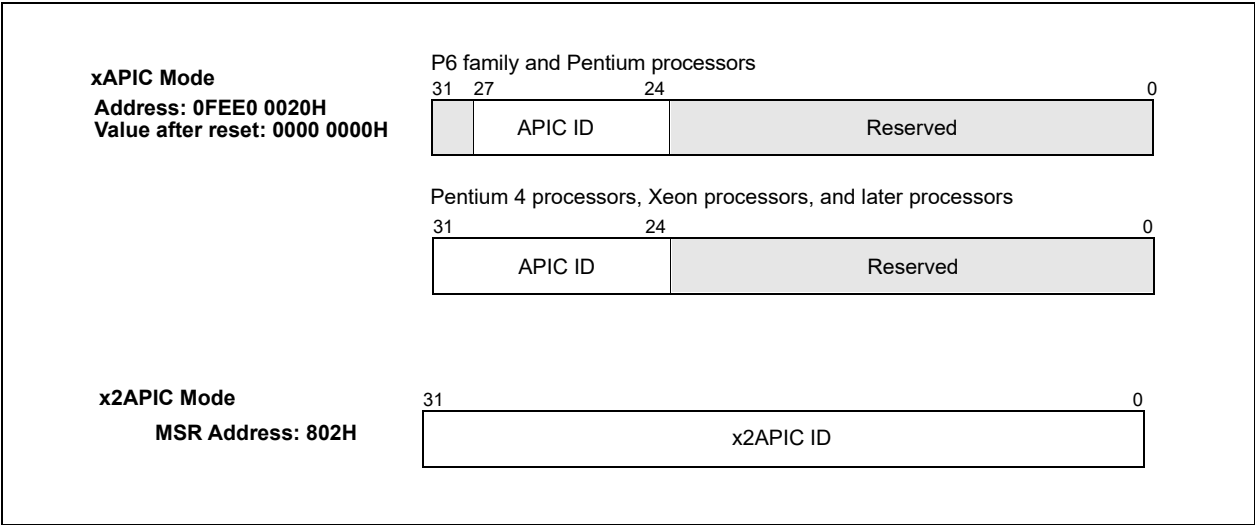


Figure 12-6. Local APIC ID Register

For the P6 family and Pentium processors, the local APIC ID field in the local APIC ID register is 4 bits. Encodings 0H through EH can be used to uniquely identify 15 different processors connected to the APIC bus. For the Pentium 4 and Intel Xeon processors, the xAPIC specification extends the local APIC ID field to 8 bits. These can be used to identify up to 255 processors in the system.

12.4.7 Local APIC State

The following sections describe the state of the local APIC and its registers following a power-up or reset, after the local APIC has been software disabled, following an INIT reset, and following an INIT-deassert message.

x2APIC will introduce 32-bit ID; see Section 12.12.

12.4.7.1 Local APIC State After Power-Up or Reset

Following a power-up or reset of the processor, the state of local APIC and its registers are as follows:

- The following registers are reset to all 0s.
 - IRR, ISR, TMR, ICR, LDR, and TPR.
 - Timer initial count and timer current count registers.
 - Divide configuration register.
- The DFR register is reset to all 1s.
- The LVT register is reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The local APIC ID register is set to a unique APIC ID. (Pentium and P6 family processors only). The Arb ID register is set to the value in the APIC ID register.
- The spurious-interrupt vector register is initialized to 000000FFH. By setting bit 8 to 0, software disables the local APIC.
- If the processor is the only processor in the system or it is the BSP in an MP system (see Section 10.4.1, “BSP and AP Processors”); the local APIC will respond normally to INIT and NMI messages, to INIT# signals and to STPCLK# signals. If the processor is in an MP system and has been designated as an AP; the local APIC will respond the same as for the BSP. In addition, it will respond to SIPI messages. For P6 family processors only, an AP will not respond to a STPCLK# signal.

12.4.7.2 Local APIC State After It Has Been Software Disabled

When the APIC software enable/disable flag in the spurious interrupt vector register has been explicitly cleared (as opposed to being cleared during a power up or reset), the local APIC is temporarily disabled (see Section 12.4.3, “Enabling or Disabling the Local APIC”). The operation and response of a local APIC while in this software-disabled state is as follows:

- The local APIC will respond normally to INIT, NMI, SMI, and SIPI messages.
- Pending interrupts in the IRR and ISR registers are held and require masking or handling by the CPU.
- The local APIC can still issue IPIs. It is software’s responsibility to avoid issuing IPIs through the IPI mechanism and the ICR register if sending interrupts through this mechanism is not desired.
- The reception of any interrupt or transmission of any IPIs that are in progress when the local APIC is disabled are completed before the local APIC enters the software-disabled state.
- The mask bits for all the LVT entries are set. Attempts to reset these bits will be ignored.
- (For Pentium and P6 family processors) The local APIC continues to listen to all bus messages in order to keep its arbitration ID synchronized with the rest of the system.

12.4.7.3 Local APIC State After an INIT Reset (“Wait-for-SIPI” State)

An INIT reset of the processor can be initiated in either of two ways:

- By asserting the processor's INIT# pin.
- By sending the processor an INIT IPI (an IPI with the delivery mode set to INIT).

Upon receiving an INIT through either of these mechanisms, the processor responds by beginning the initialization process of the processor core and the local APIC. The state of the local APIC following an INIT reset is the same as it is after a power-up or hardware reset, except that the APIC ID and arbitration ID registers are not affected. This state is also referred to as the "wait-for-SIPI" state (see also: Section 10.4.2, "MP Initialization Protocol Requirements and Restrictions").

12.4.7.4 Local APIC State After It Receives an INIT-Deassert IPI

Only the Pentium and P6 family processors support the INIT-deassert IPI. An INIT-deassert IPI has no effect on the state of the APIC, other than to reload the arbitration ID register with the value in the APIC ID register.

12.4.8 Local APIC Version Register

The local APIC contains a hardwired version register. Software can use this register to identify the APIC version (see Figure 12-7). In addition, the register specifies the number of entries in the local vector table (LVT) for a specific implementation.

The fields in the local APIC version register are as follows:

Version	The version numbers of the local APIC:
0XH	82489DX discrete APIC.
10H - 15H	Integrated APIC.
Other values	reserved.

Max LVT Entry Shows the number of LVT entries minus 1. For the Pentium 4 and Intel Xeon processors (which have 6 LVT entries), the value returned in the Max LVT field is 5; for the P6 family processors (which have 5 LVT entries), the value returned is 4; for the Pentium processor (which has 4 LVT entries), the value returned is 3. For processors based on the Nehalem microarchitecture (which has 7 LVT entries) and onward, the value returned is 6.

Suppress EOI-broadcasts

Indicates whether software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register; see Section 12.8.5 and Section 12.9.

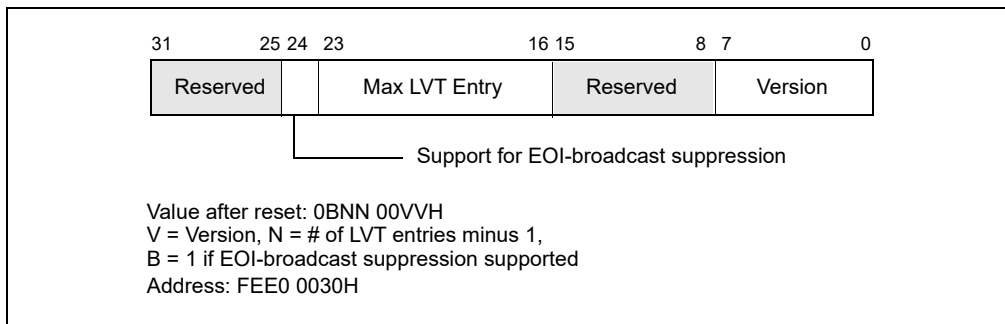


Figure 12-7. Local APIC Version Register

12.5 HANDLING LOCAL INTERRUPTS

The following sections describe facilities that are provided in the local APIC for handling local interrupts. These include: the processor's LINT0 and LINT1 pins, the APIC timer, the performance-monitoring counters, Intel Processor Trace, the thermal sensor, and the internal APIC error detector. Local interrupt handling facilities include: the LVT, the error status register (ESR), the divide configuration register (DCR), and the initial count and current count registers.

12.5.1 Local Vector Table

The local vector table (LVT) allows software to specify the manner in which the local interrupts are delivered to the processor core. It consists of the following 32-bit APIC registers (see Figure 12-8), one for each local interrupt:

- **LVT CMCI Register (FEE0 02F0H)** — Specifies interrupt delivery when an overflow condition of corrected machine check error count reaching a threshold value occurred in a machine check bank supporting CMCI (see Section 17.5.1, “CMCI Local APIC Interface”).
- **LVT Timer Register (FEE0 0320H)** — Specifies interrupt delivery when the APIC timer signals an interrupt (see Section 12.5.4, “APIC Timer”).
- **LVT Thermal Monitor Register (FEE0 0330H)** — Specifies interrupt delivery when the thermal sensor generates an interrupt (see Section 16.8.2, “Thermal Monitor”). This LVT entry is implementation specific, not architectural. If implemented, it will always be at base address FEE0 0330H.
- **LVT Performance Counter Register (FEE0 0340H)** — Specifies interrupt delivery when a performance counter generates an interrupt on overflow (see Section 21.6.3.5.8, “Generating an Interrupt on Overflow”) or when Intel PT signals a ToPA PMI (see Section 35.2.7.2). This LVT entry is implementation specific, not architectural. If implemented, it is not guaranteed to be at base address FEE0 0340H.
- **LVT LINT0 Register (FEE0 0350H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT0 pin.
- **LVT LINT1 Register (FEE0 0360H)** — Specifies interrupt delivery when an interrupt is signaled at the LINT1 pin.
- **LVT Error Register (FEE0 0370H)** — Specifies interrupt delivery when the APIC detects an internal error (see Section 12.5.3, “Error Handling”).

The LVT performance counter register and its associated interrupt were introduced in the P6 processors and are also present in the Pentium 4 and Intel Xeon processors. The LVT thermal monitor register and its associated interrupt were introduced in the Pentium 4 and Intel Xeon processors. The LVT CMCI register and its associated interrupt were introduced in the Intel Xeon 5500 processors.

As shown in Figure 12-8, some of these fields and flags are not available (and reserved) for some entries.

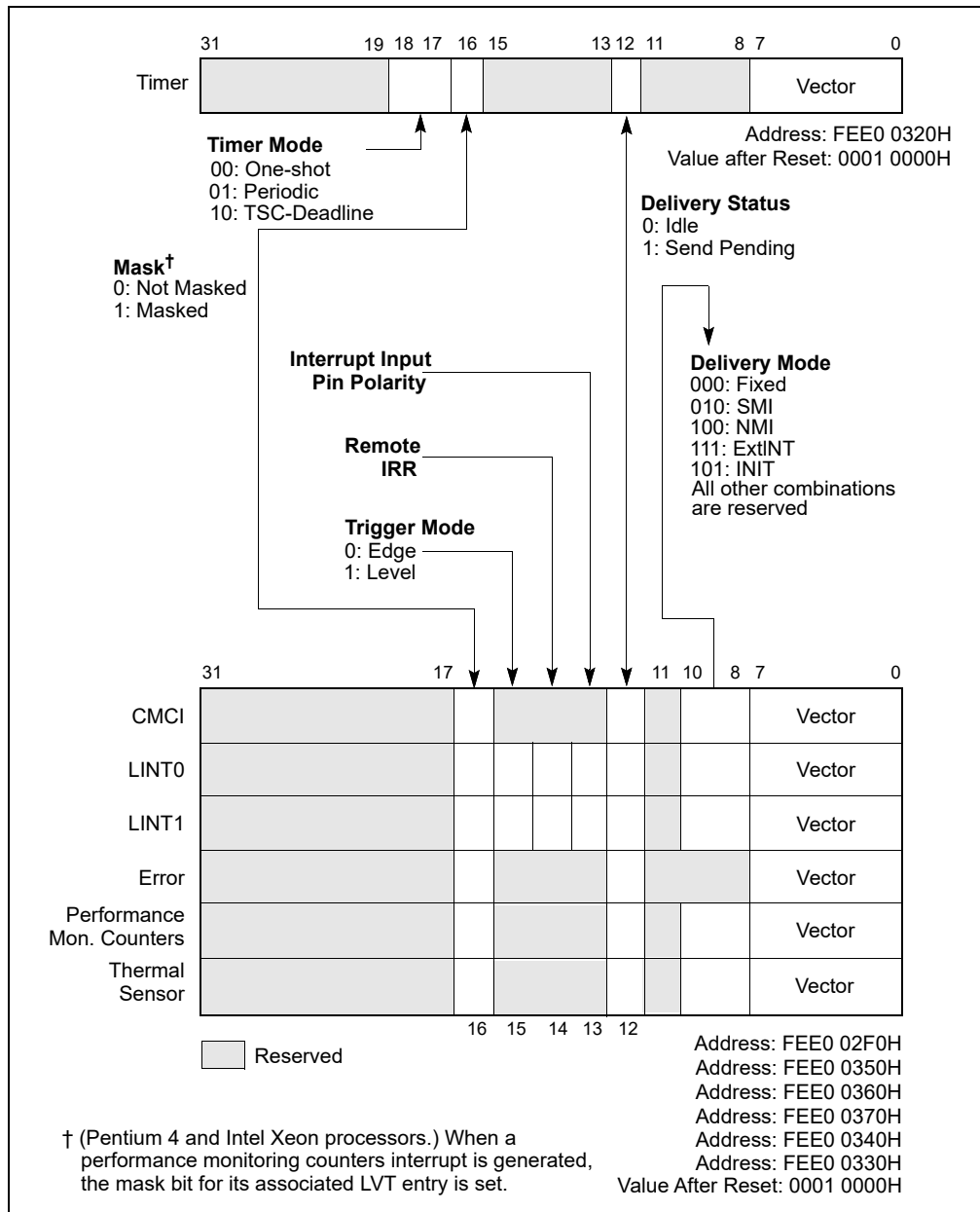


Figure 12-8. Local Vector Table (LVT)

The setup information that can be specified in the registers of the LVT table is as follows:

Vector

Interrupt vector number.

Delivery Mode

Specifies the type of interrupt to be sent to the processor. Some delivery modes will only operate as intended when used in conjunction with a specific trigger mode. The allowable delivery modes are as follows:

000 (Fixed)

Delivers the interrupt specified in the vector field.

010 (SMI)

Delivers an SMI interrupt to the processor core through the processor's local SMI signal path. When using this delivery mode, the vector field should be set to 00H for future compatibility.

100 (NMI)	Delivers an NMI interrupt to the processor. The vector information is ignored.
101 (INIT)	Delivers an INIT request to the processor core, which causes the processor to perform an INIT. When using this delivery mode, the vector field should be set to 00H for future compatibility. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.
110	Reserved; not supported for any LVT register.
111 (ExtINT)	Causes the processor to respond to the interrupt as if the interrupt originated in an externally connected (8259A-compatible) interrupt controller. A special INTA bus cycle corresponding to ExtINT, is routed to the external controller. The external controller is expected to supply the vector information. The APIC architecture supports only one ExtINT source in a system, usually contained in the compatibility bridge. Only one processor in the system should have an LVT entry configured to use the ExtINT delivery mode. Not supported for the LVT CMCI register, the LVT thermal monitor register, or the LVT performance counter register.

Delivery Status (Read Only)

Indicates the interrupt delivery status, as follows:

0 (Idle)	There is currently no activity for this interrupt source, or the previous interrupt from this source was delivered to the processor core and accepted.
1 (Send Pending)	Indicates that an interrupt from this source has been delivered to the processor core but has not yet been accepted (see Section 12.5.5, "Local Interrupt Acceptance").

Interrupt Input Pin Polarity

Specifies the polarity of the corresponding interrupt pin: (0) active high or (1) active low.

Remote IRR Flag (Read Only)

For fixed mode, level-triggered interrupts; this flag is set when the local APIC accepts the interrupt for servicing and is reset when an EOI command is received from the processor. The meaning of this flag is undefined for edge-triggered interrupts and other delivery modes.

Trigger Mode

Selects the trigger mode for the local LINT0 and LINT1 pins: (0) edge sensitive and (1) level sensitive. This flag is only used when the delivery mode is Fixed. When the delivery mode is NMI, SMI, or INIT, the trigger mode is always edge sensitive. When the delivery mode is ExtINT, the trigger mode is always level sensitive. The timer and error interrupts are always treated as edge sensitive.

If the local APIC is not used in conjunction with an I/O APIC and fixed delivery mode is selected; the Pentium 4, Intel Xeon, and P6 family processors will always use level-sensitive triggering, regardless if edge-sensitive triggering is selected.

Software should always set the trigger mode in the LVT LINT1 register to 0 (edge sensitive). Level-sensitive interrupts are not supported for LINT1.

Mask

Interrupt mask: (0) enables reception of the interrupt and (1) inhibits reception of the interrupt. When the local APIC handles a performance-monitoring counters interrupt, it automatically sets the mask flag in the LVT performance counter register. This flag is set to 1 on reset. It can be cleared only by software.

Timer Mode

Bits 18:17 selects the timer mode (see Section 12.5.4):

- (00b) one-shot mode using a count-down value,
- (01b) periodic mode reloading a count-down value,
- (10b) TSC-Deadline mode using absolute target value in IA32_TSC_DEADLINE MSR (see Section 12.5.4.1),
- (11b) is reserved.

12.5.2 Valid Interrupt Vectors

The Intel 64 and IA-32 architectures define 256 vector numbers, ranging from 0 through 255 (see Section 7.2, “Exception and Interrupt Vectors”). Local and I/O APICs support 240 of these vectors (in the range of 16 to 255) as valid interrupts.

When an interrupt vector in the range of 0 to 15 is sent or received through the local APIC, the APIC indicates an illegal vector in its Error Status Register (see Section 12.5.3, “Error Handling”). The Intel 64 and IA-32 architectures reserve vectors 16 through 31 for predefined interrupts, exceptions, and Intel-reserved encodings (see Table 7-1). However, the local APIC does not treat vectors in this range as illegal.

When an illegal vector value (0 to 15) is written to an LVT entry and the delivery mode is Fixed (bits 8-11 equal 0), the APIC may signal an illegal vector error, without regard to whether the mask bit is set or whether an interrupt is actually seen on the input.

12.5.3 Error Handling

The local APIC records errors detected during interrupt handling in the error status register (ESR). The format of the ESR is given in Figure 12-9; it contains the following flags:

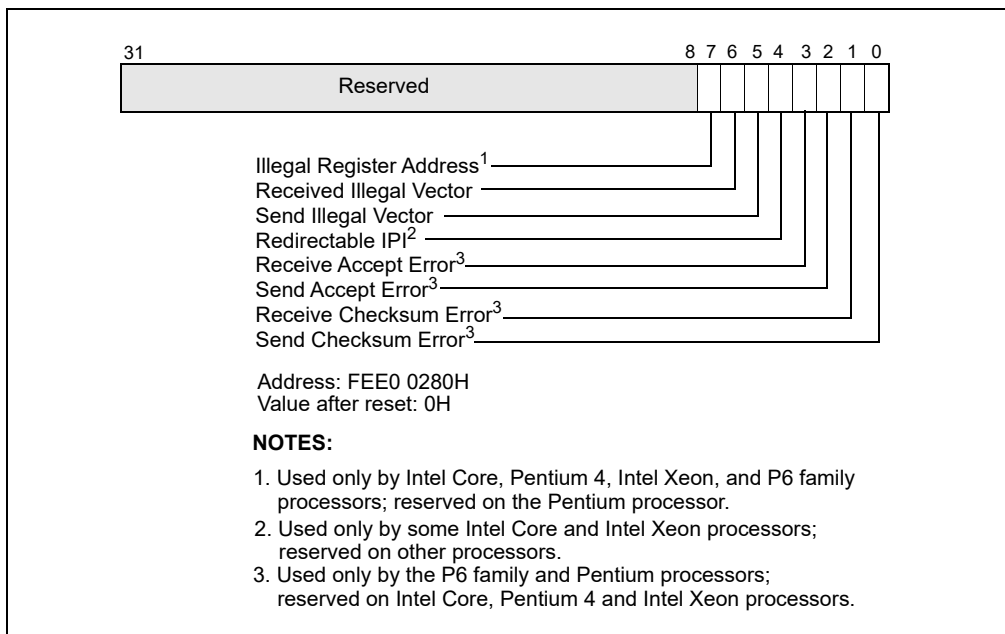


Figure 12-9. Error Status Register (ESR)

- **Bit 0: Send Checksum Error.**
Set when the local APIC detects a checksum error for a message that it sent on the APIC bus. Used only on P6 family and Pentium processors.
- **Bit 1: Receive Checksum Error.**
Set when the local APIC detects a checksum error for a message that it received on the APIC bus. Used only on P6 family and Pentium processors.
- **Bit 2: Send Accept Error.**
Set when the local APIC detects that a message it sent was not accepted by any APIC on the APIC bus. Used only on P6 family and Pentium processors.
- **Bit 3: Receive Accept Error.**
Set when the local APIC detects that the message it received was not accepted by any APIC on the APIC bus, including itself. Used only on P6 family and Pentium processors.

- **Bit 4: Redirectable IPI.**
Set when the local APIC detects an attempt to send an IPI with the lowest-priority delivery mode and the local APIC does not support the sending of such IPIs. This bit is used on some Intel Core and Intel Xeon processors. As noted in Section 12.6.2, the ability of a processor to send a lowest-priority IPI is model-specific and should be avoided.
- **Bit 5: Send Illegal Vector.**
Set when the local APIC detects an illegal vector (one in the range 0 to 15) in the message that it is sending. This occurs as the result of a write to the ICR (in both xAPIC and x2APIC modes) or to SELF IPI register (x2APIC mode only) with an illegal vector.

If the local APIC does not support the sending of lowest-priority IPIs and software writes the ICR to send a lowest-priority IPI with an illegal vector, the local APIC sets only the “redirectable IPI” error bit. The interrupt is not processed and hence the “Send Illegal Vector” bit is not set in the ESR.
- **Bit 6: Receive Illegal Vector.**
Set when the local APIC detects an illegal vector (one in the range 0 to 15) in an interrupt message it receives or in an interrupt generated locally from the local vector table or via a self IPI. Such interrupts are not delivered to the processor; the local APIC will never set an IRR bit in the range 0 to 15.
- **Bit 7: Illegal Register Address**
Set when the local APIC is in xAPIC mode and software attempts to access a register that is reserved in the processor's local-APIC register-address space; see Table 10-1. (The local-APIC register-address space comprises the 4 KBytes at the physical address specified in the IA32_APIC_BASE MSR.) Used only on Intel Core, Intel Atom, Pentium 4, Intel Xeon, and P6 family processors.

In x2APIC mode, software accesses the APIC registers using the RDMSR and WRMSR instructions. Use of one of these instructions to access a reserved register cause a general-protection exception (see Section 10.12.1.3). They do not set the “Illegal Register Access” bit in the ESR.

The ESR is a write/read register. Before attempt to read from the ESR, software should first write to it. (The value written does not affect the values read subsequently; only zero may be written in x2APIC mode.) This write clears any previously logged errors and updates the ESR with any errors detected since the last write to the ESR. This write also rearms the APIC error interrupt triggering mechanism.

The LVT Error Register (see Section 12.5.1) allows specification of the vector of the interrupt to be delivered to the processor core when APIC error is detected. The register also provides a means of masking an APIC-error interrupt. This masking only prevents delivery of APIC-error interrupts; the APIC continues to record errors in the ESR.

12.5.4 APIC Timer

The local APIC unit contains a 32-bit programmable timer that is available to software to time events or operations. This timer is set up by programming four registers: the divide configuration register (see Figure 12-10), the initial-count and current-count registers (see Figure 12-11), and the LVT timer register (see Figure 12-8).

If CPUID.06H:EAX.ALWAYS_RUNNING_APIC_TIMER[2] = 1, the processor's APIC timer runs at a constant rate regardless of P-state transitions and it continues to run at the same rate in deep C-states.

If CPUID.06H:EAX.ALWAYS_RUNNING_APIC_TIMER[2] = 0 or if CPUID 06H is not supported, the APIC timer may temporarily stop while the processor is in deep C-states or during transitions caused by Enhanced Intel Speed-Step® Technology.

The APIC timer frequency will be the processor's bus clock or core crystal clock frequency (when TSC/core crystal clock ratio is enumerated in CPUID.15H) divided by the value specified in the divide configuration register.

The timer can be configured through the timer LVT entry for one-shot or periodic operation. In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, a timer interrupt is generated and the timer remains at its 0 value until reprogrammed.

In periodic mode, the timer is started by writing to the initial-count register (as in one-shot mode), and the value written is copied into the current-count register, which counts down. The current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-

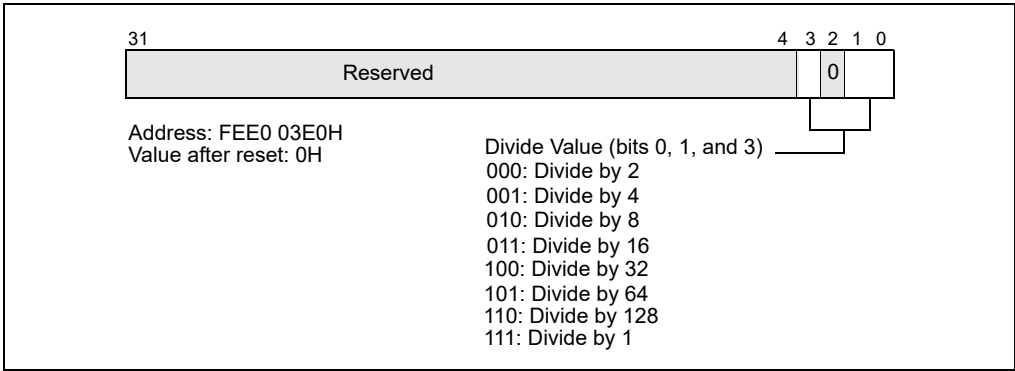


Figure 12-10. Divide Configuration Register

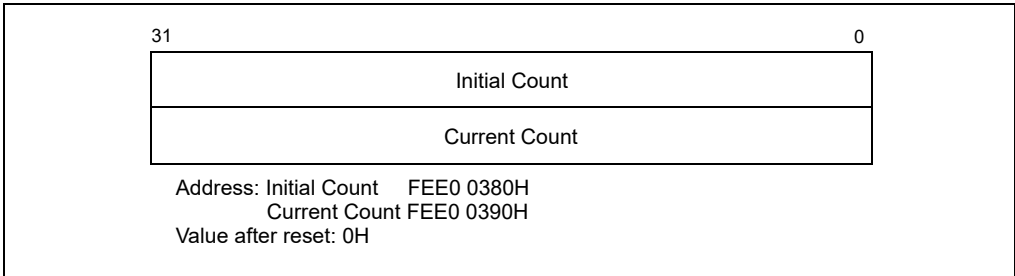


Figure 12-11. Initial Count and Current Count Registers

down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value. The initial-count register is a read-write register; the current-count register is read only. A write of 0 to the initial-count register effectively stops the local APIC timer, in both one-shot and periodic mode. The LVT timer register determines the vector number that is delivered to the processor with the timer interrupt that is generated when the timer count reaches zero. The mask flag in the LVT timer register can be used to mask the timer interrupt.

NOTE

Changing the mode of the APIC timer (from one-shot to periodic or vice versa) by writing to the timer LVT entry does not start the timer. To start the timer, it is necessary to write to the initial-count register as described above.

12.5.4.1 TSC-Deadline Mode

The mode of operation of the local-APIC timer is determined by the LVT Timer Register. Specifically:

- If CPUID.01H:ECX.TSC_DEADLINE[24] = 0, the mode is determined by bit 17 of the register.
- If CPUID.01H:ECX.TSC_DEADLINE[24] = 1, the mode is determined by bits 18:17. See Figure 12-8. (If CPUID.01H:ECX.TSC_DEADLINE[24] = 0, bit 18 of the register is reserved.)

The supported timer modes are given in Table 12-2. The three modes of the local APIC timer are mutually exclusive.

Table 12-2. Local APIC Timer Modes

LVT Bits [18:17]	Timer Mode
00b	One-shot mode, program count-down value in an initial-count register. See Section 12.5.4
01b	Periodic mode, program interval value in an initial-count register. See Section 12.5.4
10b	TSC-Deadline mode, program target value in IA32_TSC_DEADLINE MSR.
11b	Reserved

TSC-deadline mode allows software to use the local APIC timer to signal an interrupt at an absolute time. In TSC-deadline mode, writes to the initial-count register are ignored; and current-count register always reads 0. Instead, timer behavior is controlled using the IA32_TSC_DEADLINE MSR.

The IA32_TSC_DEADLINE MSR (MSR address 6E0H) is a per-logical processor MSR that specifies the time at which a timer interrupt should occur. Writing a non-zero 64-bit value into IA32_TSC_DEADLINE arms the timer. An interrupt is generated when the logical processor's time-stamp counter equals or exceeds the target value in the IA32_TSC_DEADLINE MSR.¹ When the timer generates an interrupt, it disarms itself and clears the IA32_TSC_DEADLINE MSR. Thus, each write to the IA32_TSC_DEADLINE MSR generates at most one timer interrupt.

In TSC-deadline mode, writing 0 to the IA32_TSC_DEADLINE MSR disarms the local-APIC timer. Transitioning between TSC-deadline mode and other timer modes also disarms the timer.

The hardware reset value of the IA32_TSC_DEADLINE MSR is 0. In other timer modes (LVT bit 18 = 0), the IA32_TSC_DEADLINE MSR reads zero and writes are ignored.

Software can configure the TSC-deadline timer to deliver a single interrupt using the following algorithm:

1. Detect support for TSC-deadline mode by verifying CPUID.01H:ECX[24] = 1.
2. Select the TSC-deadline mode by programming bits 18:17 of the LVT Timer register with 10b.
3. Program the IA32_TSC_DEADLINE MSR with the target TSC value at which the timer interrupt is desired. This causes the processor to arm the timer.
4. The processor generates a timer interrupt when the value of time-stamp counter is greater than or equal to that of IA32_TSC_DEADLINE. It then disarms the timer and clear the IA32_TSC_DEADLINE MSR. (Both the time-stamp counter and the IA32_TSC_DEADLINE MSR are 64-bit unsigned integers.)
5. Software can re-arm the timer by repeating step 3.

The following are usage guidelines for TSC-deadline mode:

- Writes to the IA32_TSC_DEADLINE MSR are not serialized. Therefore, system software should not use WRMSR to the IA32_TSC_DEADLINE MSR as a serializing instruction. Read and write accesses to the IA32_TSC_DEADLINE and other MSR registers will occur in program order.
- Software can disarm the timer at any time by writing 0 to the IA32_TSC_DEADLINE MSR.
- If timer is armed, software can change the deadline (forward or backward) by writing a new value to the IA32_TSC_DEADLINE MSR.
- If software disarms the timer or postpones the deadline, race conditions may result in delivery of a timer interrupt associated with the original deadline. If the deadline has been postponed, software can identify such interrupts by reading the time-stamp counter and comparing its value to the new deadline.¹
- In xAPIC mode (in which the local-APIC registers are memory-mapped), software must order the memory-mapped write to the LVT entry that enables TSC-deadline mode and any subsequent WRMSR to the IA32_TSC_DEADLINE MSR. Software can assure proper ordering by executing the MFENCE instruction after the memory-mapped write and before any WRMSR. (In x2APIC mode, the WRMSR instruction is used to write to the LVT entry. The processor ensures the ordering of this write and any subsequent WRMSR to the deadline; no fencing is required.)

1. If the logical processor is in VMX non-root operation, a read of the time-stamp counter (using either RDMSR, RDTSC, or RDTSCP) may not return the actual value of the time-stamp counter; see Chapter 27 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C. It is the responsibility of software operating in VMX root operation to coordinate the virtualization of the time-stamp counter and the IA32_TSC_DEADLINE MSR.

12.5.5 Local Interrupt Acceptance

When a local interrupt is sent to the processor core, it is subject to the acceptance criteria specified in the interrupt acceptance flow chart in Figure 12-17. If the interrupt is accepted, it is logged into the IRR register and handled by the processor according to its priority (see Section 12.8.4, “Interrupt Acceptance for Fixed Interrupts”). If the interrupt is not accepted, it is sent back to the local APIC and retried.

12.6 ISSUING INTERPROCESSOR INTERRUPTS

The following sections describe the local APIC facilities that are provided for issuing interprocessor interrupts (IPIs) from software. The primary local APIC facility for issuing IPIs is the interrupt command register (ICR). The ICR can be used for the following functions:

- To send an interrupt to another processor.
- To allow a processor to forward an interrupt that it received but did not service to another processor for servicing.
- To direct the processor to interrupt itself (perform a self interrupt).
- To deliver special IPIs, such as the start-up IPI (SIPI) message, to other processors.

Interrupts generated with this facility are delivered to the other processors in the system through the system bus (for Pentium 4 and Intel Xeon processors) or the APIC bus (for P6 family and Pentium processors). The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.

12.6.1 Interrupt Command Register (ICR)

The interrupt command register (ICR) is a 64-bit¹ local APIC register (see Figure 12-12) that allows software running on the processor to specify and send interprocessor interrupts (IPIs) to other processors in the system.

1. In XAPIC mode the ICR is addressed as two 32-bit registers, ICR_LOW (FFE0 0300H) and ICR_HIGH (FFE0 0310H). In x2APIC mode, the ICR uses MSR 830H.

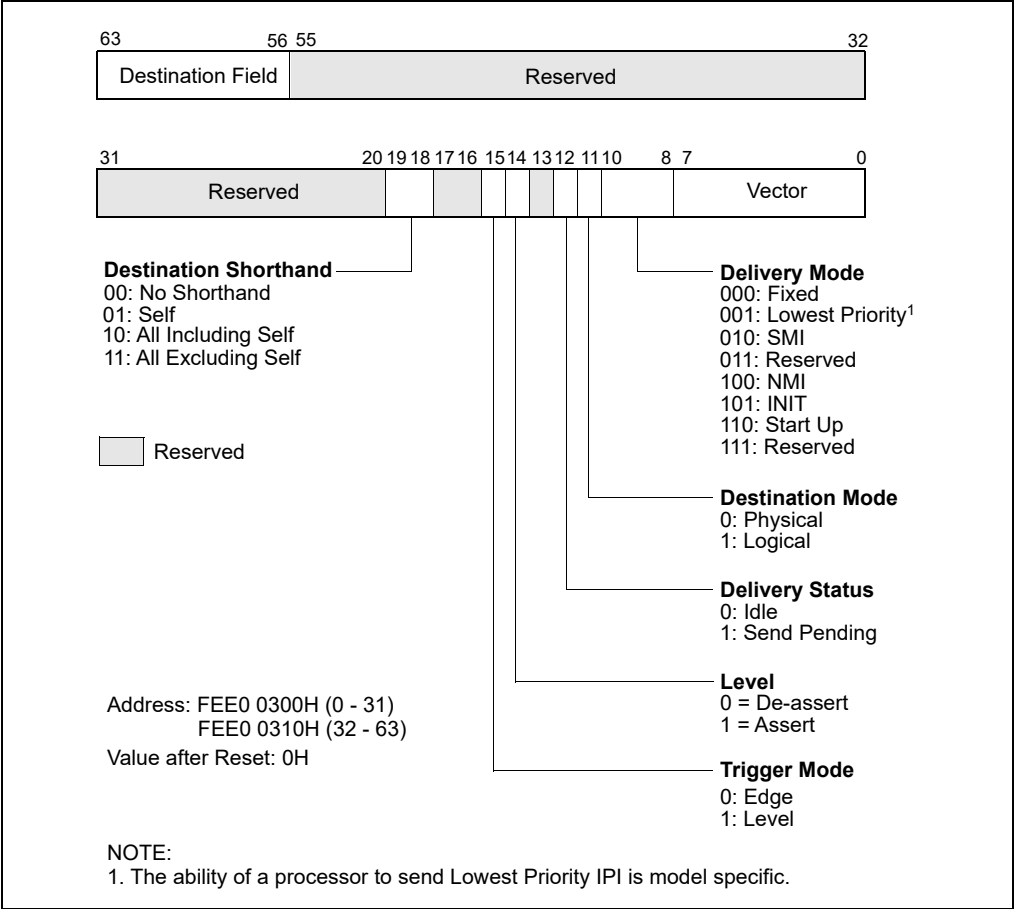


Figure 12-12. Interrupt Command Register (ICR)

To send an IPI, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. (All fields of the ICR are read-write by software with the exception of the delivery status field, which is read-only.) The act of writing to the low doubleword of the ICR causes the IPI to be sent.

The ICR consists of the following fields.

Vector	The vector number of the interrupt being sent.
Delivery Mode	Specifies the type of IPI to be sent. This field is also known as the IPI message type field. 000 (Fixed) Delivers the interrupt specified in the vector field to the target processor or processors. 001 (Lowest Priority) Same as fixed mode, except that the interrupt is delivered to the processor executing at the lowest priority among the set of processors specified in the destination field. The ability for a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software. 010 (SMI) Delivers an SMI interrupt to the target processor or processors. The vector field must be programmed to 00H for future compatibility. 011 (Reserved) 100 (NMI) Delivers an NMI interrupt to the target processor or processors. The vector information is ignored.

101 (INIT) Delivers an INIT request to the target processor or processors, which causes them to perform an INIT. As a result of this IPI message, all the target processors perform an INIT. The vector field must be programmed to 00H for future compatibility.

101 (INIT Level De-assert)

(Not supported in the Pentium 4 and Intel Xeon processors.) Sends a synchronization message to all the local APICs in the system to set their arbitration IDs (stored in their Arb ID registers) to the values of their APIC IDs (see Section 12.7, “System and APIC Bus Arbitration”). For this delivery mode, the level flag must be set to 0 and trigger mode flag to 1. This IPI is sent to all processors, regardless of the value in the destination field or the destination shorthand field; however, software should specify the “all including self” shorthand.

110 (Start-Up)

Sends a special “start-up” IPI (called a SIPI) to the target processor or processors. The vector typically points to a start-up routine that is part of the BIOS boot-strap code (see Section 10.4, “Multiple-Processor (MP) Initialization”). IPIs sent with this delivery mode are not automatically retried if the source APIC is unable to deliver it. It is up to the software to determine if the SIPI was not successfully delivered and to reissue the SIPI if necessary.

Destination Mode Selects either physical (0) or logical (1) destination mode (see Section 12.6.2, “Determining IPI Destination”).

Delivery Status (Read Only)

Indicates the IPI delivery status, as follows:

0 (Idle) Indicates that this local APIC has completed sending any previous IPIs.

1 (Send Pending)

Indicates that this local APIC has not completed sending the last IPI.

Level

For the INIT level de-assert delivery mode this flag must be set to 0; for all other delivery modes it must be set to 1. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 1.)

Trigger Mode

Selects the trigger mode when using the INIT level de-assert delivery mode: edge (0) or level (1). It is ignored for all other delivery modes. (This flag has no meaning in Pentium 4 and Intel Xeon processors, and will always be issued as a 0.)

Destination Shorthand

Indicates whether a shorthand notation is used to specify the destination of the interrupt and, if so, which shorthand is used. Destination shorthands are used in place of the 8-bit destination field, and can be sent by software using a single write to the low doubleword of the ICR. Shorthands are defined for the following cases: software self interrupt, IPIs to all processors in the system including the sender, IPIs to all processors in the system excluding the sender.

00: (No Shorthand)

The destination is specified in the destination field.

01: (Self)

The issuing APIC is the one and only destination of the IPI. This destination shorthand allows software to interrupt the processor on which it is executing. An APIC implementation is free to deliver the self-interrupt message internally or to issue the message to the bus and “snoop” it as with any other IPI message.

10: (All Including Self)

The IPI is sent to all processors in the system including the processor sending the IPI. The APIC will broadcast an IPI message with the destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors.

11: (All Excluding Self)

The IPI is sent to all processors in a system with the exception of the pro-

cessor sending the IPI. The APIC broadcasts a message with the physical destination mode and destination field set to FH for Pentium and P6 family processors and to FFH for Pentium 4 and Intel Xeon processors. Support for this destination shorthand in conjunction with the lowest-priority delivery mode is model specific. For Pentium 4 and Intel Xeon processors, when this shorthand is used together with lowest priority delivery mode, the IPI may be redirected back to the issuing processor.

Destination Specifies the target processor or processors. This field is only used when the destination shorthand field is set to 00B. If the destination mode is set to physical, then bits 56 through 59 contain the APIC ID of the target processor for Pentium and P6 family processors and bits 56 through 63 contain the APIC ID of the target processor the for Pentium 4 and Intel Xeon processors. If the destination mode is set to logical, the interpretation of the 8-bit destination field depends on the settings of the DFR and LDR registers of the local APICs in all the processors in the system (see Section 12.6.2, “Determining IPI Destination”).

Not all combinations of options for the ICR are valid. Table 12-3 shows the valid combinations for the fields in the ICR for the Pentium 4 and Intel Xeon processors; Table 12-4 shows the valid combinations for the fields in the ICR for the P6 family processors. Also note that the lower half of the ICR may not be preserved over transitions to the deepest C-States.

ICR operation in x2APIC mode is discussed in Section 12.12.9.

Table 12-3. Valid Combinations for Pentium 4 and Intel Xeon Processors Local xAPIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes ¹	Physical or Logical
No Shorthand	Invalid ²	Level	All Modes	Physical or Logical
Self	Valid	Edge	Fixed	X ³
Self	Invalid ²	Level	Fixed	X
Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Including Self	Valid	Edge	Fixed	X
All Including Self	Invalid ²	Level	Fixed	X
All Including Self	Invalid	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All Excluding Self	Valid	Edge	Fixed, Lowest Priority ^{1, 4} , NMI, INIT, SMI, Start-Up	X
All Excluding Self	Invalid ²	Level	Fixed, Lowest Priority ⁴ , NMI, INIT, SMI, Start-Up	X

NOTES:

1. The ability of a processor to send a lowest priority IPI is model specific.
2. For these interrupts, if the trigger mode bit is 1 (Level), the local xAPIC will override the bit setting and issue the interrupt as an edge triggered interrupt.
3. X means the setting is ignored.
4. When using the “lowest priority” delivery mode and the “all excluding self” destination, the IPI can be redirected back to the issuing APIC, which is essentially the same as the “all including self” destination mode.

Table 12-4. Valid Combinations for the P6 Family Processor Local APIC Interrupt Command Register

Destination Shorthand	Valid/Invalid	Trigger Mode	Delivery Mode	Destination Mode
No Shorthand	Valid	Edge	All Modes ¹	Physical or Logical
No Shorthand	Valid ²	Level	Fixed, Lowest Priority ¹ , NMI	Physical or Logical
No Shorthand	Valid ³	Level	INIT	Physical or Logical
Self	Valid	Edge	Fixed	X ⁴
Self	Valid ²	Level	Fixed	X
Self	Invalid ⁵	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All including Self	Valid	Edge	Fixed	X
All including Self	Valid ²	Level	Fixed	X
All including Self	Invalid ⁵	X	Lowest Priority, NMI, INIT, SMI, Start-Up	X
All excluding Self	Valid	Edge	All Modes ¹	X
All excluding Self	Valid ²	Level	Fixed, Lowest Priority ¹ , NMI	X
All excluding Self	Invalid ⁵	Level	SMI, Start-Up	X
All excluding Self	Valid ³	Level	INIT	X
X	Invalid ⁵	Level	SMI, Start-Up	X

NOTES:

1. The ability of a processor to send a lowest priority IPI is model specific.
2. Treated as edge triggered if level bit is set to 1, otherwise ignored.
3. Treated as edge triggered when Level bit is set to 1; treated as “INIT Level Deassert” message when level bit is set to 0 (deassert). Only INIT level deassert messages are allowed to have the level bit set to 0. For all other messages the level bit must be set to 1.
4. X means the setting is ignored.
5. The behavior of the APIC is undefined.

12.6.2 Determining IPI Destination

The destination of an IPI¹ can be one, all, or a subset (group) of the processors on the system bus. The sender of the IPI specifies the destination of an IPI with the following APIC registers and fields within the registers:

- **ICR Register** — The following fields in the ICR register are used to specify the destination of an IPI.
 - **Destination Mode** — Selects one of two destination modes (physical or logical).
 - **Destination Field** — In physical destination mode, used to specify the APIC ID of the destination processor; in logical destination mode, used to specify a message destination address (MDA) that can be used to select specific processors in clusters.
 - **Destination Shorthand** — A quick method of specifying all processors, all excluding self, or self as the destination.
 - **Delivery mode, Lowest Priority** — Architecturally specifies that a lowest-priority arbitration mechanism be used to select a destination processor from a specified group of processors. The ability of a processor to send a lowest priority IPI is model specific and should be avoided by BIOS and operating system software.
- **Local destination register (LDR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.
- **Destination format register (DFR)** — Used in conjunction with the logical destination mode and MDAs to select the destination processors.

1. Determination of IPI destinations in x2APIC mode is discussed in Section 10.12.10.

How the ICR, LDR, and DFR are used to select an IPI destination depends on the destination mode used: physical, logical, broadcast/self, or lowest-priority delivery mode. These destination modes are described in the following sections.

12.6.2.1 Physical Destination Mode

In physical destination mode, the destination processor is specified by its local APIC ID (see Section 12.4.6, “Local APIC ID”). For Pentium 4 and Intel Xeon processors, either a single destination (local APIC IDs 00H through FEH) or a broadcast to all APICs (the APIC ID is FFH) may be specified in physical destination mode.

A broadcast IPI (bits 28-31 of the MDA are 1's) or I/O subsystem initiated interrupt with lowest priority delivery mode is not supported in physical destination mode and must not be configured by software. Also, for any non-broadcast IPI or I/O subsystem initiated interrupt with lowest priority delivery mode, software must ensure that APICs defined in the interrupt address are present and enabled to receive interrupts.

For the P6 family and Pentium processors, a single destination is specified in physical destination mode with a local APIC ID of 0H through 0EH, allowing up to 15 local APICs to be addressed on the APIC bus. A broadcast to all local APICs is specified with 0FH.

NOTE

The number of local APICs that can be addressed on the system bus may be restricted by hardware.

12.6.2.2 Logical Destination Mode

In logical destination mode, IPI destination is specified using an 8-bit message destination address (MDA), which is entered in the destination field of the ICR. Upon receiving an IPI message that was sent using logical destination mode, a local APIC compares the MDA in the message with the values in its LDR and DFR to determine if it should accept and handle the IPI. For both configurations of logical destination mode, when combined with lowest priority delivery mode, software is responsible for ensuring that all of the local APICs included in or addressed by the IPI or I/O subsystem interrupt are present and enabled to receive the interrupt.

Figure 12-13 shows the layout of the logical destination register (LDR). The 8-bit logical APIC ID field in this register is used to create an identifier that can be compared with the MDA.

NOTE

The logical APIC ID should not be confused with the local APIC ID that is contained in the local APIC ID register.

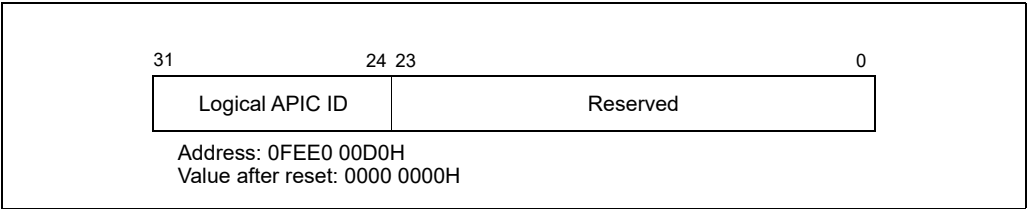


Figure 12-13. Logical Destination Register (LDR)

Figure 12-14 shows the layout of the destination format register (DFR). The 4-bit model field in this register selects one of two models (flat or cluster) that can be used to interpret the MDA when using logical destination mode.

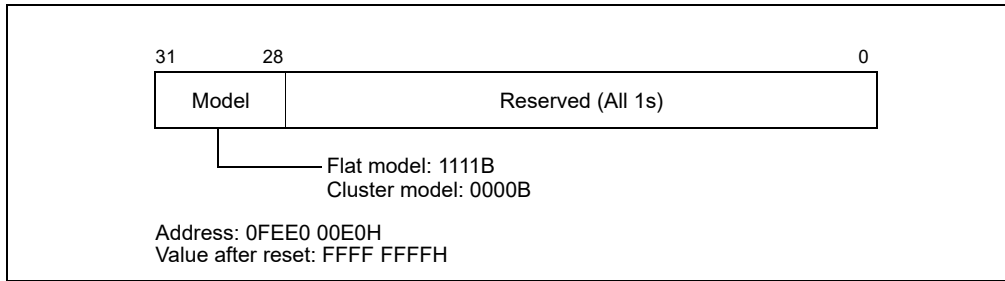


Figure 12-14. Destination Format Register (DFR)

The interpretation of MDA for the two models is described in the following paragraphs.

1. **Flat Model** — This model is selected by programming DFR bits 28 through 31 to 1111. Here, a unique logical APIC ID can be established for up to 8 local APICs by setting a different bit in the logical APIC ID field of the LDR for each local APIC. A group of local APICs can then be selected by setting one or more bits in the MDA. Each local APIC performs a bit-wise AND of the MDA and its logical APIC ID. If a true condition (non-zero) is detected, the local APIC accepts the IPI message. A broadcast to all APICs is achieved by setting the MDA to 1s.
2. **Cluster Model** — This model is selected by programming DFR bits 28 through 31 to 0000. This model supports two basic destination schemes: flat cluster and hierarchical cluster.

The flat cluster destination model is only supported for P6 family and Pentium processors. Using this model, all APICs are assumed to be connected through the APIC bus. Bits 60 through 63 of the MDA contains the encoded address of the destination cluster and bits 56 through 59 identify up to four local APICs within the cluster (each bit is assigned to one local APIC in the cluster, as in the flat connection model). To identify one or more local APICs, bits 60 through 63 of the MDA are compared with bits 28 through 31 of the LDR to determine if a local APIC is part of the cluster. Bits 56 through 59 of the MDA are compared with Bits 24 through 27 of the LDR to identify a local APICs within the cluster.

Sets of processors within a cluster can be specified by writing the target cluster address in bits 60 through 63 of the MDA and setting selected bits in bits 56 through 59 of the MDA, corresponding to the chosen members of the cluster. In this mode, 15 clusters (with cluster addresses of 0 through 14) each having 4 local APICs can be specified in the message. For the P6 and Pentium processor's local APICs, however, the APIC arbitration ID supports only 15 APIC agents. Therefore, the total number of processors and their local APICs supported in this mode is limited to 15. Broadcast to all local APICs is achieved by setting all destination bits to one. This guarantees a match on all clusters and selects all APICs in each cluster. A broadcast IPI or I/O subsystem broadcast interrupt with lowest priority delivery mode is not supported in cluster mode and must not be configured by software.

The hierarchical cluster destination model can be used with Pentium 4, Intel Xeon, P6 family, or Pentium processors. With this model, a hierarchical network can be created by connecting different flat clusters via independent system or APIC buses. This scheme requires a cluster manager within each cluster, which is responsible for handling message passing between system or APIC buses. One cluster contains up to 4 agents. Thus 15 cluster managers, each with 4 agents, can form a network of up to 60 APIC agents. Note that hierarchical APIC networks requires a special cluster manager device, which is not part of the local or the I/O APIC units.

NOTES

All processors that have their APIC software enabled (using the spurious vector enable/disable bit) must have their DFRs (Destination Format Registers) programmed identically.

The default mode for DFR is flat mode. If you are using cluster mode, DFRs must be programmed before the APIC is software enabled. Since some chipsets do not accurately track a system view of the logical mode, program DFRs as soon as possible after starting the processor.

12.6.2.3 Broadcast/Self Delivery Mode

The destination shorthand field of the ICR allows the delivery mode to be by-passed in favor of broadcasting the IPI to all the processors on the system bus and/or back to itself (see Section 12.6.1, "Interrupt Command Register (ICR)"). Three destination shorthands are supported: self, all excluding self, and all including self. The destination mode is ignored when a destination shorthand is used.

12.6.2.4 Lowest Priority Delivery Mode

With lowest priority delivery mode, the ICR is programmed to send an IPI to several processors on the system bus, using the logical or shorthand destination mechanism for selecting the processor. The selected processors then arbitrate with one another over the system bus or the APIC bus, with the lowest-priority processor accepting the IPI.

For systems based on the Intel Xeon processor, the chipset bus controller accepts messages from the I/O APIC agents in the system and directs interrupts to the processors on the system bus. When using the lowest priority delivery mode, the chipset chooses a target processor to receive the interrupt out of the set of possible targets. The Pentium 4 processor provides a special bus cycle on the system bus that informs the chipset of the current task priority for each logical processor in the system. The chipset saves this information and uses it to choose the lowest priority processor when an interrupt is received.

For systems based on P6 family processors, the processor priority used in lowest-priority arbitration is contained in the arbitration priority register (APR) in each local APIC. Figure 12-15 shows the layout of the APR.

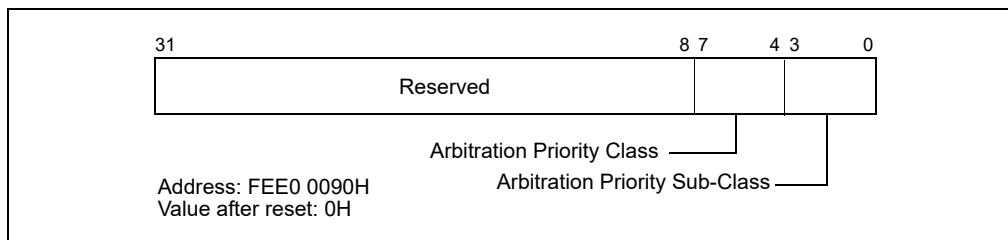


Figure 12-15. Arbitration Priority Register (APR)

The APR value is computed as follows:

```

IF (TPR[7:4] ≥ IRRV[7:4]) AND (TPR[7:4] > ISRV[7:4])
    THEN
        APR[7:0] ← TPR[7:0]
    ELSE
        APR[7:4] ← max(TPR[7:4] AND ISRV[7:4], IRRV[7:4])
        APR[3:0] ← 0.
    
```

Here, the TPR value is the task priority value in the TPR (see Figure 12-18), the IRRV value is the vector number for the highest priority bit that is set in the IRR (see Figure 12-20) or 00H (if no IRR bit is set), and the ISRV value is the vector number for the highest priority bit that is set in the ISR (see Figure 12-20). Following arbitration among the destination processors, the processor with the lowest value in its APR handles the IPI and the other processors ignore it.

(P6 family and Pentium processors.) For these processors, if a **focus processor** exists, it may accept the interrupt, regardless of its priority. A processor is said to be the focus of an interrupt if it is currently servicing that interrupt or if it has a pending request for that interrupt. For Intel Xeon processors, the concept of a focus processor is not supported.

In operating systems that use the lowest priority delivery mode but do not update the TPR, the TPR information saved in the chipset will potentially cause the interrupt to be always delivered to the same processor from the logical set. This behavior is functionally backward compatible with the P6 family processor but may result in unexpected performance implications.

12.6.3 IPI Delivery and Acceptance

When the low double-word of the ICR is written to, the local APIC creates an IPI message from the information contained in the ICR and sends the message out on the system bus (Pentium 4 and Intel Xeon processors) or the APIC bus (P6 family and Pentium processors). The manner in which these IPIs are handled after being issued is described in Section 12.8, “Handling Interrupts.”

12.7 SYSTEM AND APIC BUS ARBITRATION

When several local APICs and the I/O APIC are sending IPI and interrupt messages on the system bus (or APIC bus), the order in which the messages are sent and handled is determined through bus arbitration.

For the Pentium 4 and Intel Xeon processors, the local and I/O APICs use the arbitration mechanism defined for the system bus to determine the order in which IPIs are handled. This mechanism is non-architectural and cannot be controlled by software.

For the P6 family and Pentium processors, the local and I/O APICs use an APIC-based arbitration mechanism to determine the order in which IPIs are handled. Here, each local APIC is given an arbitration priority of from 0 to 15, which the I/O APIC uses during arbitration to determine which local APIC should be given access to the APIC bus. The local APIC with the highest arbitration priority always wins bus access. Upon completion of an arbitration round, the winning local APIC lowers its arbitration priority to 0 and the losing local APICs each raise theirs by 1.

The current arbitration priority for a local APIC is stored in a 4-bit, software-transparent arbitration ID (Arb ID) register. During reset, this register is initialized to the APIC ID number (stored in the local APIC ID register). The INIT level-deassert IPI, which is issued with an ICR command, can be used to resynchronize the arbitration priorities of the local APICs by resetting Arb ID register of each agent to its current APIC ID value. (The Pentium 4 and Intel Xeon processors do not implement the Arb ID register.)

Section 12.10, “APIC Bus Message Passing Mechanism and Protocol (P6 Family, Pentium Processors),” describes the APIC bus arbitration protocols and bus message formats, while Section 12.6.1, “Interrupt Command Register (ICR),” describes the INIT level de-assert IPI message.

Note that except for the SIPI IPI (see Section 12.6.1, “Interrupt Command Register (ICR)”), all bus messages that fail to be delivered to their specified destination or destinations are automatically retried. Software should avoid situations in which IPIs are sent to disabled or nonexistent local APICs, causing the messages to be resent repeatedly. Additionally, interrupt sources that target the APIC should be masked or changed to no longer target the APIC.

12.8 HANDLING INTERRUPTS

When a local APIC receives an interrupt from a local source, an interrupt message from an I/O APIC, or an IPI, the manner in which it handles the message depends on processor implementation, as described in the following sections.

12.8.1 Interrupt Handling with the Pentium 4 and Intel Xeon Processors

With the Pentium 4 and Intel Xeon processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows:

1. It determines if it is the specified destination or not (see Figure 12-16). If it is the specified destination, it accepts the message; if it is not, it discards the message.

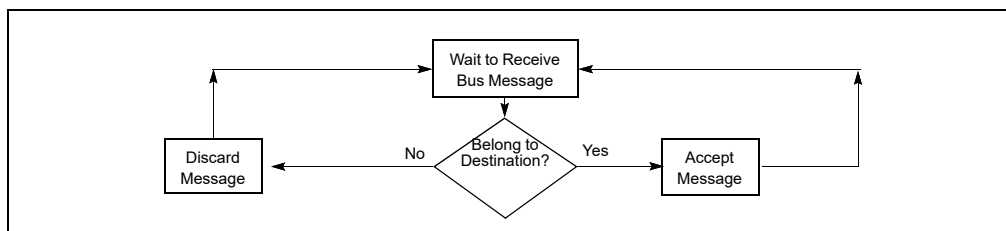


Figure 12-16. Interrupt Acceptance Flow Chart for the Local APIC (Pentium 4 and Intel Xeon Processors)

2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or SIPI, the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC sets the appropriate bit in the IRR.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 12.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 12.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its ISR queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

12.8.2 Interrupt Handling with the P6 Family and Pentium Processors

With the P6 family and Pentium processors, the local APIC handles the local interrupts, interrupt messages, and IPIs it receives as follows (see Figure 12-17).

1. (IPIs only) The local APIC examines the IPI message to determine if it is the specified destination for the IPI as described in Section 12.6.2, "Determining IPI Destination." If it is the specified destination, it continues its acceptance procedure; if it is not the destination, it discards the IPI message. When the message specifies lowest-priority delivery mode, the local APIC will arbitrate with the other processors that were designated as recipients of the IPI message (see Section 12.6.2.4, "Lowest Priority Delivery Mode").
2. If the local APIC determines that it is the designated destination for the interrupt and if the interrupt request is an NMI, SMI, INIT, ExtINT, or INIT-deassert interrupt, or one of the MP protocol IPI messages (BIPI, FIPI, and SIPI), the interrupt is sent directly to the processor core for handling.
3. If the local APIC determines that it is the designated destination for the interrupt but the interrupt request is not one of the interrupts given in step 2, the local APIC looks for an open slot in one of its two pending interrupt queues contained in the IRR and ISR registers (see Figure 12-20). If a slot is available (see Section 12.8.4, "Interrupt Acceptance for Fixed Interrupts"), places the interrupt in the slot. If a slot is not available, it rejects the interrupt request and sends it back to the sender with a retry message.
4. When interrupts are pending in the IRR register, the local APIC dispatches them to the processor one at a time, based on their priority and the current processor priority in the PPR (see Section 12.8.3.1, "Task and Processor Priorities").
5. When a fixed interrupt has been dispatched to the processor core for handling, the completion of the handler routine is indicated with an instruction in the instruction handler code that writes to the end-of-interrupt (EOI) register in the local APIC (see Section 12.8.5, "Signaling Interrupt Servicing Completion"). The act of writing to the EOI register causes the local APIC to delete the interrupt from its queue and (for level-triggered interrupts) send a message on the bus indicating that the interrupt handling has been completed. (A write to the EOI register must not be included in the handler routine for an NMI, SMI, INIT, ExtINT, or SIPI.)

The following sections describe the acceptance of interrupts and their handling by the local APIC and processor in greater detail.

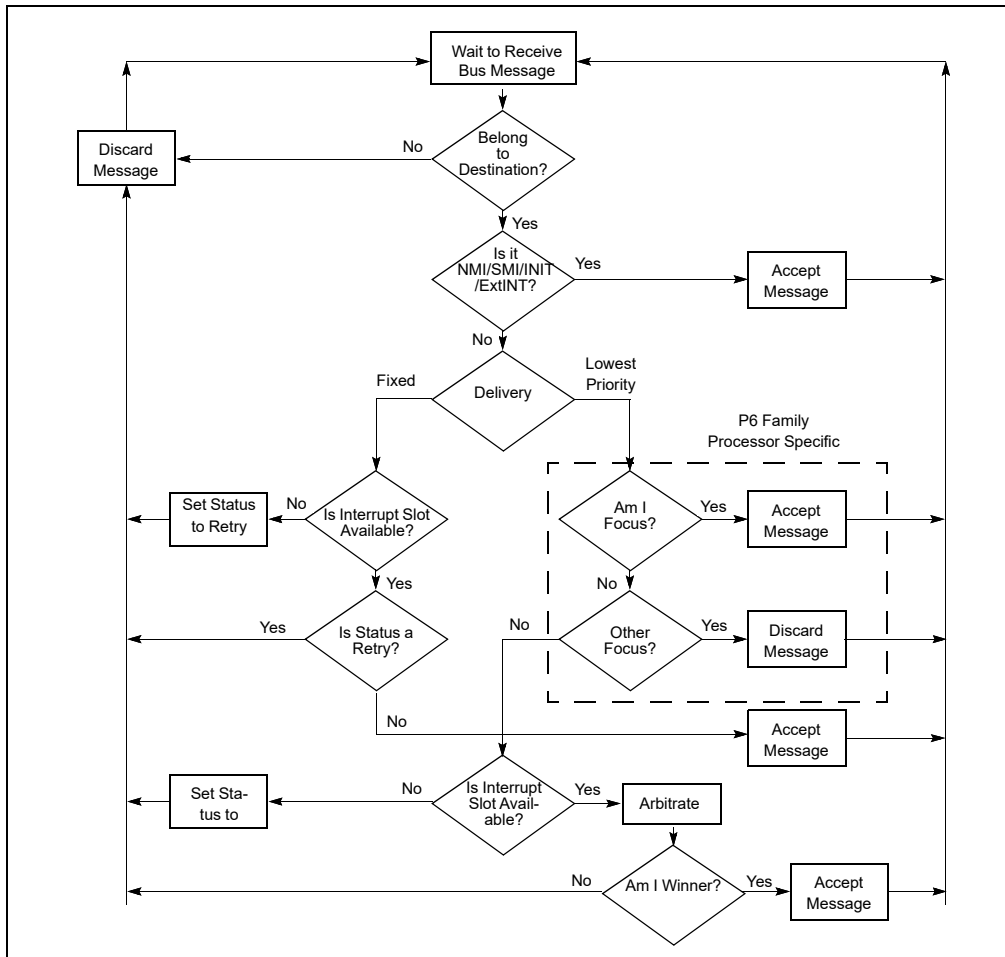


Figure 12-17. Interrupt Acceptance Flow Chart for the Local APIC (P6 Family and Pentium Processors)

12.8.3 Interrupt, Task, and Processor Priority

Each interrupt delivered to the processor through the local APIC has a priority based on its vector number. The local APIC uses this priority to determine when to service the interrupt relative to the other activities of the processor, including the servicing of other interrupts.

Each interrupt vector is an 8-bit value. The **interrupt-priority class** is the value of bits 7:4 of the interrupt vector. The lowest interrupt-priority class is 1 and the highest is 15; interrupts with vectors in the range 0–15 (with interrupt-priority class 0) are illegal and are never delivered. Because vectors 0–31 are reserved for dedicated uses by the Intel 64 and IA-32 architectures, software should configure interrupt vectors to use interrupt-priority classes in the range 2–15.

Each interrupt-priority class encompasses 16 vectors. The relative priority of interrupts within an interrupt-priority class is determined by the value of bits 3:0 of the vector number. The higher the value of those bits, the higher the priority within that interrupt-priority class. Thus, each interrupt vector comprises two parts, with the high 4 bits indicating its interrupt-priority class and the low 4 bits indicating its ranking within the interrupt-priority class.

12.8.3.1 Task and Processor Priorities

The local APIC also defines a **task priority** and a **processor priority** that determine the order in which interrupts are handled. The **task-priority class** is the value of bits 7:4 of the task-priority register (TPR), which can be written by software (TPR is a read/write register); see Figure 12-18.

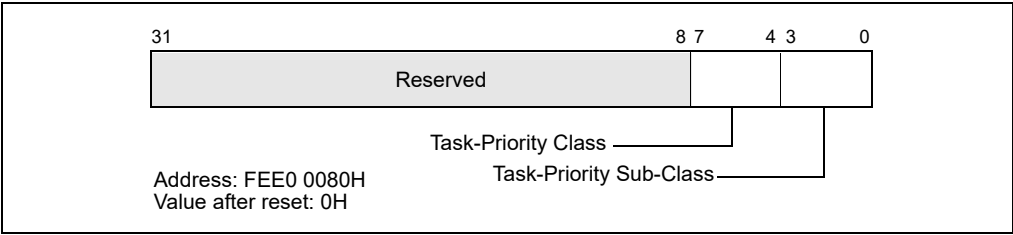


Figure 12-18. Task-Priority Register (TPR)

NOTE

In this discussion, the term “task” refers to a software defined task, process, thread, program, or routine that is dispatched to run on the processor by the operating system. It does not refer to an IA-32 architecture defined task as described in Chapter 9, “Task Management.”

The task priority allows software to set a priority threshold for interrupting the processor. This mechanism enables the operating system to temporarily block low priority interrupts from disturbing high-priority work that the processor is doing. The ability to block such interrupts using task priority results from the way that the TPR controls the value of the processor-priority register (PPR).¹

The **processor-priority class** is a value in the range 0–15 that is maintained in bits 7:4 of the processor-priority register (PPR); see Figure 12-19. The PPR is a read-only register. The processor-priority class represents the current priority at which the processor is executing.

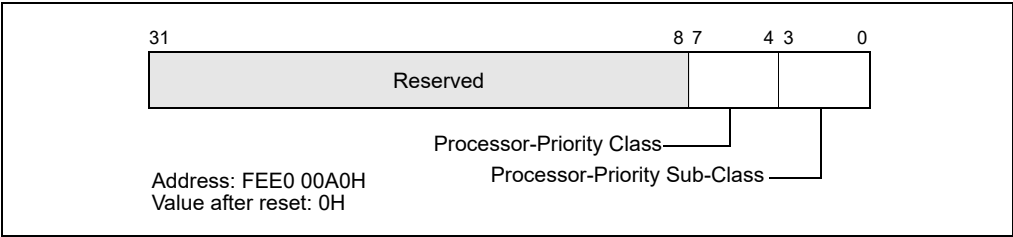


Figure 12-19. Processor-Priority Register (PPR)

The value of the PPR is based on the value of TPR and the value ISRV; ISRV is the vector number of the highest priority bit that is set in the ISR or 00H if no bit is set in the ISR. (See Section 12.8.4 for more details on the ISR.) The value of PPR is determined as follows:

- PPR[7:4] (the processor-priority class) the maximum of TPR[7:4] (the task- priority class) and ISRV[7:4] (the priority of the highest priority interrupt in service).
- PPR[3:0] (the processor-priority sub-class) is determined as follows:
 - If TPR[7:4] > ISRV[7:4], PPR[3:0] is TPR[3:0] (the task-priority sub-class).
 - If TPR[7:4] < ISRV[7:4], PPR[3:0] is 0.
 - If TPR[7:4] = ISRV[7:4], PPR[3:0] may be either TPR[3:0] or 0. The actual behavior is model-specific.

The processor-priority class determines the priority threshold for interrupting the processor. The processor will deliver only those interrupts that have an interrupt-priority class higher than the processor-priority class in the PPR. If the processor-priority class is 0, the PPR does not inhibit the delivery any interrupt; if it is 15, the processor inhibits the delivery of all interrupts. (The processor-priority mechanism does not affect the delivery of interrupts with the NMI, SMI, INIT, ExtINT, INIT-deassert, and start-up delivery modes.)

The processor does not use the processor-priority sub-class to determine which interrupts to deliver and which to inhibit. (The processor uses the processor-priority sub-class only to satisfy reads of the PPR.)

1. The TPR also determines the arbitration priority of the local processor; see Section 12.6.2.4, “Lowest Priority Delivery Mode.”

12.8.4 Interrupt Acceptance for Fixed Interrupts

The local APIC queues the fixed interrupts that it accepts in one of two interrupt pending registers: the interrupt request register (IRR) or in-service register (ISR). These two 256-bit read-only registers are shown in Figure 12-20. The 256 bits in these registers represent the 256 possible vectors; vectors 0 through 15 are reserved by the APIC (see also: Section 12.5.2, “Valid Interrupt Vectors”).

NOTE

All interrupts with an NMI, SMI, INIT, ExtINT, start-up, or INIT-deassert delivery mode bypass the IRR and ISR registers and are sent directly to the processor core for servicing.

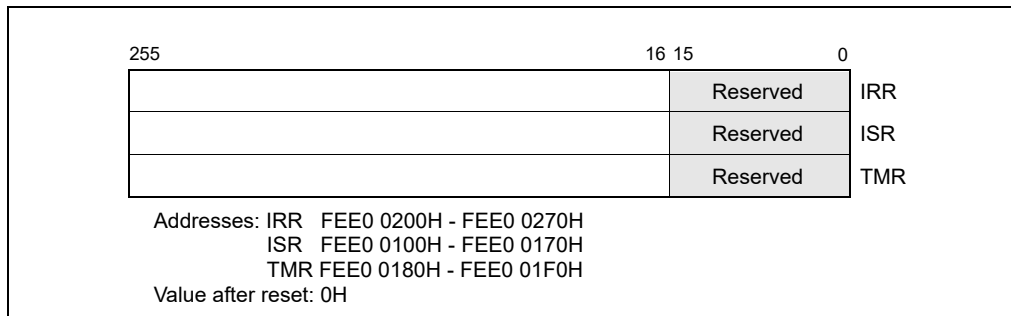


Figure 12-20. IRR, ISR, and TMR Registers

The IRR contains the active interrupt requests that have been accepted, but not yet dispatched to the processor for servicing. When the local APIC accepts an interrupt, it sets the bit in the IRR that corresponds to the vector of the accepted interrupt. When the processor core is ready to handle the next interrupt, the local APIC clears the highest priority IRR bit that is set and sets the corresponding ISR bit. The vector for the highest priority bit set in the ISR is then dispatched to the processor core for servicing.

While the processor is servicing the highest priority interrupt, the local APIC can send additional fixed interrupts by setting bits in the IRR. When the interrupt service routine issues a write to the EOI register (see Section 12.8.5, “Signaling Interrupt Servicing Completion”), the local APIC responds by clearing the highest priority ISR bit that is set. It then repeats the process of clearing the highest priority bit in the IRR and setting the corresponding bit in the ISR. The processor core then begins executing the service routine for the highest priority bit set in the ISR.

If more than one interrupt is generated with the same vector number, the local APIC can set the bit for the vector both in the IRR and the ISR. This means that for the Pentium 4 and Intel Xeon processors, the IRR and ISR can queue two interrupts for each interrupt vector: one in the IRR and one in the ISR. Any additional interrupts issued for the same interrupt vector are collapsed into the single bit in the IRR.

For the P6 family and Pentium processors, the IRR and ISR registers can queue no more than two interrupts per interrupt vector and will reject other interrupts that are received within the same vector.

If the local APIC receives an interrupt with an interrupt-priority class higher than that of the interrupt currently in service, and interrupts are enabled in the processor core, the local APIC dispatches the higher priority interrupt to the processor immediately (without waiting for a write to the EOI register). The currently executing interrupt handler is then interrupted so the higher-priority interrupt can be handled. When the handling of the higher-priority interrupt has been completed, the servicing of the interrupted interrupt is resumed.

The trigger mode register (TMR) indicates the trigger mode of the interrupt (see Figure 12-20). Upon acceptance of an interrupt into the IRR, the corresponding TMR bit is cleared for edge-triggered interrupts and set for level-triggered interrupts. If a TMR bit is set when an EOI cycle for its corresponding interrupt vector is generated, an EOI message is sent to all I/O APICs.

12.8.5 Signaling Interrupt Servicing Completion

For all interrupts except those delivered with the NMI, SMI, INIT, ExtINT, the start-up, or INIT-Deassert delivery mode, the interrupt handler must include a write to the end-of-interrupt (EOI) register (see Figure 12-21). This

write must occur at the end of the handler routine, sometime before the IRET instruction. This action indicates that the servicing of the current interrupt is complete and the local APIC can issue the next interrupt from the ISR.

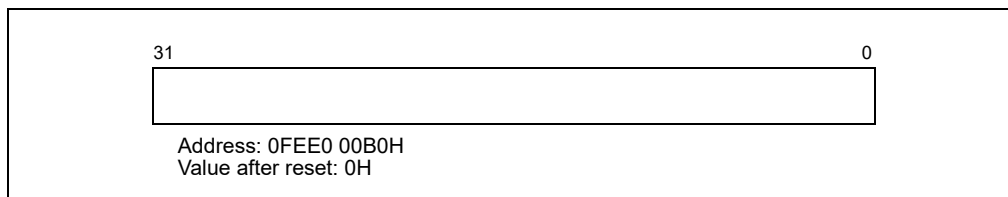


Figure 12-21. EOI Register

Upon receiving an EOI, the APIC clears the highest priority bit in the ISR and dispatches the next highest priority interrupt to the processor. If the terminated interrupt was a level-triggered interrupt, the local APIC also sends an end-of-interrupt message to all I/O APICs.

System software may prefer to direct EOIs to specific I/O APICs rather than having the local APIC send end-of-interrupt messages to all I/O APICs.

Software can inhibit the broadcast of EOI message by setting bit 12 of the Spurious Interrupt Vector Register (see Section 12.9). If this bit is set, a broadcast EOI is not generated on an EOI cycle even if the associated TMR bit indicates that the current interrupt was level-triggered. The default value for the bit is 0, indicating that EOI broadcasts are performed.

Bit 12 of the Spurious Interrupt Vector Register is reserved to 0 if the processor does not support suppression of EOI broadcasts. Support for EOI-broadcast suppression is reported in bit 24 in the Local APIC Version Register (see Section 12.4.8); the feature is supported if that bit is set to 1. When supported, the feature is available in both xAPIC mode and x2APIC mode.

System software desiring to perform directed EOIs for level-triggered interrupts should set bit 12 of the Spurious Interrupt Vector Register and follow each the EOI to the local xAPIC for a level triggered interrupt with a directed EOI to the I/O APIC generating the interrupt (this is done by writing to the I/O APIC's EOI register). System software performing directed EOIs must retain a mapping associating level-triggered interrupts with the I/O APICs in the system.

12.8.6 Task Priority in IA-32e Mode

In IA-32e mode, operating systems can manage the 16 interrupt-priority classes (see Section 12.8.3, "Interrupt, Task, and Processor Priority") explicitly using the task priority register (TPR). Operating systems can use the TPR to temporarily block specific (low-priority) interrupts from interrupting a high-priority task. This is done by loading TPR with a value in which the task-priority class corresponds to the highest interrupt-priority class that is to be blocked. For example:

- Loading the TPR with a task-priority class of 8 (01000B) blocks all interrupts with an interrupt-priority class of 8 or less while allowing all interrupts with an interrupt-priority class of 9 or more to be recognized.
- Loading the TPR with a task-priority class of 0 enables all external interrupts.
- Loading the TPR with a task-priority class of 0FH (01111B) disables all external interrupts.

The TPR (shown in Figure 12-18) is cleared to 0 on reset. In 64-bit mode, software can read and write the TPR using an alternate interface, MOV CR8 instruction. The new task-priority class is established when the MOV CR8 instruction completes execution. Software does not need to force serialization after loading the TPR using MOV CR8.

Use of the MOV CRn instruction requires a privilege level of 0. Programs running at privilege level greater than 0 cannot read or write the TPR. An attempt to do so causes a general-protection exception. The TPR is abstracted from the interrupt controller (IC), which prioritizes and manages external interrupt delivery to the processor. The IC can be an external device, such as an APIC or 8259. Typically, the IC provides a priority mechanism similar or identical to the TPR. The IC, however, is considered implementation-dependent with the under-lying priority mechanisms subject to change. CR8, by contrast, is part of the Intel 64 architecture. Software can depend on this definition remaining unchanged.

Figure 12-22 shows the layout of CR8; only the low four bits are used. The remaining 60 bits are reserved and must be written with zeros. Failure to do this causes a general-protection exception.

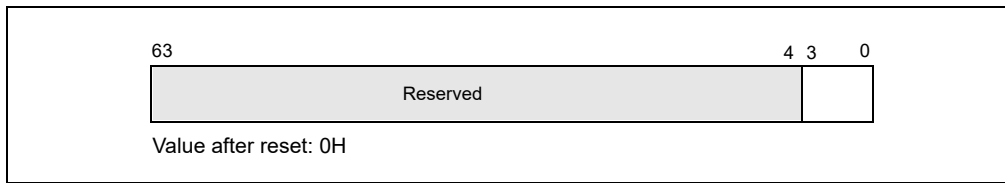


Figure 12-22. CR8 Register

12.8.6.1 Interaction of Task Priorities between CR8 and APIC

The first implementation of Intel 64 architecture includes a local advanced programmable interrupt controller (APIC) that is similar to the APIC used with previous IA-32 processors. Some aspects of the local APIC affect the operation of the architecturally defined task priority register and the programming interface using CR8.

Notable CR8 and APIC interactions are:

- The processor powers up with the local APIC enabled.
- The APIC must be enabled for CR8 to function as the TPR. Writes to CR8 are reflected into the APIC Task Priority Register.
- $\text{APIC.TPR}[\text{bits } 7:4] = \text{CR8}[\text{bits } 3:0]$, $\text{APIC.TPR}[\text{bits } 3:0] = 0$. A read of CR8 returns a 64-bit value which is the value of $\text{TPR}[\text{bits } 7:4]$, zero extended to 64 bits.

There are no ordering mechanisms between direct updates of the APIC.TPR and CR8. Operating software should implement either direct APIC TPR updates or CR8 style TPR updates but not mix them. Software can use a serializing instruction (for example, CPUID) to serialize updates between MOV CR8 and stores to the APIC.

12.9 SPURIOUS INTERRUPT

A special situation may occur when a processor raises its task priority to be greater than or equal to the level of the interrupt for which the processor INTR signal is currently being asserted. If at the time the INTA cycle is issued, the interrupt that was to be dispensed has become masked (programmed by software), the local APIC will deliver a spurious-interrupt vector. Dispensing the spurious-interrupt vector does not affect the ISR, so the handler for this vector should return without an EOI.

The vector number for the spurious-interrupt vector is specified in the spurious-interrupt vector register (see Figure 12-23). The functions of the fields in this register are as follows:

- Spurious Vector** Determines the vector number to be delivered to the processor when the local APIC generates a spurious vector.
- (Pentium 4 and Intel Xeon processors.) Bits 0 through 7 of the this field are programmable by software.
- (P6 family and Pentium processors). Bits 4 through 7 of the this field are programmable by software, and bits 0 through 3 are hardwired to logical ones. Software writes to bits 0 through 3 have no effect.

APIC Software Enable/Disable

Allows software to temporarily enable (1) or disable (0) the local APIC (see Section 12.4.3, “Enabling or Disabling the Local APIC”).

Focus Processor Checking

Determines if focus processor checking is enabled (0) or disabled (1) when using the lowest-priority delivery mode. In Pentium 4 and Intel Xeon processors, this bit is reserved and should be cleared to 0.

Suppress EOI Broadcasts

Determines whether an EOI for a level-triggered interrupt causes EOI messages to be broadcast to the I/O APICs (0) or not (1). See Section 12.8.5. The default value for this bit is 0, indicating that EOI broadcasts are performed. This bit is reserved to 0 if the processor does not support EOI-broadcast suppression.

NOTE

Do not program an LVT or IOAPIC RTE with a spurious vector even if you set the mask bit. A spurious vector ISR does not do an EOI. If for some reason an interrupt is generated by an LVT or RTE entry, the bit in the in-service register will be left set for the spurious vector. This will mask all interrupts at the same or lower priority

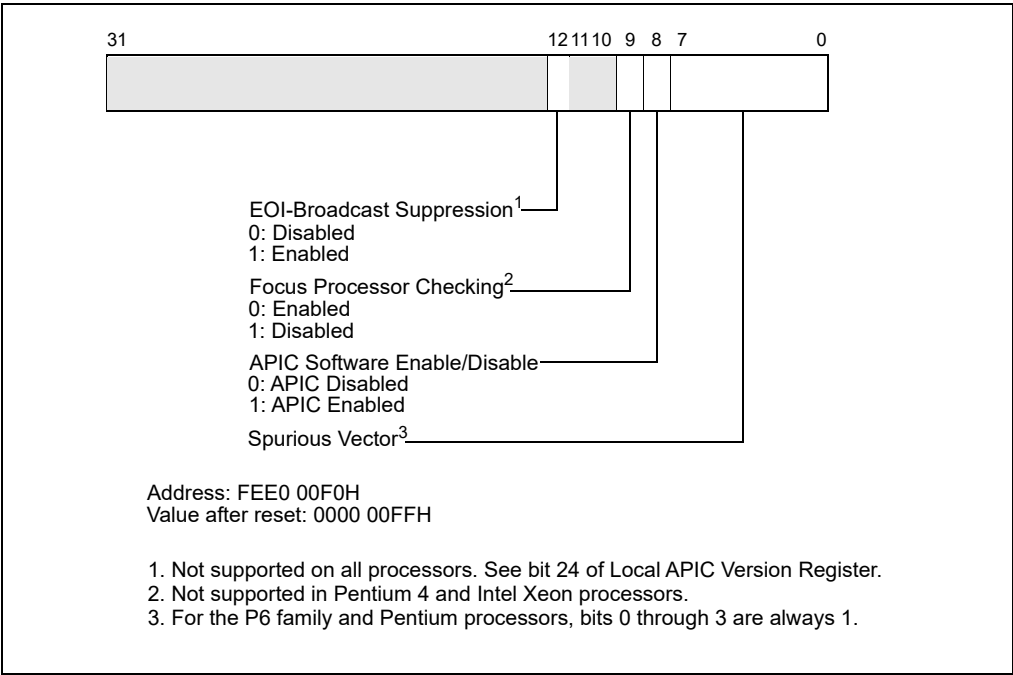


Figure 12-23. Spurious-Interrupt Vector Register (SVR)

12.10 APIC BUS MESSAGE PASSING MECHANISM AND PROTOCOL (P6 FAMILY, PENTIUM PROCESSORS)

The Pentium 4 and Intel Xeon processors pass messages among the local and I/O APICs on the system bus, using the system bus message passing mechanism and protocol.

The P6 family and Pentium processors, pass messages among the local and I/O APICs on the serial APIC bus, as follows. Because only one message can be sent at a time on the APIC bus, the I/O APIC and local APICs employ a “rotating priority” arbitration protocol to gain permission to send a message on the APIC bus. One or more APICs may start sending their messages simultaneously. At the beginning of every message, each APIC presents the type of the message it is sending and its current arbitration priority on the APIC bus. This information is used for arbitration. After each arbitration cycle (within an arbitration round), only the potential winners keep driving the bus.

By the time all arbitration cycles are completed, there will be only one APIC left driving the bus. Once a winner is selected, it is granted exclusive use of the bus, and will continue driving the bus to send its actual message.

After each successfully transmitted message, all APICs increase their arbitration priority by 1. The previous winner (that is, the one that has just successfully transmitted its message) assumes a priority of 0 (lowest). An agent whose arbitration priority was 15 (highest) during arbitration, but did not send a message, adopts the previous winner's arbitration priority, incremented by 1.

Note that the arbitration protocol described above is slightly different if one of the APICs issues a special End-Of-Interrupt (EOI). This high-priority message is granted the bus regardless of its sender's arbitration priority, unless more than one APIC issues an EOI message simultaneously. In the latter case, the APICs sending the EOI messages arbitrate using their arbitration priorities.

If the APICs are set up to use "lowest priority" arbitration (see Section 12.6.2.4, "Lowest Priority Delivery Mode") and multiple APICs are currently executing at the lowest priority (the value in the APR register), the arbitration priorities (unique values in the Arb ID register) are used to break ties. All 8 bits of the APR are used for the lowest priority arbitration.

12.10.1 Bus Message Formats

See Section 12.13, "APIC Bus Message Formats," for a description of bus message formats used to transmit messages on the serial APIC bus.

12.11 MESSAGE SIGNALLED INTERRUPTS

The PCI Local Bus Specification, Rev 2.2 (www.pcisig.com) introduces the concept of message signalled interrupts. As the specification indicates:

"Message signalled interrupts (MSI) is an optional feature that enables PCI devices to request service by writing a system-specified message to a system-specified address (PCI DWORD memory write transaction). The transaction address specifies the message destination while the transaction data specifies the message. System software is expected to initialize the message destination and message during device configuration, allocating one or more non-shared messages to each MSI capable function."

The capabilities mechanism provided by the PCI Local Bus Specification is used to identify and configure MSI capable PCI devices. Among other fields, this structure contains a Message Data Register and a Message Address Register. To request service, the PCI device function writes the contents of the Message Data Register to the address contained in the Message Address Register (and the Message Upper Address register for 64-bit message addresses).

Section 12.11.1 and Section 12.11.2 provide layout details for the Message Address Register and the Message Data Register. The operation issued by the device is a PCI write command to the Message Address Register with the Message Data Register contents. The operation follows semantic rules as defined for PCI write operations and is a DWORD operation.

12.11.1 Message Address Register Format

The format of the Message Address Register (lower 32-bits) is shown in Figure 12-24.

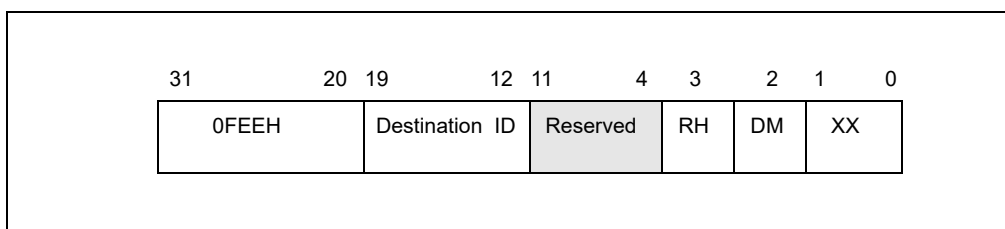


Figure 12-24. Layout of the MSI Message Address Register

Fields in the Message Address Register are as follows:

1. **Bits 31-20** — These bits contain a fixed value for interrupt messages (0FEEH). This value locates interrupts at the 1-MByte area with a base address of 4G – 18M. All accesses to this region are directed as interrupt messages. Care must to be taken to ensure that no other device claims the region as I/O space.
2. **Destination ID** — This field contains an 8-bit destination ID. It identifies the message's target processor(s). The destination ID corresponds to bits 63:56 of the I/O APIC Redirection Table Entry if the IOAPIC is used to dispatch the interrupt to the processor(s).
3. **Redirection hint indication (RH)** — When this bit is set, the message is directed to the processor with the lowest interrupt priority among processors that can receive the interrupt.
 - When RH is 0, the interrupt is directed to the processor listed in the Destination ID field.
 - When RH is 1 and the physical destination mode is used, the Destination ID field must not be set to FFH; it must point to a processor that is present and enabled to receive the interrupt.
 - When RH is 1 and the logical destination mode is active in a system using a flat addressing model, the Destination ID field must be set so that bits set to 1 identify processors that are present and enabled to receive the interrupt.
 - If RH is set to 1 and the logical destination mode is active in a system using cluster addressing model, then Destination ID field must not be set to FFH; the processors identified with this field must be present and enabled to receive the interrupt.
4. **Destination mode (DM)** — This bit indicates whether the Destination ID field should be interpreted as logical or physical APIC ID for delivery of the lowest priority interrupt.
 - If RH is 1 and DM is 0, the Destination ID field is in physical destination mode and only the processor in the system that has the matching APIC ID is considered for delivery of that interrupt (this means no redirection).
 - If RH is 1 and DM is 1, the Destination ID Field is interpreted as in logical destination mode and the redirection is limited to only those processors that are part of the logical group of processors based on the processor's logical APIC ID and the Destination ID field in the message. The logical group of processors consists of those identified by matching the 8-bit Destination ID with the logical destination identified by the Destination Format Register and the Logical Destination Register in each local APIC. The details are similar to those described in Section 12.6.2, "Determining IPI Destination."
 - If RH is 0, then the DM bit is ignored and the message is sent ahead independent of whether the physical or logical destination mode is used.

12.11.2 Message Data Register Format

The layout of the Message Data Register is shown in Figure 12-25.

Reserved fields are not assumed to be any value. Software must preserve their contents on writes. Other fields in the Message Data Register are described below.

1. **Vector** — This 8-bit field contains the interrupt vector associated with the message. Values range from 010H to 0FEH. Software must guarantee that the field is not programmed with vector 00H to 0FH.
2. **Delivery Mode** — This 3-bit field specifies how the interrupt receipt is handled. Delivery Modes operate only in conjunction with specified Trigger Modes. Correct Trigger Modes must be guaranteed by software. Restrictions are indicated below:
 - a. **000B (Fixed Mode)** — Deliver the signal to all the agents listed in the destination. The Trigger Mode for fixed delivery mode can be edge or level.
 - b. **001B (Lowest Priority)** — Deliver the signal to the agent that is executing at the lowest priority of all agents listed in the destination field. The trigger mode can be edge or level.
 - c. **010B (System Management Interrupt or SMI)** — The delivery mode is edge only. For systems that rely on SMI semantics, the vector field is ignored but must be programmed to all zeroes for future compatibility.

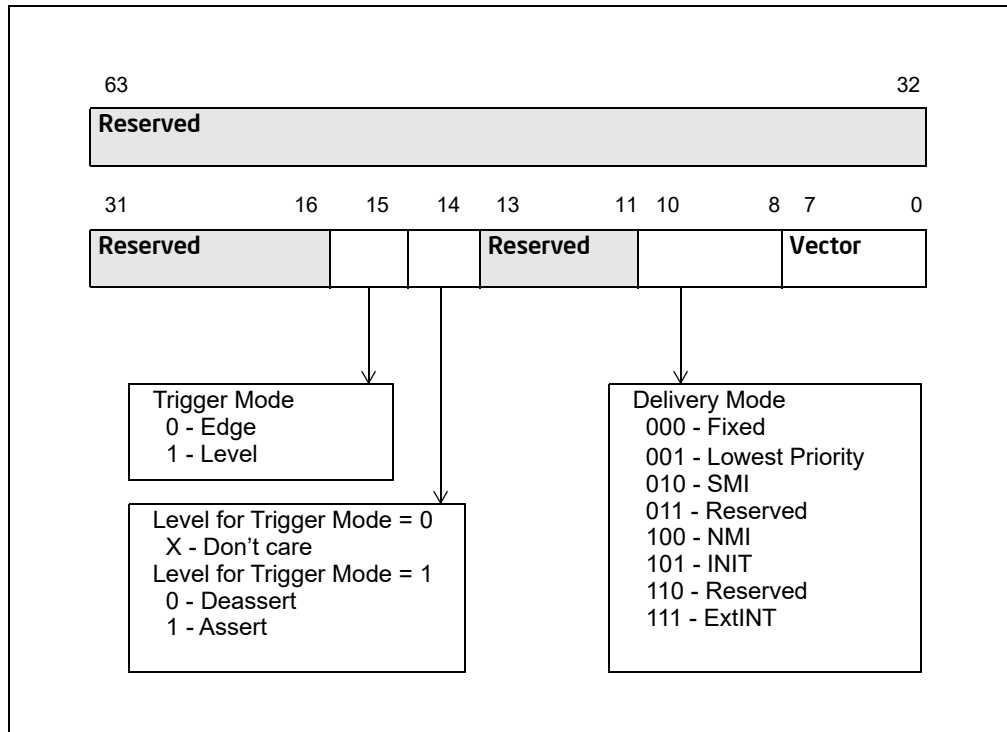


Figure 12-25. Layout of the MSI Message Data Register

- d. **100B (NMI)** — Deliver the signal to all the agents listed in the destination field. The vector information is ignored. NMI is an edge triggered interrupt regardless of the Trigger Mode Setting.
 - e. **101B (INIT)** — Deliver this signal to all the agents listed in the destination field. The vector information is ignored. INIT is an edge triggered interrupt regardless of the Trigger Mode Setting.
 - f. **111B (ExtINT)** — Deliver the signal to the INTR signal of all agents in the destination field (as an interrupt that originated from an 8259A compatible interrupt controller). The vector is supplied by the INTA cycle issued by the activation of the ExtINT. ExtINT is an edge triggered interrupt.
3. **Level** — Edge triggered interrupt messages are always interpreted as assert messages. For edge triggered interrupts this field is not used. For level triggered interrupts, this bit reflects the state of the interrupt input.
 4. **Trigger Mode** — This field indicates the signal type that will trigger a message.
 - a. **0** — Indicates edge sensitive.
 - b. **1** — Indicates level sensitive.

12.12 EXTENDED XAPIC (X2APIC)

The x2APIC architecture extends the xAPIC architecture (described in Section 12.4) in a backward compatible manner and provides forward extendability for future Intel platform innovations. Specifically, the x2APIC architecture does the following.

- Retains all key elements of compatibility to the xAPIC architecture.
 - Delivery modes.
 - Interrupt and processor priorities.
 - Interrupt sources.
 - Interrupt destination types.
- Provides extensions to scale processor addressability for both the logical and physical destination modes.

- Adds new features to enhance performance of interrupt delivery.
- Reduces complexity of logical destination mode interrupt delivery on link based platform architectures.
- Uses MSR programming interface to access APIC registers in x2APIC mode instead of memory-mapped interfaces. Memory-mapped interface is supported when operating in xAPIC mode.

12.12.1 Detecting and Enabling x2APIC Mode

Processor support for x2APIC mode can be detected by checking CPUID.01H. If CPUID.01H:ECX[21] is set, the processor supports the x2APIC capability and can be placed into the x2APIC mode.

System software can place the local APIC in the x2APIC mode by setting the x2APIC mode enable bit (bit 10) in the IA32_APIC_BASE MSR at MSR address 01BH. The layout for the IA32_APIC_BASE MSR is shown in Figure 12-26.

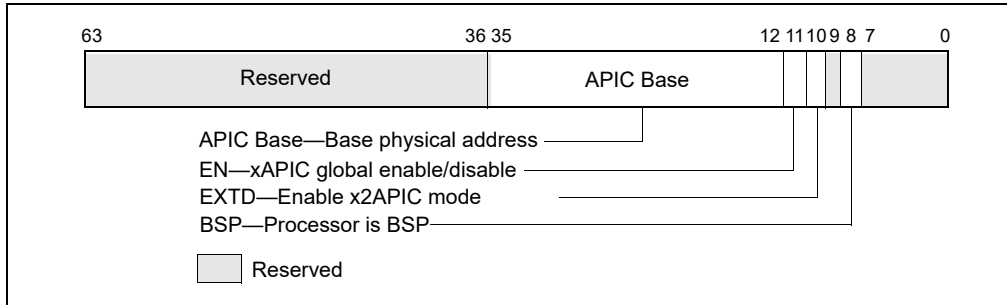


Figure 12-26. IA32_APIC_BASE MSR Supporting x2APIC

Table 12-5, “x2APIC operating mode configurations” describe the possible combinations of the enable bit (EN - bit 11) and the extended mode bit (EXTD - bit 10) in the IA32_APIC_BASE MSR.

Table 12-5. x2APIC Operating Mode Configurations

xAPIC global enable (IA32_APIC_BASE[11])	x2APIC enable (IA32_APIC_BASE[10])	Description
0	0	local APIC is disabled
0	1	Invalid
1	0	local APIC is enabled in xAPIC mode
1	1	local APIC is enabled in x2APIC mode

Once the local APIC has been switched to x2APIC mode (EN = 1, EXTD = 1), switching back to xAPIC mode would require system software to disable the local APIC unit. Specifically, attempting to write a value to the IA32_APIC_BASE MSR that has (EN= 1, EXTD = 0) when the local APIC is enabled and in x2APIC mode causes a general-protection exception. Once bit 10 in IA32_APIC_BASE MSR is set, the only way to leave x2APIC mode using IA32_APIC_BASE would require a WRMSR to set both bit 11 and bit 10 to zero. Section 12.12.5, “x2APIC State Transitions,” provides a detailed state diagram for the state transitions allowed for the local APIC.

12.12.1.1 Instructions to Access APIC Registers

In x2APIC mode, system software uses RDMSR and WRMSR to access the APIC registers. The MSR addresses for accessing the x2APIC registers are architecturally defined and specified in Section 12.12.1.2, “x2APIC Register Address Space.” Executing the RDMSR instruction with the APIC register address specified in ECX returns the content of bits 0 through 31 of the APIC registers in EAX. Bits 32 through 63 are returned in register EDX - these bits are reserved if the APIC register being read is a 32-bit register. Similarly executing the WRMSR instruction with the APIC register address in ECX, writes bits 0 to 31 of register EAX to bits 0 to 31 of the specified APIC register. If the register is a 64-bit register then bits 0 to 31 of register EDX are written to bits 32 to 63 of the APIC register. The

Interrupt Command Register is the only APIC register that is implemented as a 64-bit MSR. The semantics of handling reserved bits are defined in Section 12.12.1.3, “Reserved Bit Checking.”

12.12.1.2 x2APIC Register Address Space

The MSR address range 800H through 8FFH is architecturally reserved and dedicated for accessing APIC registers in x2APIC mode. Table 12-6 lists the APIC registers that are available in x2APIC mode. When appropriate, the table also gives the offset at which each register is available on the page referenced by IA32_APIC_BASE[35:12] in xAPIC mode.

There is a one-to-one mapping between the x2APIC MSRs and the legacy xAPIC register offsets with the following exceptions:

- The Destination Format Register (DFR): The DFR, supported at offset 0E0H in xAPIC mode, is not supported in x2APIC mode. There is no MSR with address 80EH.
- The Interrupt Command Register (ICR): The two 32-bit registers in xAPIC mode (at offsets 300H and 310H) are merged into a single 64-bit MSR in x2APIC mode (with MSR address 830H). There is no MSR with address 831H.
- The SELF IPI register. This register is available only in x2APIC mode at address 83FH. In xAPIC mode, there is no register defined at offset 3F0H.

MSR addresses in the range 800H–8FFH that are not listed in Table 12-6 (including 80EH and 831H) are reserved. Executions of RDMSR and WRMSR that attempt to access such addresses cause general-protection exceptions.

The MSR address space is compressed to allow for future growth. Every 32-bit register on a 128-bit boundary in the legacy MMIO space is mapped to a single MSR in the local x2APIC MSR address space. The upper 32-bits of all x2APIC MSRs (except for the ICR) are reserved.

Table 12-6. Local APIC Register Address Map Supported by x2APIC

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
802H	020H	Local APIC ID register	Read-only ¹	See Section 12.12.5.1 for initial values.
803H	030H	Local APIC Version register	Read-only	Same version used in xAPIC mode and x2APIC mode.
808H	080H	Task Priority Register (TPR)	Read/write	Bits 31:8 are reserved. ²
80AH	0A0H	Processor Priority Register (PPR)	Read-only	
80BH	0B0H	EOI register	Write-only ³	WRMSR of a non-zero value causes #GP(0).
80DH	0D0H	Logical Destination Register (LDR)	Read-only	Read/write in xAPIC mode.
80FH	0F0H	Spurious Interrupt Vector Register (SVR)	Read/write	See Section 12.9 for reserved bits.
810H	100H	In-Service Register (ISR); bits 31:0	Read-only	
811H	110H	ISR bits 63:32	Read-only	
812H	120H	ISR bits 95:64	Read-only	
813H	130H	ISR bits 127:96	Read-only	
814H	140H	ISR bits 159:128	Read-only	
815H	150H	ISR bits 191:160	Read-only	
816H	160H	ISR bits 223:192	Read-only	

Table 12-6. Local APIC Register Address Map Supported by x2APIC (Contd.)

MSR Address (x2APIC mode)	MMIO Offset (xAPIC mode)	Register Name	MSR R/W Semantics	Comments
817H	170H	ISR bits 255:224	Read-only	
818H	180H	Trigger Mode Register (TMR); bits 31:0	Read-only	
819H	190H	TMR bits 63:32	Read-only	
81AH	1A0H	TMR bits 95:64	Read-only	
81BH	1B0H	TMR bits 127:96	Read-only	
81CH	1C0H	TMR bits 159:128	Read-only	
81DH	1D0H	TMR bits 191:160	Read-only	
81EH	1E0H	TMR bits 223:192	Read-only	
81FH	1F0H	TMR bits 255:224	Read-only	
820H	200H	Interrupt Request Register (IRR); bits 31:0	Read-only	
821H	210H	IRR bits 63:32	Read-only	
822H	220H	IRR bits 95:64	Read-only	
823H	230H	IRR bits 127:96	Read-only	
824H	240H	IRR bits 159:128	Read-only	
825H	250H	IRR bits 191:160	Read-only	
826H	260H	IRR bits 223:192	Read-only	
827H	270H	IRR bits 255:224	Read-only	
828H	280H	Error Status Register (ESR)	Read/write	WRMSR of a non-zero value causes #GP(0). See Section 12.5.3.
82FH	2F0H	LVT CMCi register	Read/write	See Figure 12-8 for reserved bits.
830H ⁴	300H and 310H	Interrupt Command Register (ICR)	Read/write	See Figure 12-28 for reserved bits
832H	320H	LVT Timer register	Read/write	See Figure 12-8 for reserved bits.
833H	330H	LVT Thermal Sensor register	Read/write	See Figure 12-8 for reserved bits.
834H	340H	LVT Performance Monitoring register	Read/write	See Figure 12-8 for reserved bits.
835H	350H	LVT LINT0 register	Read/write	See Figure 12-8 for reserved bits.
836H	360H	LVT LINT1 register	Read/write	See Figure 12-8 for reserved bits.
837H	370H	LVT Error register	Read/write	See Figure 12-8 for reserved bits.
838H	380H	Initial Count register (for Timer)	Read/write	
839H	390H	Current Count register (for Timer)	Read-only	
83EH	3E0H	Divide Configuration Register (DCR; for Timer)	Read/write	See Figure 12-10 for reserved bits.
83FH	Not available	SELF IPI ⁵	Write-only	Available only in x2APIC mode.

NOTES:

1. WRMSR causes #GP(0) for read-only registers.

2. WRMSR causes #GP(0) for attempts to set a reserved bit to 1 in a read/write register (including bits 63:32 of each register).
3. RDMSR causes #GP(0) for write-only registers.
4. MSR 831H is reserved; read/write operations cause general-protection exceptions. The contents of the APIC register at MMIO offset 310H are accessible in x2APIC mode through the MSR at address 830H.
5. SELF IPI register is supported only in x2APIC mode.

12.12.1.3 Reserved Bit Checking

Section 12.12.1.2 and Table 12-6 specifies the reserved bit definitions for the APIC registers in x2APIC mode. Non-zero writes (by WRMSR instruction) to reserved bits to these registers will raise a general protection fault exception while reads return zeros (RsvdZ semantics).

In x2APIC mode, the local APIC ID register is increased to 32 bits wide. This enables $2^{32}-1$ processors to be addressable in physical destination mode. This 32-bit value is referred to as “x2APIC ID”. A processor implementation may choose to support less than 32 bits in its hardware. System software should be agnostic to the actual number of bits that are implemented. All non-implemented bits will return zeros on reads by software.

The APIC ID value of FFFF_FFFFH and the highest value corresponding to the implemented bit-width of the local APIC ID register in the system are reserved and cannot be assigned to any logical processor.

In x2APIC mode, the local APIC ID register is a read-only register to system software and will be initialized by hardware. It is accessed via the RDMSR instruction reading the MSR at address 0802H.

Each logical processor in the system (including clusters with a communication fabric) must be configured with a unique x2APIC ID to avoid collisions of x2APIC IDs. On DP and high-end MP processors targeted to specific market segments and depending on the system configuration, it is possible that logical processors in different and “unconnected” clusters power up initialized with overlapping x2APIC IDs. In these configurations, a model-specific means may be provided in those product segments to enable BIOS and/or platform firmware to re-configure the x2APIC IDs in some clusters to provide for unique and non-overlapping system wide IDs before configuring the disconnected components into a single system.

12.12.2 x2APIC Register Availability

The local APIC registers can be accessed via the MSR interface only when the local APIC has been switched to the x2APIC mode as described in Section 12.12.1. Accessing any APIC register in the MSR address range 0800H through 08FFH via RDMSR or WRMSR when the local APIC is not in x2APIC mode causes a general-protection exception. In x2APIC mode, the memory mapped interface is not available and any access to the MMIO interface will behave similar to that of a legacy xAPIC in globally disabled state. Table 12-7 provides the interactions between the legacy & extended modes and the legacy and register interfaces.

Table 12-7. MSR/MMIO Interface of a Local x2APIC in Different Modes of Operation

	MMIO Interface	MSR Interface
xAPIC mode	Available	General-protection exception
x2APIC mode	Behavior identical to xAPIC in globally disabled state	Available

12.12.3 MSR Access in x2APIC Mode

To allow for efficient access to the APIC registers in x2APIC mode, the serializing semantics of WRMSR are relaxed when writing to the APIC registers. Thus, system software should not use “WRMSR to APIC registers in x2APIC mode” as a serializing instruction. Read and write accesses to the APIC registers will occur in program order. A WRMSR to an APIC register may complete before all preceding stores are globally visible; software can prevent this by inserting a serializing instruction or the sequence MFENCE;LFENCE before the WRMSR.

The RDMSR instruction is not serializing and this behavior is unchanged when reading APIC registers in x2APIC mode. System software accessing the APIC registers using the RDMSR instruction should not expect a serializing behavior. (Note: The MMIO-based xAPIC interface is mapped by system software as an un-cached region. Consequently, read/writes to the xAPIC-MMIO interface have serializing semantics in the xAPIC mode.)

12.12.4 VM-Exit Controls for MSRs and x2APIC Registers

The VMX architecture allows a VMM to specify lists of MSRs to be loaded or stored on VMX transitions using the VMX-transition MSR areas (see VM-exit MSR-store address field, VM-exit MSR-load address field, and VM-entry MSR-load address field in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C).

The X2APIC MSRs cannot to be loaded and stored on VMX transitions. A VMX transition fails if the VMM has specified that the transition should access any MSRs in the address range from 0000_0800H to 0000_08FFH (the range used for accessing the X2APIC registers). Specifically, processing of an 128-bit entry in any of the VMX-transition MSR areas fails if bits 31:0 of that entry (represented as ENTRY_LOW_DW) satisfies the expression: "ENTRY_LOW_DW & FFFFF800H = 00000800H". Such a failure causes an associated VM entry to fail (by reloading host state) and causes an associated VM exit to lead to VMX abort.

12.12.5 x2APIC State Transitions

This section provides a detailed description of the x2APIC states of a local x2APIC unit, transitions between these states as well as interactions of these states with INIT and reset.

12.12.5.1 x2APIC States

The valid states for a local x2APIC unit are listed in Table 12-5.

- APIC disabled: IA32_APIC_BASE[EN]=0 and IA32_APIC_BASE[EXTD]=0.
- xAPIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=0.
- x2APIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=1.
- Invalid: IA32_APIC_BASE[EN]=0 and IA32_APIC_BASE[EXTD]=1.

The state corresponding to EXTD=1 and EN=0 is not valid and it is not possible to get into this state. An execution of WRMSR to the IA32_APIC_BASE_MSR that attempts a transition from a valid state to this invalid state causes a general-protection exception. Figure 12-27 shows the comprehensive state transition diagram for a local x2APIC unit.

On coming out of reset, the local APIC unit is enabled and is in the xAPIC mode: IA32_APIC_BASE[EN]=1 and IA32_APIC_BASE[EXTD]=0. The APIC registers are initialized as follows.

- The local APIC ID is initialized by hardware with a 32 bit ID (x2APIC ID). The lowest 8 bits of the x2APIC ID are the legacy local xAPIC ID, and are stored in the upper 8 bits of the APIC register for access in xAPIC mode.
- The following APIC registers are reset to all zeros for those fields that are defined in the xAPIC mode.
 - IRR, ISR, TMR, ICR, LDR, TPR, Divide Configuration Register (See Section 12.4 through Section 12.6 for details of individual APIC registers).
 - Timer initial count and timer current count registers.
- The LVT registers are reset to 0s except for the mask bits; these are set to 1s.
- The local APIC version register is not affected.
- The Spurious Interrupt Vector Register is initialized to 000000FFH.
- The DFR (available only in xAPIC mode) is reset to all 1s.
- SELF IPI register is reset to zero.

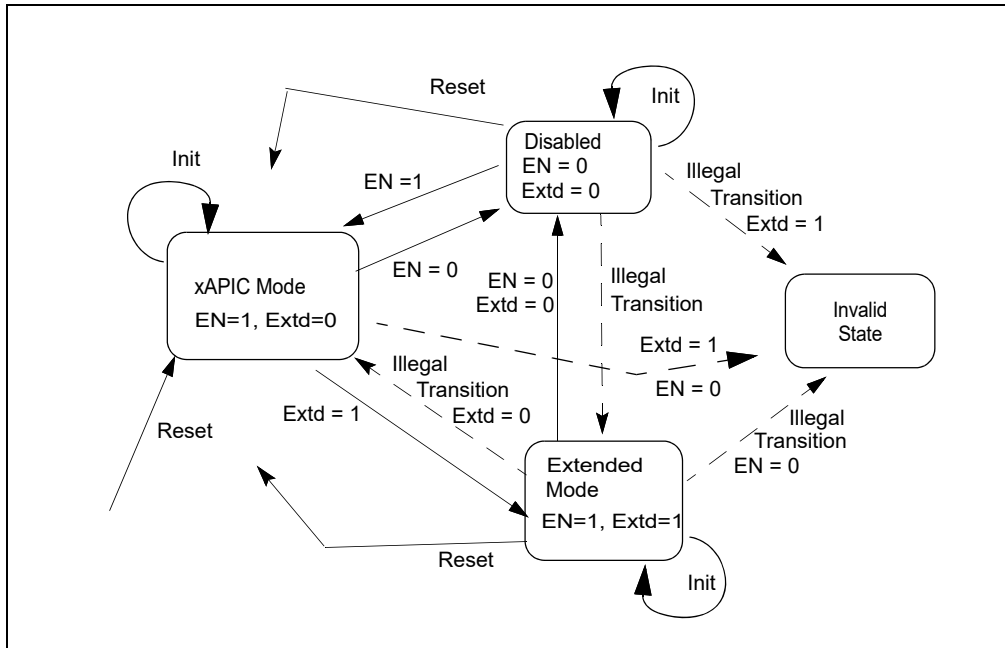


Figure 12-27. Local x2APIC State Transitions with IA32_APIC_BASE, INIT, and Reset

x2APIC After Reset

The valid transitions from the xAPIC mode state are:

- to the x2APIC mode by setting EXT to 1 (resulting EN=1, EXTD= 1). The physical x2APIC ID (see Figure 12-6) is preserved across this transition and the logical x2APIC ID (see Figure 12-29) is initialized by hardware during this transition as documented in Section 12.12.10.2. The state of the extended fields in other APIC registers, which was not initialized at reset, is not architecturally defined across this transition and system software should explicitly initialize those programmable APIC registers.
- to the disabled state by setting EN to 0 (resulting EN=0, EXTD= 0).

The result of an INIT in the xAPIC state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is preserved (the 8-bit xAPIC ID is in the upper 8 bits of the APIC ID register). All the other APIC registers are initialized as a result of INIT.

A reset in this state places the APIC in the state with EN= 1, EXTD= 0. The state of the local APIC ID register is initialized as described in Section 12.12.5.1. All the other APIC registers are initialized described in Section 12.12.5.1.

x2APIC Transitions From x2APIC Mode

From the x2APIC mode, the only valid x2APIC transition using IA32_APIC_BASE is to the state where the x2APIC is disabled by setting EN to 0 and EXTD to 0. The x2APIC ID (32 bits) and the legacy local xAPIC ID (8 bits) are preserved across this transition. A transition from the x2APIC mode to xAPIC mode is not valid, and the corresponding WRMSR to the IA32_APIC_BASE MSR causes a general-protection exception.

A reset in this state places the x2APIC in xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 12.12.5.1.

An INIT in this state keeps the x2APIC in the x2APIC mode. The state of the local APIC ID register is preserved (all 32 bits). However, all the other APIC registers are initialized as a result of the INIT transition.

x2APIC Transitions From Disabled Mode

From the disabled state, the only valid x2APIC transition using IA32_APIC_BASE is to the xAPIC mode (EN=1, EXTD = 0). Thus the only means to transition from x2APIC mode to xAPIC mode is a two-step process:

- first transition from x2APIC mode to local APIC disabled mode (EN= 0, EXTD = 0),
- followed by another transition from disabled mode to xAPIC mode (EN= 1, EXTD= 0).

Consequently, all the APIC register states in the x2APIC, except for the x2APIC ID (32 bits), are not preserved across mode transitions.

A reset in the disabled state places the x2APIC in the xAPIC mode. All APIC registers (including the local APIC ID register) are initialized as described in Section 12.12.5.1.

An INIT in the disabled state keeps the x2APIC in the disabled state.

State Changes From xAPIC Mode to x2APIC Mode

After APIC register states have been initialized by software in xAPIC mode, a transition from xAPIC mode to x2APIC mode does not affect most of the APIC register states, except the following:

- The Logical Destination Register is not preserved.
- Any APIC ID value written to the memory-mapped local APIC ID register is not preserved.
- The high half of the Interrupt Command Register is not preserved.

12.12.6 Routing of Device Interrupts in x2APIC Mode

The x2APIC architecture is intended to work with all existing IOxAPIC units as well as all PCI and PCI Express (PCIe) devices that support the capability for message-signaled interrupts (MSI). Support for x2APIC modifies only the following:

- the local APIC units;
- the interconnects joining IOxAPIC units to the local APIC units; and
- the interconnects joining MSI-capable PCI and PCIe devices to the local APIC units.

No modifications are required to MSI-capable PCI and PCIe devices. Similarly, no modifications are required to IOxAPIC units. This is made possible through use of the interrupt-remapping architecture specified in the Intel® Virtualization Technology for Directed I/O Specification, Revision 1.3 and/or later versions, for the routing of interrupts from MSI-capable devices to local APIC units operating in x2APIC mode.

12.12.7 Initialization by System Software

Routing of device interrupts to local APIC units operating in x2APIC mode requires use of the interrupt-remapping architecture specified in the Intel® Virtualization Technology for Directed I/O Specification (Revision 1.3 and/or later versions). Because of this, BIOS must enumerate support for and software must enable this interrupt remapping with Extended Interrupt Mode Enabled before it enabling x2APIC mode in the local APIC units.

The ACPI interfaces for the x2APIC are described in Section 5.2, “ACPI System Description Tables,” of the Advanced Configuration and Power Interface Specification, Revision 4.0a (<http://www.acpi.info/spec.htm>). The default behavior for BIOS is to pass the control to the operating system with the local x2APICs in xAPIC mode if all APIC IDs reported by CPUID.0BH:EDX are less than 255, and in x2APIC mode if there are any logical processor reporting an APIC ID of 255 or greater.

12.12.8 CPUID Extensions And Topology Enumeration

For Intel 64 and IA-32 processors that support x2APIC, a value of 1 reported by CPUID.01H:ECX[21] indicates that the processor supports x2APIC and the extended topology enumeration leaf (CPUID.0BH).

The extended topology enumeration leaf can be accessed by executing CPUID with EAX = 0BH. Processors that do not support x2APIC may support CPUID.0BH. Software can detect the availability of the extended topology enumeration leaf (0BH) by performing two steps:

- Check maximum input value for basic CPUID information by checking CPUID.00H. If CPUID.00H:EAX is greater than or equal to 11 (0BH), then proceed to next step
- Check CPUID.0BH.00H:EBX is non-zero.

If both of the above conditions are true, extended topology enumeration leaf is available. If available, the extended topology enumeration leaf is the preferred mechanism for enumerating topology. The presence of CPUID.0BH in a processor does not guarantee support for x2APIC. If CPUID.0BH.00H:EBX returns zero and maximum input value for basic CPUID information is greater than 0BH, then CPUID.0BH leaf is not supported on that processor.

The extended topology enumeration leaf is intended to assist software with enumerating processor topology on systems that requires 32-bit x2APIC IDs to address individual logical processors. Details of CPUID.0BH can be found in the reference pages of CPUID in Chapter 3 of Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

Processor topology enumeration algorithm for processors supporting the extended topology enumeration leaf of CPUID and processors that do not support CPUID.0BH are treated in Section 10.9.4, "Algorithm for Three-Domain Mappings of APIC_ID."

12.12.8.1 Consistency of APIC IDs and CPUID

The consistency of physical x2APIC ID in MSR 802H in x2APIC mode and the 32-bit value returned in CPUID.0BH:EDX is facilitated by processor hardware.

CPUID.0BH:EDX will report the full 32 bit ID, in xAPIC and x2APIC mode. This allows BIOS to determine if a system has processors with IDs exceeding the 8-bit initial APIC ID limit (CPUID.01H:EBX[31:24]). Initial APIC ID (CPUID.01H:EBX[31:24]) is always equal to CPUID.0BH:EDX[7:0].

If the values of CPUID.0BH:EDX reported by all logical processors in a system are less than 255, BIOS can transfer control to OS in xAPIC mode.

If the values of CPUID.0BH:EDX reported by some logical processors in a system are greater than or equal to 255, BIOS must support two options to hand off to OS.

- If BIOS enables logical processors with x2APIC IDs greater than 255, then it should enable x2APIC in the Boot Strap Processor (BSP) and all Application Processors (AP) before passing control to the OS. Applications requiring processor topology information must use OS provided services based on x2APIC IDs or CPUID.0BH leaf.
- If a BIOS transfers control to OS in xAPIC mode, then the BIOS must ensure that only logical processors with CPUID.0BH:EDX value less than 255 are enabled. BIOS initialization on all logical processors with CPUID.0BH:EDX values greater than or equal to 255 must (a) disable APIC and execute CLI in each logical processor, and (b) leave these logical processor in the lowest power state so that these processors do not respond to INIT IPI during OS boot. The BSP and all the enabled logical processor operate in xAPIC mode after BIOS passed control to OS. Application requiring processor topology information can use OS provided legacy services based on 8-bit initial APIC IDs or legacy topology information from CPUID.01H and CPUID 04H leaves. Even if the BIOS passes control in xAPIC mode, an OS can switch the processors to x2APIC mode later. BIOS SMM handler should always read the APIC_BASE_MSR, determine the APIC mode and use the corresponding access method.

12.12.9 ICR Operation in x2APIC Mode

In x2APIC mode, the layout of the Interrupt Command Register is shown in Figure 12-28. The lower 32 bits of ICR in x2APIC mode is identical to the lower half of the ICR in xAPIC mode, except the Delivery Status bit is removed since it is not needed in x2APIC mode. The destination ID field is expanded to 32 bits in x2APIC mode.

To send an IPI using the ICR, software must set up the ICR to indicate the type of IPI message to be sent and the destination processor or processors. Self IPIs can also be sent using the SELF IPI register (see Section 12.12.11).

A single MSR write to the Interrupt Command Register is required for dispatching an interrupt in x2APIC mode. With the removal of the Delivery Status bit, system software no longer has a reason to read the ICR. It remains

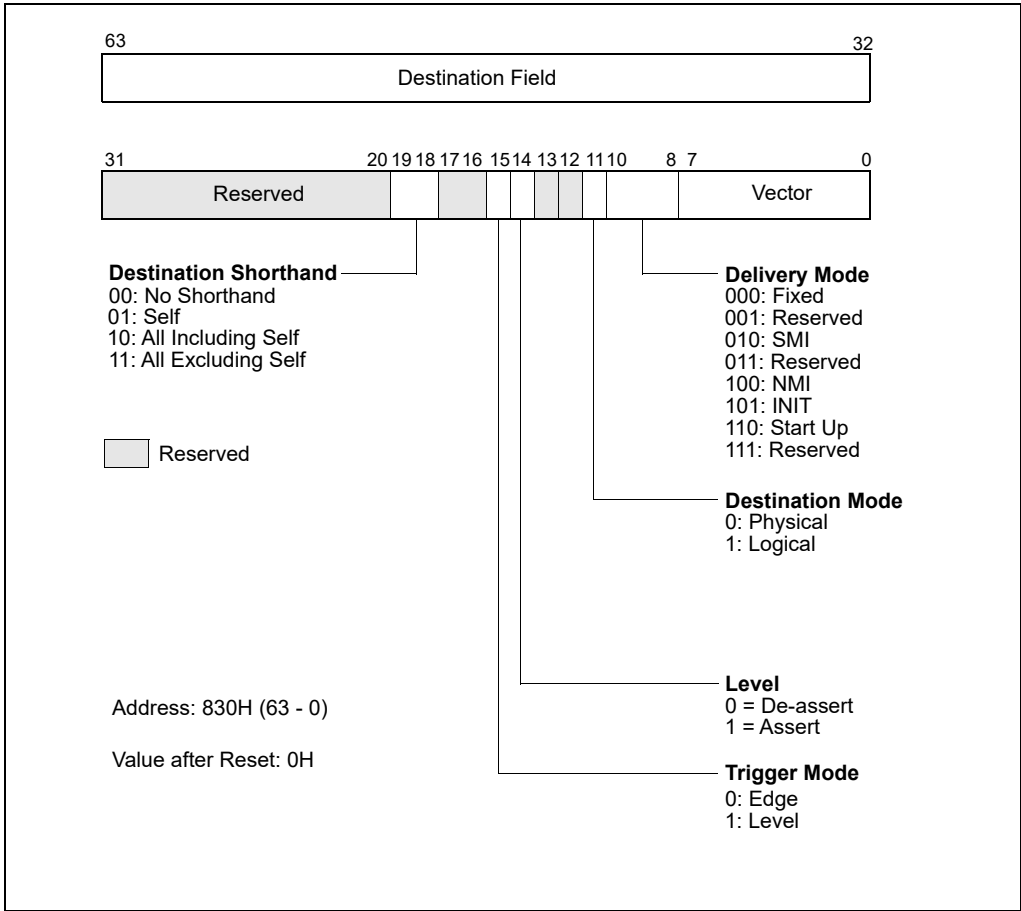


Figure 12-28. Interrupt Command Register (ICR) in x2APIC Mode

readable only to aid in debugging; however, software should not assume the value returned by reading the ICR is the last written value.

A destination ID value of FFFF_FFFFH is used for broadcast of interrupts in both logical destination and physical destination modes.

12.12.10 Determining IPI Destination in x2APIC Mode

12.12.10.1 Logical Destination Mode in x2APIC Mode

In x2APIC mode, the Logical Destination Register (LDR) is increased to 32 bits wide. It is a read-only register to system software. This 32-bit value is referred to as "logical x2APIC ID". System software accesses this register via the RDMSR instruction reading the MSR at address 80DH. Figure 12-29 provides the layout of the Logical Destination Register in x2APIC mode.

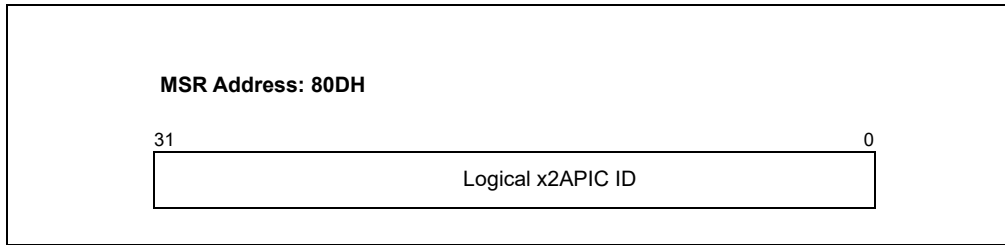


Figure 12-29. Logical Destination Register in x2APIC Mode

In the xAPIC mode, the Destination Format Register (DFR) through the MMIO interface determines the choice of a flat logical mode or a clustered logical mode. Flat logical mode is not supported in the x2APIC mode. Hence the Destination Format Register (DFR) is eliminated in x2APIC mode.

The 32-bit logical x2APIC ID field of LDR is partitioned into two sub-fields:

- Cluster ID (LDR[31:16]): is the address of the destination cluster
- Logical ID (LDR[15:0]): defines a logical ID of the individual local x2APIC within the cluster specified by LDR[31:16].

This layout enables $2^{16}-1$ clusters each with up to 16 unique logical IDs - effectively providing an addressability of $((2^{20}) - 16)$ processors in logical destination mode.

It is likely that processor implementations may choose to support less than 16 bits of the cluster ID or less than 16-bits of the Logical ID in the Logical Destination Register. However system software should be agnostic to the number of bits implemented in the cluster ID and logical ID sub-fields. The x2APIC hardware initialization will ensure that the appropriately initialized logical x2APIC IDs are available to system software and reads of non-implemented bits return zero. This is a read-only register that software must read to determine the logical x2APIC ID of the processor. Specifically, software can apply a 16-bit mask to the lowest 16 bits of the logical x2APIC ID to identify the logical address of a processor within a cluster without needing to know the number of implemented bits in cluster ID and Logical ID sub-fields. Similarly, software can create a message destination address for cluster model, by bit-Oring the Logical X2APIC ID (31:0) of processors that have matching Cluster ID(31:16).

To enable cluster ID assignment in a fashion that matches the system topology characteristics and to enable efficient routing of logical mode lowest priority device interrupts in link based platform interconnects, the LDR are initialized by hardware based on the value of x2APIC ID upon x2APIC state transitions. Details of this initialization are provided in Section 12.12.10.2.

12.12.10.2 Deriving Logical x2APIC ID from the Local x2APIC ID

In x2APIC mode, the 32-bit logical x2APIC ID, which can be read from LDR, is derived from the 32-bit local x2APIC ID. Specifically, the 16-bit logical ID sub-field is derived by shifting 1 by the lowest 4 bits of the x2APIC ID, i.e., $\text{Logical ID} = 1 \ll \text{x2APIC ID}[3:0]$. The remaining bits of the x2APIC ID then form the cluster ID portion of the logical x2APIC ID:

$$\text{Logical x2APIC ID} = [(\text{x2APIC ID}[19:4] \ll 16) | (1 \ll \text{x2APIC ID}[3:0])]$$

The use of the lowest 4 bits in the x2APIC ID implies that at least 16 APIC IDs are reserved for logical processors within a socket in multi-socket configurations. If more than 16 APIC IDs are reserved for logical processors in a socket/package then multiple cluster IDs can exist within the package.

The LDR initialization occurs whenever the x2APIC mode is enabled (see Section 12.12.5).

12.12.11 SELF IPI Register

SELF IPIs are used extensively by some system software. The x2APIC architecture introduces a new register interface. This new register is dedicated to the purpose of sending self-IPIs with the intent of enabling a highly optimized path for sending self-IPIs.

Figure 12-30 provides the layout of the SELF IPI register. System software only specifies the vector associated with the interrupt to be sent. The semantics of sending a self-IPI via the SELF IPI register are identical to sending a self targeted edge triggered fixed interrupt with the specified vector. Specifically the semantics are identical to the following settings for an inter-processor interrupt sent via the ICR - Destination Shorthand (ICR[19:18] = 01 (Self)), Trigger Mode (ICR[15] = 0 (Edge)), Delivery Mode (ICR[10:8] = 000 (Fixed)), Vector (ICR[7:0] = Vector).

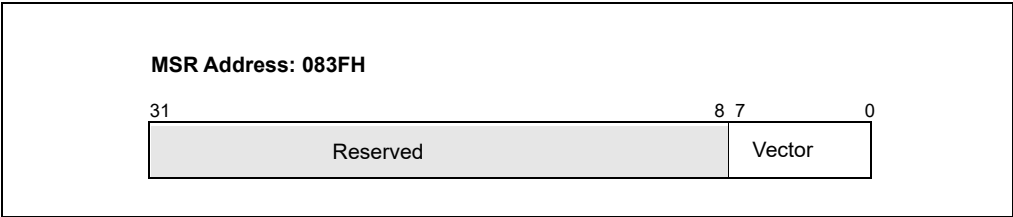


Figure 12-30. SELF IPI register

The SELF IPI register is a write-only register. A RDMSR instruction with address of the SELF IPI register causes a general-protection exception.

The handling and prioritization of a self-IPI sent via the SELF IPI register is architecturally identical to that for an IPI sent via the ICR from a legacy xAPIC unit. Specifically the state of the interrupt would be tracked via the Interrupt Request Register (IRR) and In Service Register (ISR) and Trigger Mode Register (TMR) as if it were received from the system bus. Also sending the IPI via the Self Interrupt Register ensures that interrupt is delivered to the processor core. Specifically completion of the WRMSR instruction to the SELF IPI register implies that the interrupt has been logged into the IRR. As expected for edge triggered interrupts, depending on the processor priority and readiness to accept interrupts, it is possible that interrupts sent via the SELF IPI register or via the ICR with identical vectors can be combined.

12.13 APIC BUS MESSAGE FORMATS

This section describes the message formats used when transmitting messages on the serial APIC bus. The information described here pertains only to the Pentium and P6 family processors.

12.13.1 Bus Message Formats

The local and I/O APICs transmit three types of messages on the serial APIC bus: EOI message, short message, and non-focused lowest priority message. The purpose of each type of message and its format are described below.

12.13.2 EOI Message

Local APICs send 14-cycle EOI messages to the I/O APIC to indicate that a level triggered interrupt has been accepted by the processor. This interrupt, in turn, is a result of software writing into the EOI register of the local APIC. Table 12-8 shows the cycles in an EOI message.

Table 12-8. EOI Message (14 Cycles)

Cycle	Bit1	Bit0	
1	1	1	11 = EOI
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	

Table 12-8. EOI Message (14 Cycles) (Contd.)

Cycle	Bit1	Bit0	
4	ArbID1	0	
5	ArbID0	0	
6	V7	V6	Interrupt vector V7 - V0
7	V5	V4	
8	V3	V2	
9	V1	V0	
10	C	C	Checksum for cycles 6 - 9
11	0	0	
12	A	A	Status Cycle 0
13	A1	A1	Status Cycle 1
14	0	0	Idle

The checksum is computed for cycles 6 through 9. It is a cumulative sum of the 2-bit (Bit1:Bit0) logical data values. The carry out of all but the last addition is added to the sum. If any APIC computes a different checksum than the one appearing on the bus in cycle 10, it signals an error, driving 11 on the APIC bus during cycle 12. In this case, the APICs disregard the message. The sending APIC will receive an appropriate error indication (see Section 12.5.3, "Error Handling") and resend the message. The status cycles are defined in Table 12-11.

12.13.2.1 Short Message

Short messages (21-cycles) are used for sending fixed, NMI, SMI, INIT, start-up, ExtINT, and lowest-priority-with-focus interrupts. Table 12-9 shows the cycles in a short message.

Table 12-9. Short Message (21 Cycles)

Cycle	Bit1	Bit0	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination Mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	Checksum for cycles 6-16

Table 12-9. Short Message (21 Cycles) (Contd.)

Cycle	Bit1	Bit0	
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	0	0	Idle

If the physical delivery mode is being used, then cycles 15 and 16 represent the APIC ID and cycles 13 and 14 are considered don't care by the receiver. If the logical delivery mode is being used, then cycles 13 through 16 are the 8-bit logical destination field.

For shorthands of "all-incl-self" and "all-excl-self," the physical delivery mode and an arbitration priority of 15 (D0:D3 = 1111) are used. The agent sending the message is the only one required to distinguish between the two cases. It does so using internal information.

When using lowest priority delivery with an existing focus processor, the focus processor identifies itself by driving 10 during cycle 19 and accepts the interrupt. This is an indication to other APICs to terminate arbitration. If the focus processor has not been found, the short message is extended on-the-fly to the non-focused lowest-priority message. Note that except for the EOI message, messages generating a checksum or an acceptance error (see Section 12.5.3, "Error Handling") terminate after cycle 21.

12.13.2.2 Non-focused Lowest Priority Message

These 34-cycle messages (see Table 12-10) are used in the lowest priority delivery mode when a focus processor is not present. Cycles 1 through 20 are same as for the short message. If during the status cycle (cycle 19) the state of the (A:A) flags is 10B, a focus processor has been identified, and the short message format is used (see Table 12-9). If the (A:A) flags are set to 00B, lowest priority arbitration is started and the 34-cycles of the non-focused lowest priority message are completed. For other combinations of status flags, refer to Section 12.13.2.3, "APIC Bus Status Cycles."

Table 12-10. Non-Focused Lowest Priority Message (34 Cycles)

Cycle	Bit0	Bit1	
1	0	1	0 1 = normal
2	ArbID3	0	Arbitration ID bits 3 through 0
3	ArbID2	0	
4	ArbID1	0	
5	ArbID0	0	
6	DM	M2	DM = Destination mode
7	M1	M0	M2-M0 = Delivery mode
8	L	TM	L = Level, TM = Trigger Mode
9	V7	V6	V7-V0 = Interrupt Vector
10	V5	V4	
11	V3	V2	
12	V1	V0	
13	D7	D6	D7-D0 = Destination
14	D5	D4	
15	D3	D2	
16	D1	D0	
17	C	C	Checksum for cycles 6-16

Table 12-10. Non-Focused Lowest Priority Message (34 Cycles) (Contd.)

Cycle	Bit0	Bit1	
18	0	0	
19	A	A	Status cycle 0
20	A1	A1	Status cycle 1
21	P7	0	P7 - P0 = Inverted Processor Priority
22	P6	0	
23	P5	0	
24	P4	0	
25	P3	0	
26	P2	0	
27	P1	0	
28	P0	0	
29	ArbID3	0	Arbitration ID 3 -0
30	ArbID2	0	
31	ArbID1	0	
32	ArbID0	0	
33	A2	A2	Status Cycle
34	0	0	Idle

Cycles 21 through 28 are used to arbitrate for the lowest priority processor. The processors participating in the arbitration drive their inverted processor priority on the bus. Only the local APICs having free interrupt slots participate in the lowest priority arbitration. If no such APIC exists, the message will be rejected, requiring it to be tried at a later time.

Cycles 29 through 32 are also used for arbitration in case two or more processors have the same lowest priority. In the lowest priority delivery mode, all combinations of errors in cycle 33 (A2 A2) will set the "accept error" bit in the error status register (see Figure 12-9). Arbitration priority update is performed in cycle 20, and is not affected by errors detected in cycle 33. Only the local APIC that wins in the lowest priority arbitration, drives cycle 33. An error in cycle 33 will force the sender to resend the message.

12.13.2.3 APIC Bus Status Cycles

Certain cycles within an APIC bus message are status cycles. During these cycles the status flags (A:A) and (A1:A1) are examined. Table 12-11 shows how these status flags are interpreted, depending on the current delivery mode and existence of a focus processor.

Table 12-11. APIC Bus Status Cycles Interpretation

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
EOI	00: CS_OK	10: Accept	XX:	Yes, 13	14 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 13	14 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	14 Cycle	Yes
	11: CS_Error	XX:	XX:	No	14 Cycle	Yes
	10: Error	XX:	XX:	No	14 Cycle	Yes
	01: Error	XX:	XX:	No	14 Cycle	Yes
Fixed	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
NMI, SMI, INIT, ExtINT, Start-Up	00: CS_OK	10: Accept	XX:	Yes, 20	21 Cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 Cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 Cycle	Yes
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	10: Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes
Lowest	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	Yes, 20	34 Cycle	No
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	10: End and Retry	XX:	Yes, 20	34 Cycle	Yes
	00: CS_OK, NoFocus	0X: Error	XX:	No	34 Cycle	Yes
	10: CS_OK, Focus	XX:	XX:	Yes, 20	34 Cycle	No
	11: CS_Error	XX:	XX:	No	21 Cycle	Yes
	01: Error	XX:	XX:	No	21 Cycle	Yes

14. Updates to Chapter 13, Volume 3A

Change bars and violet text show changes to Chapter 13 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A: System Programming Guide, Part 1*.

Changes to this chapter:

- In Section 13.5.1, "Cache Control Registers and Bits", updated Figure 13-3.
- In Section 13.11.1, "MTRR Feature Identification", added SEAMRR flag and updated Figure 13-5.
- In Section 13.11.2.3, "Variable Range MTRRs" updated PhysBase and PhysMask fields and updated Figure 13-7.
- In Section 13.11.3.1, "Base and Mask Calculations for Greater-Than 36-bit Physical Address Support", added footnote reference.

This chapter describes the memory cache and cache control mechanisms, the TLBs, and the store buffer in Intel 64 and IA-32 processors. It also describes the memory type range registers (MTRRs) introduced in the P6 family processors and how they are used to control caching of physical memory locations.

13.1 INTERNAL CACHES, TLBS, AND BUFFERS

The Intel 64 and IA-32 architectures support cache, translation look aside buffers (TLBs), and a store buffer for temporary on-chip (and external) storage of instructions and data. (Figure 13-1 shows the arrangement of caches, TLBs, and the store buffer for the Pentium 4 and Intel Xeon processors.) Table 13-1 shows the characteristics of these caches and buffers for the Pentium 4, Intel Xeon, P6 family, and Pentium processors. **The sizes and characteristics of these units are machine specific and may change in future versions of the processor.** The CPUID instruction returns the sizes and characteristics of the caches and buffers for the processor on which the instruction is executed. See “CPUID—CPU Identification” in Chapter 3, “Instruction Set Reference, A-L,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

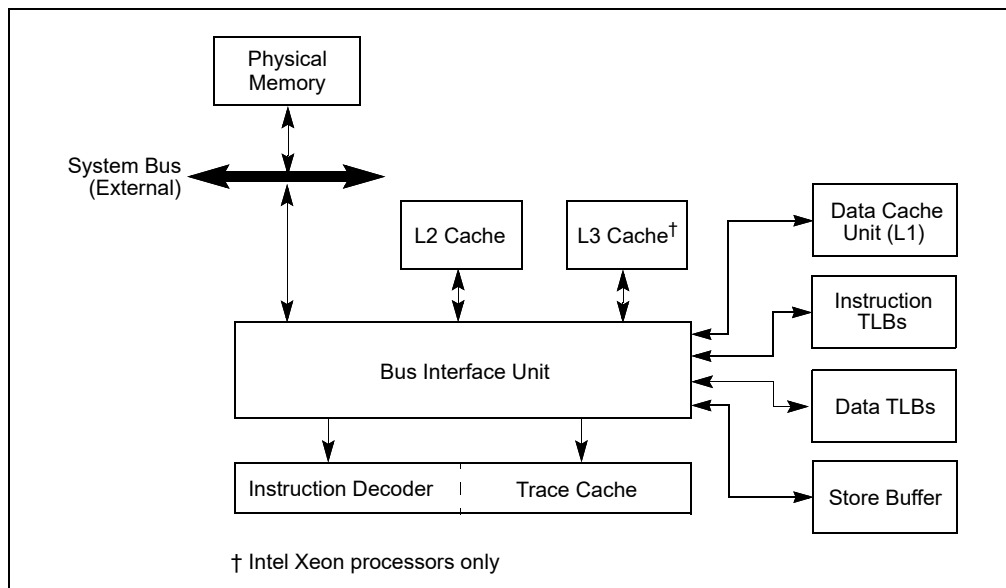


Figure 13-1. Cache Structure of the Pentium 4 and Intel Xeon Processors

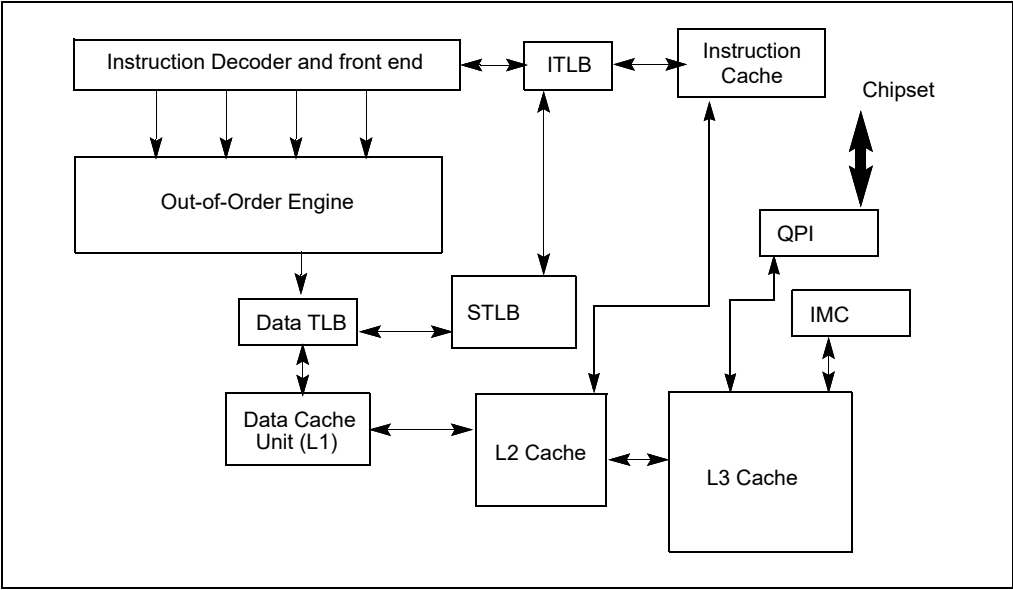


Figure 13-2. Cache Structure of the Intel Core i7 Processors

Figure 13-2 shows the cache arrangement of Intel Core i7 processor.

Table 13-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors

Cache or Buffer	Characteristics
Trace Cache ¹	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst® microarchitecture): 12 Kμops, 8-way set associative. ▪ Intel Core i7, Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: not implemented. ▪ P6 family and Pentium processors: not implemented.
L1 Instruction Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): not implemented. ▪ Intel Core i7 processor: 32-KByte, 4-way set associative. ▪ Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M processor: 32-KByte, 8-way set associative. ▪ P6 family and Pentium processors: 8- or 16-KByte, 4-way set associative, 32-byte cache line size; 2-way set associative for earlier Pentium processors.
L1 Data Cache	▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 8-KByte, 4-way set associative, 64-byte cache line size. ▪ Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 16-KByte, 8-way set associative, 64-byte cache line size. ▪ Intel Atom processors: 24-KByte, 6-way set associative, 64-byte cache line size. ▪ Intel Core i7, Intel Core 2 Duo, Intel Core Duo, Intel Core Solo, Pentium M and Intel Xeon processors: 32-KByte, 8-way set associative, 64-byte cache line size. ▪ P6 family processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KBytes, 2-way set associative for earlier P6 family processors. ▪ Pentium processors: 16-KByte, 4-way set associative, 32-byte cache line size; 8-KByte, 2-way set associative for earlier Pentium processors.

Table 13-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors (Contd.)

Cache or Buffer	Characteristics
L2 Unified Cache	- Intel Core 2 Duo and Intel Xeon processors: up to 4-MByte (or 4MBx2 in quadcore processors), 16-way set associative, 64-byte cache line size. - Intel Core 2 Duo and Intel Xeon processors: up to 6-MByte (or 6MBx2 in quadcore processors), 24-way set associative, 64-byte cache line size. - Intel Core i7, i5, i3 processors: 256-KByte, 8-way set associative, 64-byte cache line size. - Intel Atom processors: 512-KByte, 8-way set associative, 64-byte cache line size. - Intel Core Duo, Intel Core Solo processors: 2-MByte, 8-way set associative, 64-byte cache line size - Pentium 4 and Intel Xeon processors: 256, 512, 1024, or 2048-KByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. - Pentium M processor: 1 or 2-MByte, 8-way set associative, 64-byte cache line size. - P6 family processors: 128-KByte, 256-KByte, 512-KByte, 1-MByte, or 2-MByte, 4-way set associative, 32-byte cache line size. - Pentium processor (external optional): System specific, typically 256- or 512-KByte, 4-way set associative, 32-byte cache line size.
L3 Unified Cache	- Intel Xeon processors: 512-KByte, 1-MByte, 2-MByte, or 4-MByte, 8-way set associative, 64-byte cache line size, 128-byte sector size. - Intel Core i7 processor, Intel Xeon processor 5500: Up to 8MByte, 16-way set associative, 64-byte cache line size. - Intel Xeon processor 5600: Up to 12MByte, 64-byte cache line size. - Intel Xeon processor 7500: Up to 24MByte, 64-byte cache line size.
Instruction TLB (4-KByte Pages)	- Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 128 entries, 4-way set associative. - Intel Atom processors: 32-entries, fully associative. - Intel Core i7, i5, i3 processors: 64-entries per thread (128-entries per core), 4-way set associative. - Intel Core 2 Duo, Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. - P6 family processors: 32 entries, 4-way set associative. - Pentium processor: 32 entries, 4-way set associative; fully set associative for Pentium processors with MMX technology.
Data TLB (4-KByte Pages)	- Intel Core i7, i5, i3 processors, DTLB0: 64-entries, 4-way set associative. - Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 256 entries, 4 ways. - Intel Atom processors: 16-entry-per-thread micro-TLB, fully associative; 64-entry DTLB, 4-way set associative; 16-entry PDE cache, fully associative. - Pentium 4 and Intel Xeon processors (Based on Intel NetBurst microarchitecture): 64 entry, fully set associative, shared with large page DTLB. - Intel Core Duo, Intel Core Solo processors, Pentium M processor: 128 entries, 4-way set associative. - Pentium and P6 family processors: 64 entries, 4-way set associative; fully set, associative for Pentium processors with MMX technology.
Instruction TLB (Large Pages)	- Intel Core i7, i5, i3 processors: 7-entries per thread, fully associative. - Intel Core 2 Duo processors: 4 entries, 4 ways. - Pentium 4 and Intel Xeon processors: large pages are fragmented. - Intel Core Duo, Intel Core Solo, Pentium M processor: 2 entries, fully associative. - P6 family processors: 2 entries, fully associative. - Pentium processor: Uses same TLB as used for 4-KByte pages.
Data TLB (Large Pages)	- Intel Core i7, i5, i3 processors, DTLB0: 32-entries, 4-way set associative. - Intel Core 2 Duo processors: DTLB0, 16 entries, DTLB1, 32 entries, 4 ways. - Intel Atom processors: 8 entries, 4-way set associative. - Pentium 4 and Intel Xeon processors: 64 entries, fully set associative; shared with small page data TLBs. - Intel Core Duo, Intel Core Solo, Pentium M processor: 8 entries, fully associative. - P6 family processors: 8 entries, 4-way set associative. - Pentium processor: 8 entries, 4-way set associative; uses same TLB as used for 4-KByte pages in Pentium processors with MMX technology.
Second-level Unified TLB (4-KByte Pages)	Intel Core i7, i5, i3 processor, STLB: 512-entries, 4-way set associative.

Table 13-1. Characteristics of the Caches, TLBs, Store Buffer, and Write Combining Buffer in Intel 64 and IA-32 Processors (Contd.)

Cache or Buffer	Characteristics
Store Buffer	- Intel Core i7, i5, i3 processors: 32 entries. - Intel Core 2 Duo processors: 20 entries. - Intel Atom processors: 8 entries, used for both WC and store buffers. - Pentium 4 and Intel Xeon processors: 24 entries. - Pentium M processor: 16 entries. - P6 family processors: 12 entries. - Pentium processor: 2 buffers, 1 entry each (Pentium processors with MMX technology have 4 buffers for 4 entries).
Write Combining (WC) Buffer	- Intel Core 2 Duo processors: 8 entries. - Intel Atom processors: 8 entries, used for both WC and store buffers. - Pentium 4 and Intel Xeon processors: 6 or 8 entries. - Intel Core Duo, Intel Core Solo, Pentium M processors: 6 entries. - P6 family processors: 4 entries.

NOTES:

1 Introduced to the IA-32 architecture in the Pentium 4 and Intel Xeon processors.

Intel 64 and IA-32 processors may implement four types of caches: the trace cache, the level 1 (L1) cache, the level 2 (L2) cache, and the level 3 (L3) cache. See Figure 13-1. Cache availability is described below:

- **Intel Core i7, i5, i3 processor family and Intel Xeon processor family based on Nehalem microarchitecture and Westmere microarchitecture** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache. Each processor core has its own L1 and L2. The L3 cache is an inclusive, unified data and instruction cache, shared by all processor cores inside a physical package. No trace cache is implemented.
- **Intel® Core™ 2 processor family and Intel® Xeon® processor family based on Intel® Core™ microarchitecture** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip; it is shared between two processor cores in a dual-core processor implementation. Quad-core processors have two L2, each shared by two processor cores. No trace cache is implemented.
- **Intel Atom® processor** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache is located on the processor chip. No trace cache is implemented.
- **Intel® Core™ Solo and Intel® Core™ Duo processors** — The L1 cache is divided into two sections: one section is dedicated to caching instructions (pre-decoded instructions) and the other caches data. The L2 cache is a unified data and instruction cache located on the processor chip. It is shared between two processor cores in a dual-core processor implementation. No trace cache is implemented.
- **Pentium® 4 and Intel® Xeon® processors Based on Intel NetBurst® microarchitecture** — The trace cache caches decoded instructions (μ ops) from the instruction decoder and the L1 cache contains data. The L2 and L3 caches are unified data and instruction caches located on the processor chip. Dualcore processors have two L2, one in each processor core. Note that the L3 cache is only implemented on some Intel Xeon processors.
- **P6 family processors** — The L1 cache is divided into two sections: one dedicated to caching instructions (pre-decoded instructions) and the other to caching data. The L2 cache is a unified data and instruction cache located on the processor chip. P6 family processors do not implement a trace cache.
- **Pentium® processors** — The L1 cache has the same structure as on P6 family processors. There is no trace cache. The L2 cache is a unified data and instruction cache external to the processor chip on earlier Pentium processors and implemented on the processor chip in later Pentium processors. For Pentium processors where the L2 cache is external to the processor, access to the cache is through the system bus.

For Intel Core i7 processors and processors based on Intel Core, Intel Atom, and Intel NetBurst microarchitectures, Intel Core Duo, Intel Core Solo and Pentium M processors, the cache lines for the L1 and L2 caches (and L3 caches if supported) are 64 bytes wide. The processor always reads a cache line from system memory beginning on a 64-byte boundary. (A 64-byte aligned cache line begins at an address with its 6 least-significant bits clear.) A cache

line can be filled from memory with a 8-transfer burst transaction. The caches do not support partially-filled cache lines, so caching even a single doubleword requires caching an entire line.

The L1 and L2 cache lines in the P6 family and Pentium processors are 32 bytes wide, with cache line reads from system memory beginning on a 32-byte boundary (5 least-significant bits of a memory address clear.) A cache line can be filled from memory with a 4-transfer burst transaction. Partially-filled cache lines are not supported.

The trace cache in processors based on Intel NetBurst microarchitecture is available in all execution modes: protected mode, system management mode (SMM), and real-address mode. The L1, L2, and L3 caches are also available in all execution modes; however, use of them must be handled carefully in SMM (see Section 33.4.2, “SMRAM Caching”).

The TLBs store the most recently used page-directory and page-table entries. They speed up memory accesses when paging is enabled by reducing the number of memory accesses that are required to read the page tables stored in system memory. The TLBs are divided into four groups: instruction TLBs for 4-KByte pages, data TLBs for 4-KByte pages; instruction TLBs for large pages (2-MByte, 4-MByte or 1-GByte pages), and data TLBs for large pages. The TLBs are normally active only in protected mode with paging enabled. When paging is disabled or the processor is in real-address mode, the TLBs maintain their contents until explicitly or implicitly flushed (see Section 13.9, “Invalidating the Translation Lookaside Buffers (TLBs)”).

Processors based on Intel Core microarchitectures implement one level of instruction TLB and two levels of data TLB. Intel Core i7 processor provides a second-level unified TLB.

The store buffer is associated with the processors instruction execution units. It allows writes to system memory and/or the internal caches to be saved and in some cases combined to optimize the processor’s bus accesses. The store buffer is always enabled in all execution modes.

The processor’s caches are for the most part transparent to software. When enabled, instructions and data flow through these caches without the need for explicit software control. However, knowledge of the behavior of these caches may be useful in optimizing software performance. For example, knowledge of cache dimensions and replacement algorithms gives an indication of how large of a data structure can be operated on at once without causing cache thrashing.

In multiprocessor systems, maintenance of cache consistency may, in rare circumstances, require intervention by system software. For these rare cases, the processor provides privileged cache control instructions for use in flushing caches and forcing memory ordering.

There are several instructions that software can use to improve the performance of the L1, L2, and L3 caches, including the PREFETCHh, CLFLUSH, and CLFLUSHOPT instructions and the non-temporal move instructions (MOVNTI, MOVNTDQ, MOVNTPS, and MOVNTPD). The use of these instructions are discussed in Section 13.5.5, “Cache Management Instructions.”

13.2 CACHING TERMINOLOGY

IA-32 processors (beginning with the Pentium processor) and Intel 64 processors use the MESI (modified, exclusive, shared, invalid) cache protocol to maintain consistency with internal caches and caches in other processors (see Section 13.4, “Cache Control Protocol”).

When the processor recognizes that an operand being read from memory is cacheable, the processor reads an entire cache line into the appropriate cache (L1, L2, L3, or all). This operation is called a **cache line fill**. If the memory location containing that operand is still cached the next time the processor attempts to access the operand, the processor can read the operand from the cache instead of going back to memory. This operation is called a **cache hit**.

When the processor attempts to write an operand to a cacheable area of memory, it first checks if a cache line for that memory location exists in the cache. If a valid cache line does exist, the processor (depending on the write policy currently in force) can write the operand into the cache instead of writing it out to system memory. This operation is called a **write hit**. If a write misses the cache (that is, a valid cache line is not present for area of memory being written to), the processor performs a cache line fill, write allocation. Then it writes the operand into the cache line and (depending on the write policy currently in force) can also write it out to memory. If the operand is to be written out to memory, it is written first into the store buffer, and then written from the store buffer to memory when the system bus is available. (Note that for the Pentium processor, write misses do not result in a cache line fill; they always result in a write to memory. For this processor, only read misses result in cache line fills.)

When operating in an MP system, IA-32 processors (beginning with the Intel486 processor) and Intel 64 processors have the ability to **snoop** other processor's accesses to system memory and to their internal caches. They use this snooping ability to keep their internal caches consistent both with system memory and with the caches in other processors on the bus. For example, in the Pentium and P6 family processors, if through snooping one processor detects that another processor intends to write to a memory location that it currently has cached in **shared state**, the snooping processor will invalidate its cache line forcing it to perform a cache line fill the next time it accesses the same memory location.

Beginning with the P6 family processors, if a processor detects (through snooping) that another processor is trying to access a memory location that it has modified in its cache, but has not yet written back to system memory, the snooping processor will signal the other processor (by means of the HITM# signal) that the cache line is held in modified state and will perform an implicit write-back of the modified data. The implicit write-back is transferred directly to the initial requesting processor and snooped by the memory controller to assure that system memory has been updated. Here, the processor with the valid data may pass the data to the other processors without actually writing it to system memory; however, it is the responsibility of the memory controller to snoop this operation and update memory.

13.3 METHODS OF CACHING AVAILABLE

The processor allows any area of system memory to be cached in the L1, L2, and L3 caches. In individual pages or regions of system memory, it allows the type of caching (also called **memory type**) to be specified (see Section 13.5). Memory types currently defined for the Intel 64 and IA-32 architectures are (see Table 13-2):

- **Strong Uncacheable (UC)** —System memory locations are not cached. All reads and writes appear on the system bus and are executed in program order without reordering. No speculative memory accesses, page-table walks, or prefetches of speculated branch targets are made. This type of cache-control is useful for memory-mapped I/O devices. When used with normal RAM, it greatly reduces processor performance.

NOTE

The behavior of x87 and SIMD instructions referencing memory is implementation dependent. In some implementations, accesses to UC memory may occur more than once. To ensure predictable behavior, use loads and stores of general purpose registers to access UC memory that may have read or write side effects.

Table 13-2. Memory Types and Their Properties

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering
Uncacheable (UC-)	No	No	No	Strong Ordering. Can only be selected through the PAT. Can be overridden by WC in MTRRs.
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering.
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.
Write Protected (WP)	Yes for reads; no for writes	No	Yes	Speculative Processor Ordering. Available by programming MTRRs.

- **Uncacheable (UC-)** — Has same characteristics as the strong uncacheable (UC) memory type, except that this memory type can be overridden by programming the MTRRs for the WC memory type. This memory type is available in processor families starting from the Pentium III processors and can only be selected through the PAT.

- **Write Combining (WC)** — System memory locations are not cached (as with uncacheable memory) and coherency is not enforced by the processor's bus coherency protocol. Speculative reads are allowed. Writes may be delayed and combined in the write combining buffer (WC buffer) to reduce memory accesses. If the WC buffer is partially filled, the writes may be delayed until the next occurrence of a serializing event; such as an SFENCE or MFENCE instruction, CUID or other serializing instruction, a read or write to uncached memory, an interrupt occurrence, or an execution of a LOCK instruction (including one with an XACQUIRE or XRELEASE prefix). In addition, an execution of the XEND instruction (to end a transactional region) evicts any writes that were buffered before the corresponding execution of the XBEGIN instruction (to begin the transactional region) before evicting any writes that were performed inside the transactional region.

This type of cache-control is appropriate for video frame buffers, where the order of writes is unimportant as long as the writes update memory so they can be seen on the graphics display. See Section 13.3.1, "Buffering of Write Combining Memory Locations," for more information about caching the WC memory type. This memory type is available in the Pentium Pro and Pentium II processors by programming the MTRRs; or in processor families starting from the Pentium III processors by programming the MTRRs or by selecting it through the PAT.

- **Write-through (WT)** — Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. All writes are written to a cache line (when possible) and through to system memory. When writing through to memory, invalid cache lines are never filled, and valid cache lines are either filled or invalidated. Write combining is allowed. This type of cache-control is appropriate for frame buffers or when there are devices on the system bus that access system memory, but do not perform snooping of memory accesses. It enforces coherency between caches in the processors and system memory.
- **Write-back (WB)** — Writes and reads to and from system memory are cached. Reads come from cache lines on cache hits; read misses cause cache fills. Speculative reads are allowed. Write misses cause cache line fills (in processor families starting with the P6 family processors), and writes are performed entirely in the cache, when possible. Write combining is allowed. The write-back memory type reduces bus traffic by eliminating many unnecessary writes to system memory. Writes to a cache line are not immediately forwarded to system memory; instead, they are accumulated in the cache. The modified cache lines are written to system memory later, when a write-back operation is performed. Write-back operations are triggered when cache lines need to be deallocated, such as when new cache lines are being allocated in a cache that is already full. They also are triggered by the mechanisms used to maintain cache consistency. This type of cache-control provides the best performance, but it requires that all devices that access system memory on the system bus be able to snoop memory accesses to ensure system memory and cache coherency.
- **Write protected (WP)** — Reads come from cache lines when possible, and read misses cause cache fills. Writes are propagated to the system bus and cause corresponding cache lines on all processors on the bus to be invalidated. Speculative reads are allowed. This memory type is available in processor families starting from the P6 family processors by programming the MTRRs (see Table 13-6).

Table 13-3 shows which of these caching methods are available in the Pentium, P6 Family, Pentium 4, and Intel Xeon processors.

Table 13-3. Methods of Caching Available in Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4, Intel Xeon, P6 Family, and Pentium Processors

Memory Type	Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4 and Intel Xeon Processors	P6 Family Processors	Pentium Processor
Strong Uncacheable (UC)	Yes	Yes	Yes
Uncacheable (UC-)	Yes	Yes*	No
Write Combining (WC)	Yes	Yes	No
Write Through (WT)	Yes	Yes	Yes
Write Back (WB)	Yes	Yes	Yes
Write Protected (WP)	Yes	Yes	No

NOTE:

* Introduced in the Pentium III processor; not available in the Pentium Pro or Pentium II processors

13.3.1 Buffering of Write Combining Memory Locations

Writes to the WC memory type are not cached in the typical sense of the word cached. They are retained in an internal write combining buffer (WC buffer) that is separate from the internal L1, L2, and L3 caches and the store buffer. The WC buffer is not snooped and thus does not provide data coherency. Buffering of writes to WC memory is done to allow software a small window of time to supply more modified data to the WC buffer while remaining as non-intrusive to software as possible. The buffering of writes to WC memory also causes data to be collapsed; that is, multiple writes to the same memory location will leave the last data written in the location and the other writes will be lost.

The size and structure of the WC buffer is not architecturally defined. For the Intel Core 2 Duo, Intel Atom, Intel Core Duo, Pentium M, Pentium 4 and Intel Xeon processors; the WC buffer is made up of several 64-byte WC buffers. For the P6 family processors, the WC buffer is made up of several 32-byte WC buffers.

When software begins writing to WC memory, the processor begins filling the WC buffers one at a time. When one or more WC buffers has been filled, the processor has the option of evicting the buffers to system memory. The protocol for evicting the WC buffers is implementation dependent and should not be relied on by software for system memory coherency. When using the WC memory type, software **must** be sensitive to the fact that the writing of data to system memory is being delayed and **must** deliberately empty the WC buffers when system memory coherency is required.

Once the processor has started to evict data from the WC buffer into system memory, it will make a bus-transaction style decision based on how much of the buffer contains valid data. If the buffer is full (for example, all bytes are valid), the processor will execute a burst-write transaction on the bus. This results in all 32 bytes (P6 family processors) or 64 bytes (Pentium 4 and more recent processor) being transmitted on the data bus in a single burst transaction. If one or more of the WC buffer's bytes are invalid (for example, have not been written by software), the processor will transmit the data to memory using "partial write" transactions (one chunk at a time, where a "chunk" is 8 bytes).

This will result in a maximum of 4 partial write transactions (for P6 family processors) or 8 partial write transactions (for the Pentium 4 and more recent processors) for one WC buffer of data sent to memory.

The WC memory type is weakly ordered by definition. Once the eviction of a WC buffer has started, the data is subject to the weak ordering semantics of its definition. Ordering is not maintained between the successive allocation/deallocation of WC buffers (for example, writes to WC buffer 1 followed by writes to WC buffer 2 may appear as buffer 2 followed by buffer 1 on the system bus). When a WC buffer is evicted to memory as partial writes there is no guaranteed ordering between successive partial writes (for example, a partial write for chunk 2 may appear on the bus before the partial write for chunk 1 or vice versa).

The only elements of WC propagation to the system bus that are guaranteed are those provided by transaction atomicity. For example, with a P6 family processor, a completely full WC buffer will always be propagated as a single 32-bit burst transaction using any chunk order. In a WC buffer eviction where data will be evicted as partials, all data contained in the same chunk (0 mod 8 aligned) will be propagated simultaneously. Likewise, for more recent processors starting with those based on Intel NetBurst microarchitectures, a full WC buffer will always be propagated as a single burst transactions, using any chunk order within a transaction. For partial buffer propagations, all data contained in the same chunk will be propagated simultaneously.

13.3.2 Choosing a Memory Type

The simplest system memory model does not use memory-mapped I/O with read or write side effects, does not include a frame buffer, and uses the write-back memory type for all memory. An I/O agent can perform direct memory access (DMA) to write-back memory and the cache protocol maintains cache coherency.

A system can use strong uncacheable memory for other memory-mapped I/O, and should always use strong uncacheable memory for memory-mapped I/O with read side effects.

Dual-ported memory can be considered a write side effect, making relatively prompt writes desirable, because those writes cannot be observed at the other port until they reach the memory agent. A system can use strong uncacheable, uncacheable, write-through, or write-combining memory for frame buffers or dual-ported memory that contains pixel values displayed on a screen. Frame buffer memory is typically large (a few megabytes) and is usually written more than it is read by the processor. Using strong uncacheable memory for a frame buffer generates very large amounts of bus traffic, because operations on the entire buffer are implemented using partial writes rather than line writes. Using write-through memory for a frame buffer can displace almost all other useful cached

lines in the processor's L2 and L3 caches and L1 data cache. Therefore, systems should use write-combining memory for frame buffers whenever possible.

Software can use page-level cache control, to assign appropriate effective memory types when software will not access data structures in ways that benefit from write-back caching. For example, software may read a large data structure once and not access the structure again until the structure is rewritten by another agent. Such a large data structure should be marked as uncacheable, or reading it will evict cached lines that the processor will be referencing again.

A similar example would be a write-only data structure that is written to (to export the data to another agent), but never read by software. Such a structure can be marked as uncacheable, because software never reads the values that it writes (though as uncacheable memory, it will be written using partial writes, while as write-back memory, it will be written using line writes, which may not occur until the other agent reads the structure and triggers implicit write-backs).

On the Pentium III, Pentium 4, and more recent processors, new instructions are provided that give software greater control over the caching, prefetching, and the write-back characteristics of data. These instructions allow software to use weakly ordered or processor ordered memory types to improve processor performance, but when necessary to force strong ordering on memory reads and/or writes. They also allow software greater control over the caching of data. For a description of these instructions and their intended use, see Section 13.5.5, "Cache Management Instructions."

13.3.3 Code Fetches in Uncacheable Memory

Programs may execute code from uncacheable (UC) memory, but the implications are different from accessing data in UC memory. When doing code fetches, the processor never transitions from cacheable code to UC code speculatively. It also never speculatively fetches branch targets that result in UC code.

The processor may fetch the same UC cache line multiple times in order to decode an instruction once. It may decode consecutive UC instructions in a cache line without fetching between each instruction. It may also fetch additional cachelines from the same or a consecutive 4-KByte page in order to decode one non-speculative UC instruction (this can be true even when the instruction is contained fully in one line).

Because of the above and because cache line sizes may change in future processors, software should avoid placing memory-mapped I/O with read side effects in the same page or in a subsequent page used to execute UC code.

13.4 CACHE CONTROL PROTOCOL

The following section describes the cache control protocol currently defined for the Intel 64 and IA-32 architectures.

In the L1 data cache and in the L2/L3 unified caches, the MESI (modified, exclusive, shared, invalid) cache protocol maintains consistency with caches of other processors. The L1 data cache and the L2/L3 unified caches have two MESI status flags per cache line. Each line can be marked as being in one of the states defined in Table 13-4. In general, the operation of the MESI protocol is transparent to programs.

Table 13-4. MESI Cache Line States

Cache Line State	M (Modified)	E (Exclusive)	S (Shared)	I (Invalid)
This cache line is valid?	Yes	Yes	Yes	No
The memory copy is...	Out of date	Valid	Valid	—
Copies exist in caches of other processors?	No	No	Maybe	Maybe
A write to this line ...	Does not go to the system bus.	Does not go to the system bus.	Causes the processor to gain exclusive ownership of the line.	Goes directly to the system bus.

The L1 instruction cache in P6 family processors implements only the “SI” part of the MESI protocol, because the instruction cache is not writable. The instruction cache monitors changes in the data cache to maintain consistency between the caches when instructions are modified. See Section 13.6, “Self-Modifying Code,” for more information on the implications of caching instructions.

13.5 CACHE CONTROL

The Intel 64 and IA-32 architectures provide a variety of mechanisms for controlling the caching of data and instructions and for controlling the ordering of reads and writes between the processor, the caches, and memory. These mechanisms can be divided into two groups:

- **Cache control registers and bits** — The Intel 64 and IA-32 architectures define several dedicated registers and various bits within control registers and page- and directory-table entries that control the caching system memory locations in the L1, L2, and L3 caches. These mechanisms control the caching of virtual memory pages and of regions of physical memory.
- **Cache control and memory ordering instructions** — The Intel 64 and IA-32 architectures provide several instructions that control the caching of data, the ordering of memory reads and writes, and the prefetching of data. These instructions allow software to control the caching of specific data structures, to control memory coherency for specific locations in memory, and to force strong memory ordering at specific locations in a program.

The following sections describe these two groups of cache control mechanisms.

13.5.1 Cache Control Registers and Bits

Figure 13-3 depicts cache-control mechanisms in IA-32 processors. Other than for the matter of memory address space, these work the same in Intel 64 processors.

The Intel 64 and IA-32 architectures provide the following cache-control registers and bits for use in enabling or restricting caching to various pages or regions in memory:

- **CD flag, bit 30 of control register CR0** — Controls caching of system memory locations (see Section 2.5, “Control Registers”). If the CD flag is clear, caching is enabled for the whole of system memory, but may be restricted for individual pages or regions of memory by other cache-control mechanisms. When the CD flag is set, caching is restricted in the processor’s caches (cache hierarchy) for the P6 and more recent processor families and prevented for the Pentium processor (see note below). With the CD flag set, however, the caches will still respond to snoop traffic. Caches should be explicitly flushed to ensure memory coherency. For highest processor performance, both the CD and the NW flags in control register CR0 should be cleared. Table 13-5 shows the interaction of the CD and NW flags.

The effect of setting the CD flag is somewhat different for processor families starting with P6 family than the Pentium processor (see Table 13-5). To ensure memory coherency after the CD flag is set, the caches should be explicitly flushed (see Section 13.5.3, “Preventing Caching”). Setting the CD flag for the P6 and more recent processor families modifies cache line fill and update behavior. Also, setting the CD flag on these processors do not force strict ordering of memory accesses unless the MTRRs are disabled and/or all memory is referenced as uncached (see Section 10.2.5, “Strengthening or Weakening the Memory-Ordering Model”).

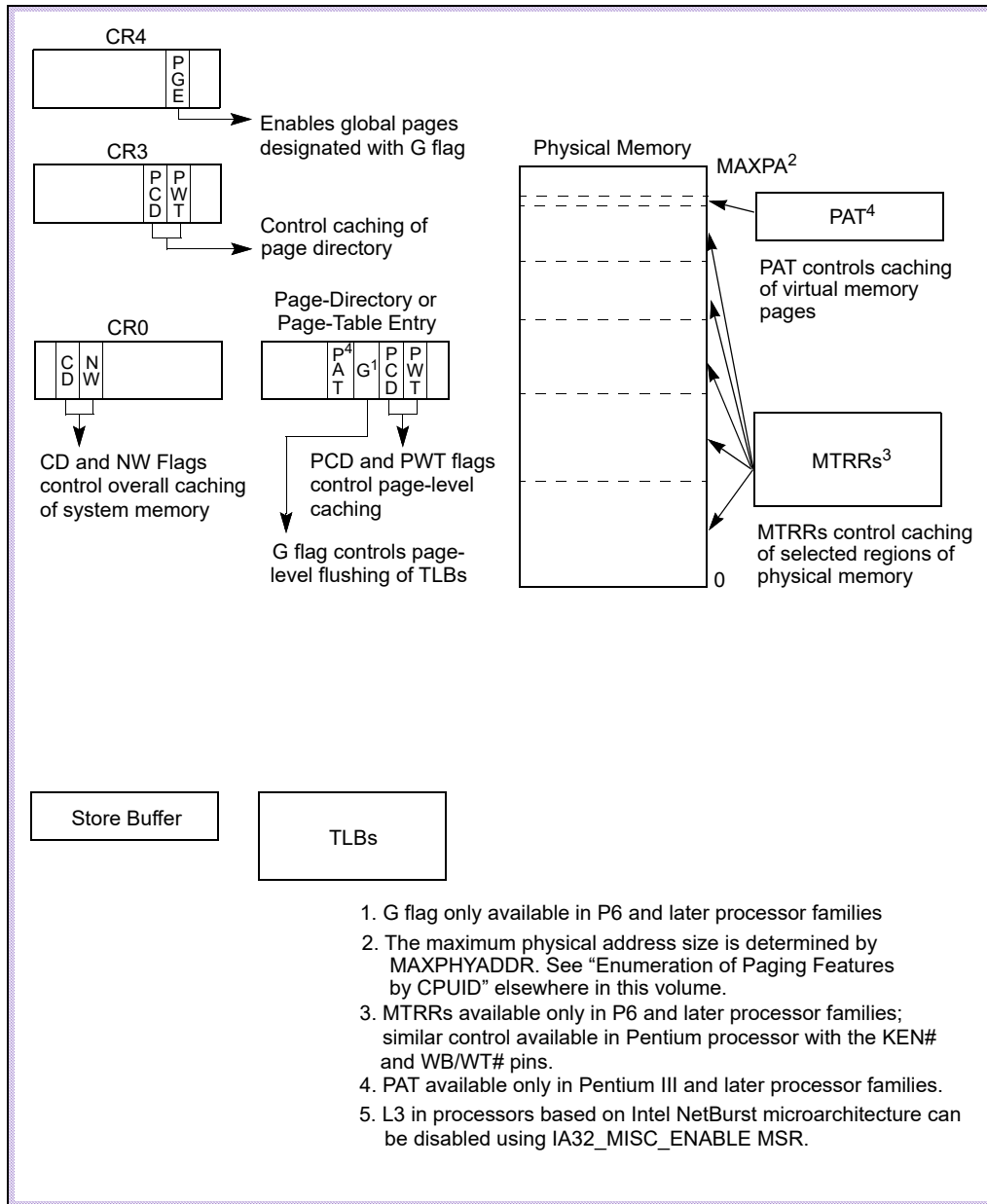


Figure 13-3. Cache-Control Registers and Bits Available in Intel 64 and IA-32 Processors

Table 13-5. Cache Operating Modes

CD	NW	Caching and Read/Write Policy	L1	L2/L3 ¹
0	0	Normal Cache Mode. Highest performance cache operation. - Read hits access the cache; read misses may cause replacement. - Write hits update the cache. - Only writes to shared lines and write misses update system memory. - Write misses cause cache line fills. - Write hits can change shared lines to modified under control of the MTRRs and with associated read invalidation cycle. (Pentium processor only.) Write misses do not cause cache line fills. (Pentium processor only.) Write hits can change shared lines to exclusive under control of WB/WT#. - Invalidation is allowed. - External snoop traffic is supported.	Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes
0	1	Invalid setting. Generates a general-protection exception (#GP) with an error code of 0.	NA	NA
1	0	No-fill Cache Mode. Memory coherency is maintained.³ (Pentium 4 and later processor families.) State of processor after a power up or reset. - Read hits access the cache; read misses do not cause replacement (see Pentium 4 and Intel Xeon processors reference below). - Write hits update the cache. - Only writes to shared lines and write misses update system memory. - Write misses access memory. - Write hits can change shared lines to exclusive under control of the MTRRs and with associated read invalidation cycle. (Pentium processor only.) Write hits can change shared lines to exclusive under control of the WB/WT#. (P6 and later processor families only.) Strict memory ordering is not enforced unless the MTRRs are disabled and/or all memory is referenced as uncached (see Section 7.2.4., "Strengthening or Weakening the Memory Ordering Model"). - Invalidation is allowed. - External snoop traffic is supported.	Yes Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes
1	1	Memory coherency is not maintained.^{2, 3} (P6 family and Pentium processors.) State of the processor after a power up or reset. - Read hits access the cache; read misses do not cause replacement. - Write hits update the cache and change exclusive lines to modified. - Shared lines remain shared after write hit. - Write misses access memory. - Invalidation is inhibited when snooping; but is allowed with INVD and WBINVD instructions. - External snoop traffic is supported.	Yes Yes Yes Yes Yes Yes No	Yes Yes Yes Yes Yes Yes Yes

NOTES:

1. The L2/L3 column in this table is definitive for the Pentium 4, Intel Xeon, and P6 family processors. It is intended to represent what could be implemented in a system based on a Pentium processor with an external, platform specific, write-back L2 cache.
2. The Pentium 4 and more recent processor families do not support this mode; setting the CD and NW bits to 1 selects the no-fill cache mode.
3. Not supported In Intel Atom processors. If CD = 1 in an Intel Atom processor, caching is disabled.

- **NW flag, bit 29 of control register CR0** — Controls the write policy for system memory locations (see Section 2.5, “Control Registers”). If the NW and CD flags are clear, write-back is enabled for the whole of system memory, but may be restricted for individual pages or regions of memory by other cache-control mechanisms. Table 13-5 shows how the other combinations of CD and NW flags affects caching.

NOTES

For the Pentium 4 and Intel Xeon processors, the NW flag is a don't care flag; that is, when the CD flag is set, the processor uses the no-fill cache mode, regardless of the setting of the NW flag.

For Intel Atom processors, the NW flag is a don't care flag; that is, when the CD flag is set, the processor disables caching, regardless of the setting of the NW flag.

For the Pentium processor, when the L1 cache is disabled (the CD and NW flags in control register CR0 are set), external snoops are accepted in DP (dual-processor) systems and inhibited in uniprocessor systems.

When snoops are inhibited, address parity is not checked and APCHK# is not asserted for a corrupt address; however, when snoops are accepted, address parity is checked and APCHK# is asserted for corrupt addresses.

- **PCD and PWT flags in paging-structure entries** — Control the memory type used to access paging structures and pages (see Section 5.9, “Paging and Memory Typing”).
- **PCD and PWT flags in control register CR3** — Control the memory type used to access the first paging structure of the current paging-structure hierarchy (see Section 5.9, “Paging and Memory Typing”).
- **G (global) flag in the page-directory and page-table entries (introduced to the IA-32 architecture in the P6 family processors)** — Controls the flushing of TLB entries for individual pages. See Section 5.10, “Caching Translation Information,” for more information about this flag.
- **PGE (page global enable) flag in control register CR4** — Enables the establishment of global pages with the G flag. See Section 5.10, “Caching Translation Information,” for more information about this flag.
- **Memory type range registers (MTRRs) (introduced in P6 family processors)** — Control the type of caching used in specific regions of physical memory. Any of the caching types described in Section 13.3, “Methods of Caching Available,” can be selected. See Section 13.11, “Memory Type Range Registers (MTRRs),” for a detailed description of the MTRRs.
- **Page Attribute Table (PAT) MSR (introduced in the Pentium III processor)** — Extends the memory typing capabilities of the processor to permit memory types to be assigned on a page-by-page basis (see Section 13.12, “Page Attribute Table (PAT)”).
- **Third-Level Cache Disable flag, bit 6 of the IA32_MISC_ENABLE MSR (Available only in processors based on Intel NetBurst microarchitecture)** — Allows the L3 cache to be disabled and enabled, independently of the L1 and L2 caches.
- **KEN# and WB/WT# pins (Pentium processor)** — Allow external hardware to control the caching method used for specific areas of memory. They perform similar (but not identical) functions to the MTRRs in the P6 family processors.
- **PCD and PWT pins (Pentium processor)** — These pins (which are associated with the PCD and PWT flags in control register CR3 and in the page-directory and page-table entries) permit caching in an external L2 cache to be controlled on a page-by-page basis, consistent with the control exercised on the L1 cache of these processors. The P6 and more recent processor families do not provide these pins because the L2 cache is internal to the chip package.

13.5.2 Precedence of Cache Controls

The cache control flags and MTRRs operate hierarchically for restricting caching. That is, if the CD flag is set, caching is prevented globally (see Table 13-5). If the CD flag is clear, the page-level cache control flags and/or the MTRRs can be used to restrict caching. If there is an overlap of page-level and MTRR caching controls, the mechanism that prevents caching has precedence. For example, if an MTRR makes a region of system memory uncachable, a page-level caching control cannot be used to enable caching for a page in that region. The converse is also

true; that is, if a page-level caching control designates a page as uncacheable, an MTRR cannot be used to make the page cacheable.

In cases where there is an overlap in the assignment of the write-back and write-through caching policies to a page and a region of memory, the write-through policy takes precedence. The write-combining policy (which can only be assigned through an MTRR or the PAT) takes precedence over either write-through or write-back.

The selection of memory types at the page level varies depending on whether PAT is being used to select memory types for pages, as described in the following sections.

On processors based on Intel NetBurst microarchitecture, the third-level cache can be disabled by bit 6 of the IA32_MISC_ENABLE MSR. Using IA32_MISC_ENABLE[bit 6] takes precedence over the CD flag, MTRRs, and PAT for the L3 cache in those processors. That is, when the third-level cache disable flag is set (cache disabled), the other cache controls have no effect on the L3 cache; when the flag is clear (enabled), the cache controls have the same effect on the L3 cache as they have on the L1 and L2 caches.

IA32_MISC_ENABLE[bit 6] is not supported in Intel Core i7 processors, nor processors based on Intel Core, and Intel Atom microarchitectures.

13.5.2.1 Selecting Memory Types for Pentium Pro and Pentium II Processors

The Pentium Pro and Pentium II processors do not support the PAT. Here, the effective memory type for a page is selected with the MTRRs and the PCD and PWT bits in the page-table or page-directory entry for the page. Table 13-6 describes the mapping of MTRR memory types and page-level caching attributes to effective memory types, when normal caching is in effect (the CD and NW flags in control register CR0 are clear). Combinations that appear in gray are implementation-defined for the Pentium Pro and Pentium II processors. System designers are encouraged to avoid these implementation-defined combinations.

Table 13-6. Effective Page-Level Memory Type for Pentium Pro and Pentium II Processors

MTRR Memory Type ¹	PCD Value	PWT Value	Effective Memory Type
UC	X	X	UC
WC	0	0	WC
	0	1	WC
	1	0	WC
	1	1	UC
WT	0	X	WT
	1	X	UC
WP	0	0	WP
	0	1	WP
	1	0	WC
	1	1	UC
WB	0	0	WB
	0	1	WT
	1	X	UC

NOTE:

1. These effective memory types also apply to the Pentium 4, Intel Xeon, and Pentium III processors when the PAT bit is not used (set to 0) in page-table and page-directory entries.

When normal caching is in effect, the effective memory type shown in Table 13-6 is determined using the following rules:

1. If the PCD and PWT attributes for the page are both 0, then the effective memory type is identical to the MTRR-defined memory type.

2. If the PCD flag is set, then the effective memory type is UC.
3. If the PCD flag is clear and the PWT flag is set, the effective memory type is WT for the WB memory type and the MTRR-defined memory type for all other memory types.
4. Setting the PCD and PWT flags to opposite values is considered model-specific for the WP and WC memory types and architecturally-defined for the WB, WT, and UC memory types.

13.5.2.2 Selecting Memory Types for Pentium III and More Recent Processor Families

The Intel Core 2 Duo, Intel Atom, Intel Core Duo, Intel Core Solo, Pentium M, Pentium 4, Intel Xeon, and Pentium III processors use the PAT to select effective page-level memory types. Here, a memory type for a page is selected by the MTRRs and the value in a PAT entry that is selected with the PAT, PCD, and PWT bits in a page-table or page-directory entry (see Section 13.12.3, “Selecting a Memory Type from the PAT”). Table 13-7 describes the mapping of MTRR memory types and PAT entry types to effective memory types, when normal caching is in effect (the CD and NW flags in control register CR0 are clear).

Table 13-7. Effective Page-Level Memory Types for Pentium III and More Recent Processor Families

MTRR Memory Type	PAT Entry Value	Effective Memory Type
UC	UC	UC ¹
	UC-	UC ¹
	WC	WC
	WT	UC ¹
	WB	UC ¹
	WP	UC ¹
WC	UC	UC ²
	UC-	WC
	WC	WC
	WT	UC ^{2,3}
	WB	WC
	WP	UC ^{2,3}
WT	UC	UC ²
	UC-	UC ²
	WC	WC
	WT	WT
	WB	WT
	WP	WP ³
WB	UC	UC ²
	UC-	UC ²
	WC	WC
	WT	WT
	WB	WB
	WP	WP

Table 13-7. Effective Page-Level Memory Types for Pentium III and More Recent Processor Families (Contd.)

MTRR Memory Type	PAT Entry Value	Effective Memory Type
WP	UC	UC ²
	UC-	WC ³
	WC	WC
	WT	WT ³
	WB	WP
	WP	WP

NOTES:

1. The UC attribute comes from the MTRRs and the processors are not required to snoop their caches since the data could never have been cached. This attribute is preferred for performance reasons.
2. The UC attribute came from the page-table or page-directory entry and processors are required to check their caches because the data may be cached due to page aliasing, which is not recommended.
3. These combinations were specified as “undefined” in previous editions of the Intel® 64 and IA-32 Architectures Software Developer’s Manual. However, all processors that support both the PAT and the MTRRs determine the effective page-level memory types for these combinations as given.

13.5.2.3 Writing Values Across Pages with Different Memory Types

If two adjoining pages in memory have different memory types, and a word or longer operand is written to a memory location that crosses the page boundary between those two pages, the operand might be written to memory twice. This action does not present a problem for writes to actual memory; however, if a device is mapped to the memory space assigned to the pages, the device might malfunction.

13.5.3 Preventing Caching

To disable the L1, L2, and L3 caches after they have been enabled and have received cache fills, perform the following steps:

1. Enter the no-fill cache mode. (Set the CD flag in control register CR0 to 1 and the NW flag to 0.
2. Flush all caches using the WBINVD instruction.
3. Disable the MTRRs and set the default memory type to uncached or set all MTRRs for the uncached memory type (see the discussion of the discussion of the TYPE field and the E flag in Section 13.11.2.1, “IA32_MTRR_DEF_TYPE MSR”).

The caches must be flushed (step 2) after the CD flag is set to ensure system memory coherency. If the caches are not flushed, cache hits on reads will still occur and data will be read from valid cache lines.

The intent of the three separate steps listed above address three distinct requirements: (i) discontinue new data replacing existing data in the cache (ii) ensure data already in the cache are evicted to memory, (iii) ensure subsequent memory references observe UC memory type semantics. Different processor implementation of caching control hardware may allow some variation of software implementation of these three requirements. See note below.

NOTES

Setting the CD flag in control register CR0 modifies the processor’s caching behavior as indicated in Table 13-5, but setting the CD flag alone may not be sufficient across all processor families to force the effective memory type for all physical memory to be UC nor does it force strict memory ordering, due to hardware implementation variations across different processor families. To force the UC memory type and strict memory ordering on all of physical memory, it is sufficient to either program the MTRRs for all physical memory to be UC memory type or disable all MTRRs.

For the Pentium 4 and Intel Xeon processors, after the sequence of steps given above has been executed, the cache lines containing the code between the end of the WBINVD instruction and before the MTRRS have actually been disabled may be retained in the cache hierarchy. Here, to

remove code from the cache completely, a second WBINVD instruction must be executed after the MTRRs have been disabled.

For Intel Atom processors, setting the CD flag forces all physical memory to observe UC semantics (without requiring memory type of physical memory to be set explicitly). Consequently, software does not need to issue a second WBINVD as some other processor generations might require.

13.5.4 Disabling and Enabling the L3 Cache

On processors based on Intel NetBurst microarchitecture, the third-level cache can be disabled by bit 6 of the IA32_MISC_ENABLE MSR. The third-level cache disable flag (bit 6 of the IA32_MISC_ENABLE MSR) allows the L3 cache to be disabled and enabled, independently of the L1 and L2 caches. Prior to using this control to disable or enable the L3 cache, software should disable and flush all the processor caches, as described earlier in Section 13.5.3, “Preventing Caching,” to prevent loss of information stored in the L3 cache. After the L3 cache has been disabled or enabled, caching for the whole processor can be restored.

Newer Intel 64 processor with L3 do not support IA32_MISC_ENABLE[bit 6], the procedure described in Section 13.5.3, “Preventing Caching,” apply to the entire cache hierarchy.

13.5.5 Cache Management Instructions

The Intel 64 and IA-32 architectures provide several instructions for managing the L1, L2, and L3 caches. The INVD and WBINVD instructions are privileged instructions and operate on the L1, L2, and L3 caches as a whole. The PREFETCHh, CLFLUSH, and CLFLUSHOPT instructions and the non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD) offer more granular control over caching, and are available to all privileged levels.

The INVD and WBINVD instructions are used to invalidate the contents of the L1, L2, and L3 caches. The INVD instruction invalidates all internal cache entries, then generates a special-function bus cycle that indicates that external caches also should be invalidated. The INVD instruction should be used with care. It does not force a write-back of modified cache lines; therefore, data stored in the caches and not written back to system memory will be lost. Unless there is a specific requirement or benefit to invalidating the caches without writing back the modified lines (such as, during testing or fault recovery where cache coherency with main memory is not a concern), software should use the WBINVD instruction.

The WBINVD instruction first writes back any modified lines in all the internal caches, then invalidates the contents of the L1, L2, and L3 caches. It ensures that cache coherency with main memory is maintained regardless of the write policy in effect (that is, write-through or write-back). Following this operation, the WBINVD instruction generates one (P6 family processors) or two (Pentium and Intel486 processors) special-function bus cycles to indicate to external cache controllers that write-back of modified data followed by invalidation of external caches should occur. The amount of time or cycles for WBINVD to complete will vary due to the size of different cache hierarchies and other factors. As a consequence, the use of the WBINVD instruction can have an impact on interrupt/event response time.

The PREFETCHh instructions allow a program to suggest to the processor that a cache line from a specified location in system memory be prefetched into the cache hierarchy (see Section 13.8, “Explicit Caching”).

The CLFLUSH and CLFLUSHOPT instructions allow selected cache lines to be flushed from memory. These instructions give a program the ability to explicitly free up cache space, when it is known that cached section of system memory will not be accessed in the near future.

The non-temporal move instructions (MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, and MOVNTPD) allow data to be moved from the processor’s registers directly into system memory without being also written into the L1, L2, and/or L3 caches. These instructions can be used to prevent cache pollution when operating on data that is going to be modified only once before being stored back into system memory. These instructions operate on data in the general-purpose, MMX, and XMM registers.

13.5.6 L1 Data Cache Context Mode

L1 data cache context mode is a feature of processors based on the Intel NetBurst microarchitecture that support Intel Hyper-Threading Technology. When CPUID.01H:ECX[10] = 1, the processor supports setting L1 data cache context mode using the L1 data cache context mode flag (IA32_MISC_ENABLE[bit 24]). Selectable modes are adaptive mode (default) and shared mode.

The BIOS is responsible for configuring the L1 data cache context mode.

13.5.6.1 Adaptive Mode

Adaptive mode facilitates L1 data cache sharing between logical processors. When running in adaptive mode, the L1 data cache is shared across logical processors in the same core if:

- CR3 control registers for logical processors sharing the cache are identical.
- The same paging mode is used by logical processors sharing the cache.

In this situation, the entire L1 data cache is available to each logical processor (instead of being competitively shared).

If CR3 values are different for the logical processors sharing an L1 data cache or the logical processors use different paging modes, processors compete for cache resources. This reduces the effective size of the cache for each logical processor. Aliasing of the cache is not allowed (which prevents data thrashing).

13.5.6.2 Shared Mode

In shared mode, the L1 data cache is competitively shared between logical processors. This is true even if the logical processors use identical CR3 registers and paging modes.

In shared mode, linear addresses in the L1 data cache can be aliased, meaning that one linear address in the cache can point to different physical locations. The mechanism for resolving aliasing can lead to thrashing. For this reason, IA32_MISC_ENABLE[bit 24] = 0 is the preferred configuration for processors based on the Intel NetBurst microarchitecture that support Intel Hyper-Threading Technology.

13.6 SELF-MODIFYING CODE

A write to a memory location in a code segment that is currently cached in the processor causes the associated cache line (or lines) to be invalidated. This check is based on the physical address of the instruction. In addition, the P6 family and Pentium processors check whether a write to a code segment may modify an instruction that has been prefetched for execution. If the write affects a prefetched instruction, the prefetch queue is invalidated. This latter check is based on the linear address of the instruction. For the Pentium 4 and Intel Xeon processors, a write or a snoop of an instruction in a code segment, where the target instruction is already decoded and resident in the trace cache, invalidates the entire trace cache. The latter behavior means that programs that self-modify code can cause severe degradation of performance when run on the Pentium 4 and Intel Xeon processors.

In practice, the check on linear addresses should not create compatibility problems among IA-32 processors. Applications that include self-modifying code use the same linear address for modifying and fetching the instruction. Systems software, such as a debugger, that might possibly modify an instruction using a different linear address than that used to fetch the instruction, will execute a serializing operation, such as a CPUID instruction, before the modified instruction is executed, which will automatically resynchronize the instruction cache and prefetch queue. (See Section 10.1.3, "Handling Self- and Cross-Modifying Code," for more information about the use of self-modifying code.)

For Intel486 processors, a write to an instruction in the cache will modify it in both the cache and memory, but if the instruction was prefetched before the write, the old version of the instruction could be the one executed. To prevent the old instruction from being executed, flush the instruction prefetch unit by coding a jump instruction immediately after any write that modifies an instruction.

13.7 IMPLICIT CACHING (PENTIUM 4, INTEL® XEON®, AND P6 FAMILY PROCESSORS)

Implicit caching occurs when a memory element is made potentially cacheable, although the element may never have been accessed in the normal von Neumann sequence. Implicit caching occurs on the P6 and more recent processor families due to aggressive prefetching, branch prediction, and TLB miss handling. Implicit caching is an extension of the behavior of existing Intel386, Intel486, and Pentium processor systems, since software running on these processor families also has not been able to deterministically predict the behavior of instruction prefetch.

To avoid problems related to implicit caching, the operating system must explicitly invalidate the cache when changes are made to cacheable data that the cache coherency mechanism does not automatically handle. This includes writes to dual-ported or physically aliased memory boards that are not detected by the snooping mechanisms of the processor, and changes to page-table entries in memory.

The code in Example 13-1 shows the effect of implicit caching on page-table entries. The linear address F000H points to physical location B000H (the page-table entry for F000H contains the value B000H), and the page-table entry for linear address F000 is PTE_F000.

Example 13-1. Effect of Implicit Caching on Page-Table Entries

```
mov EAX, CR3; Invalidate the TLB
mov CR3, EAX; by copying CR3 to itself
mov PTE_F000, A000H; Change F000H to point to A000H
mov EBX, [F000H];
```

Because of speculative execution in the P6 and more recent processor families, the last MOV instruction performed would place the value at physical location B000H into EBX, rather than the value at the new physical address A000H. This situation is remedied by placing a TLB invalidation between the load and the store.

13.8 EXPLICIT CACHING

The Pentium III processor introduced four new instructions, the `PREFETCHh` instructions, that provide software with explicit control over the caching of data. These instructions provide “hints” to the processor that the data requested by a `PREFETCHh` instruction should be read into cache hierarchy now or as soon as possible, in anticipation of its use. The instructions provide different variations of the hint that allow selection of the cache level into which data will be read.

The `PREFETCHh` instructions can help reduce the long latency typically associated with reading data from memory and thus help prevent processor “stalls.” However, these instructions should be used judiciously. Overuse can lead to resource conflicts and hence reduce the performance of an application. Also, these instructions should only be used to prefetch data from memory; they should not be used to prefetch instructions. For more detailed information on the proper use of the prefetch instruction, refer to Chapter 7, “Optimizing Cache Usage,” in the Intel® 64 and IA-32 Architectures Optimization Reference Manual.

13.9 INVALIDATING THE TRANSLATION LOOKASIDE BUFFERS (TLBS)

The processor updates its address translation caches (TLBs) transparently to software. Several mechanisms are available, however, that allow software and hardware to invalidate the TLBs either explicitly or as a side effect of another operation. Most details are given in Section 5.10.4, “Invalidation of TLBs and Paging-Structure Caches.” In addition, the following operations invalidate all TLB entries, irrespective of the setting of the G flag:

- Asserting or de-asserting the `FLUSH#` pin.
- (Pentium 4, Intel Xeon, and later processors only.) Writing to an MTRR (with a `WRMSR` instruction).
- Writing to control register `CR0` to modify the PG or PE flag.

- (Pentium 4, Intel Xeon, and later processors only.) Writing to control register CR4 to modify the PSE, PGE, or PAE flag.
- Writing to control register CR4 to change the PCIDE flag from 1 to 0.

See Section 5.10, “Caching Translation Information,” for additional information about the TLBs.

13.10 STORE BUFFER

Intel 64 and IA-32 processors temporarily store each write (store) to memory in a store buffer. The store buffer improves processor performance by allowing the processor to continue executing instructions without having to wait until a write to memory and/or to a cache is complete. It also allows writes to be delayed for more efficient use of memory-access bus cycles.

In general, the existence of the store buffer is transparent to software, even in systems that use multiple processors. The processor ensures that write operations are always carried out in program order. It also ensures that the contents of the store buffer are always drained to memory in the following situations:

- When an exception or interrupt is generated.
- (P6 and more recent processor families only) When a serializing instruction is executed.
- When an I/O instruction is executed.
- When a LOCK operation is performed.
- (P6 and more recent processor families only) When a BINIT operation is performed.
- (Pentium III, and more recent processor families only) When using an SFENCE instruction to order stores.
- (Pentium 4 and more recent processor families only) When using an MFENCE instruction to order stores.

The discussion of write ordering in Section 10.2, “Memory Ordering,” gives a detailed description of the operation of the store buffer.

13.11 MEMORY TYPE RANGE REGISTERS (MTRRS)

The following section pertains only to the P6 and more recent processor families.

The memory type range registers (MTRRs) provide a mechanism for associating the memory types (see Section 13.3, “Methods of Caching Available”) with physical-address ranges in system memory. They allow the processor to optimize operations for different types of memory such as RAM, ROM, frame-buffer memory, and memory-mapped I/O devices. They also simplify system hardware design by eliminating the memory control pins used for this function on earlier IA-32 processors and the external logic needed to drive them.

The MTRR mechanism allows multiple ranges to be defined in physical memory, and it defines a set of model-specific registers (MSRs) for specifying the type of memory that is contained in each range. Table 13-8 shows the memory types that can be specified and their properties; Figure 13-4 shows the mapping of physical memory with MTRRs. See Section 13.3, “Methods of Caching Available,” for a more detailed description of each memory type.

Following a hardware reset, the P6 and more recent processor families disable all the fixed and variable MTRRs, which in effect makes all of physical memory uncacheable. Initialization software should then set the MTRRs to a specific, system-defined memory map. Typically, the BIOS (basic input/output system) software configures the MTRRs. The operating system or executive is then free to modify the memory map using the normal page-level cacheability attributes.

In a multiprocessor system using a processor in the P6 family or a more recent family, each processor **MUST** use the identical MTRR memory map so that software will have a consistent view of memory.

NOTE

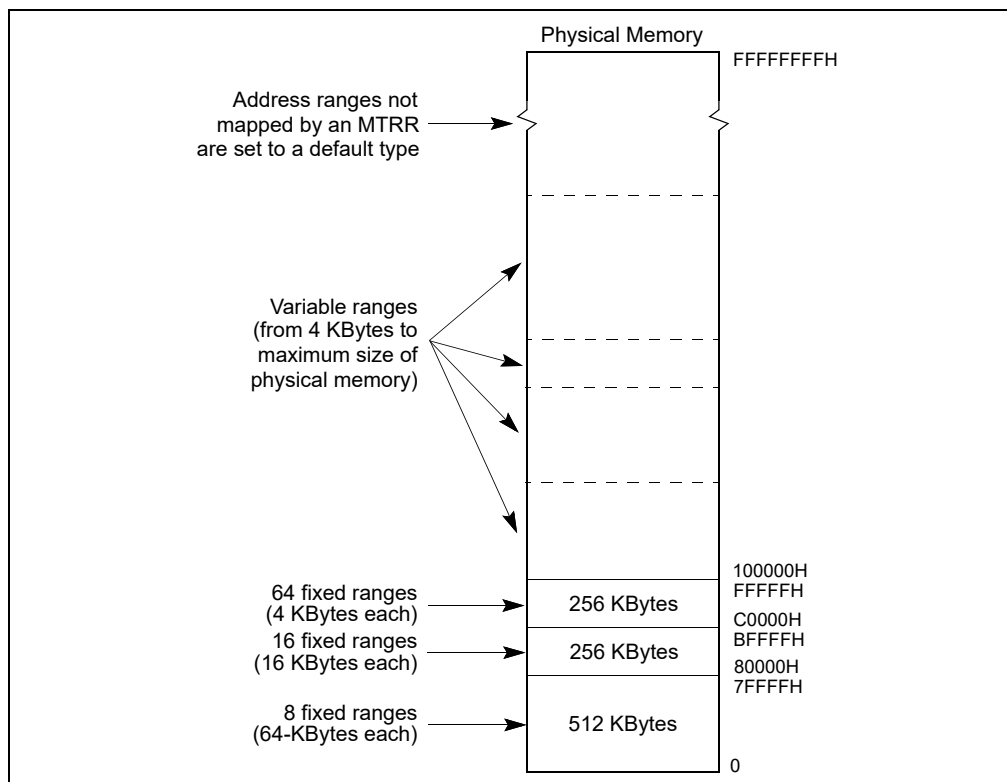
In multiple processor systems, the operating system must maintain MTRR consistency between all the processors in the system (that is, all processors must use the same MTRR values). The P6 and more recent processor families provide no hardware support for maintaining this consistency.

Table 13-8. Memory Types That Can Be Encoded in MTRRs

Memory Type and Mnemonic	Encoding in MTRR
Uncacheable (UC)	00H
Write Combining (WC)	01H
Reserved*	02H
Reserved*	03H
Write-through (WT)	04H
Write-protected (WP)	05H
Writeback (WB)	06H
Reserved*	7H through FFH

NOTE:

* Use of these encodings results in a general-protection exception (#GP).

**Figure 13-4. Mapping Physical Memory With MTRRs****13.11.1 MTRR Feature Identification**

The availability of the MTRR feature is model-specific. Software can determine if MTRRs are supported on a processor by executing the CPUID instruction and reading the state of the MTRR flag (bit 12) in the feature information register (EDX).

If the MTRR flag is set (indicating that the processor implements MTRRs), additional information about MTRRs can be obtained from the 64-bit IA32_MTRRCAP MSR (named MTRRcap MSR for the P6 family processors). The IA32_MTRRCAP MSR is a read-only MSR that can be read with the RDMSR instruction. Figure 13-5 shows the contents of the IA32_MTRRCAP MSR. The functions of the flags and field in this register are as follows:

- **VCNT (variable range registers count) field, bits 0 through 7** — Indicates the number of variable ranges implemented on the processor.
- **FIX (fixed range registers supported) flag, bit 8** — Fixed range MTRRs (IA32_MTRR_FIX64K_00000 through IA32_MTRR_FIX4K_0F8000) are supported when set; no fixed range registers are supported when clear.
- **WC (write combining) flag, bit 10** — The write-combining (WC) memory type is supported when set; the WC type is not supported when clear.
- **SMRR (System-Management Range Register) flag, bit 11** — The system-management range register (SMRR) interface is supported when bit 11 is set; the SMRR interface is not supported when clear.
- **SEAMRR (SEAM Range Register) flag, bit 15** — The SEAMRR interface to restrict access to a specified memory address range is supported when bit 15 is set.

Bit 9, bits 14:12, and bits 63:16 of the IA32_MTRRCAP MSR are not currently defined. If software attempts to write to the IA32_MTRRCAP MSR, a general-protection exception (#GP) is generated.

Software must read IA32_MTRRCAP VCNT field to determine the number of variable MTRRs and query other feature bits in IA32_MTRRCAP to determine additional capabilities that are supported in a processor. For example, some processors may report a value of '8' in the VCNT field, other processors may report VCNT with different values.

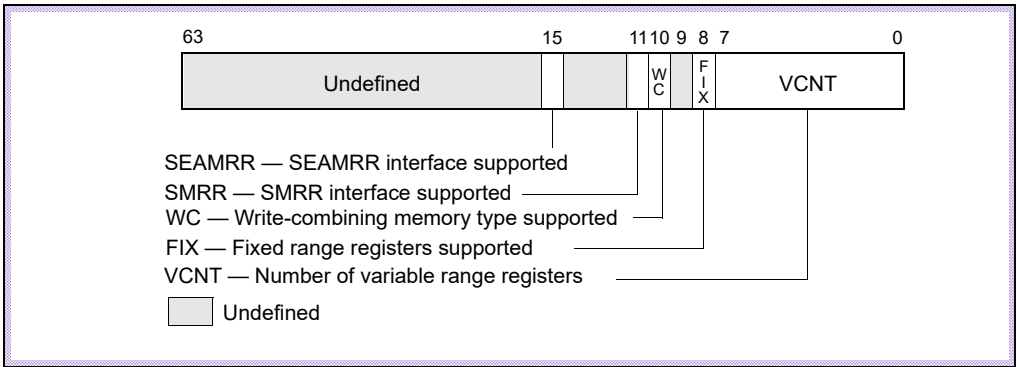


Figure 13-5. IA32_MTRRCAP Register

13.11.2 Setting Memory Ranges with MTRRs

The memory ranges and the types of memory specified in each range are set by three groups of registers: the IA32_MTRR_DEF_TYPE MSR, the fixed-range MTRRs, and the variable range MTRRs. These registers can be read and written to using the RDMSR and WRMSR instructions, respectively. The IA32_MTRRCAP MSR indicates the availability of these registers on the processor (see Section 13.11.1, “MTRR Feature Identification”).

13.11.2.1 IA32_MTRR_DEF_TYPE MSR

The IA32_MTRR_DEF_TYPE MSR (named MTRRdefType MSR for the P6 family processors) sets the default properties of the regions of physical memory that are not encompassed by MTRRs. The functions of the flags and field in this register are as follows:

- **Type field, bits 0 through 7** — Indicates the default memory type used for those physical memory address ranges that do not have a memory type specified for them by an MTRR (see Table 13-8 for the encoding of this field). The legal values for this field are 0, 1, 4, 5, and 6. All other values result in a general-protection exception (#GP) being generated.

Intel recommends the use of the UC (uncached) memory type for all physical memory addresses where memory does not exist. To assign the UC type to nonexistent memory locations, it can either be specified as the default type in the Type field or be explicitly assigned with the fixed and variable MTRRs.

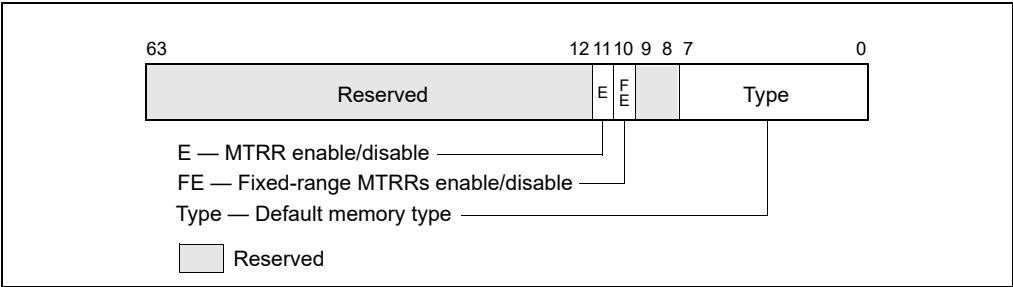


Figure 13-6. IA32_MTRR_DEF_TYPE MSR

- **FE (fixed MTRRs enabled) flag, bit 10** — Fixed-range MTRRs are enabled when set; fixed-range MTRRs are disabled when clear. When the fixed-range MTRRs are enabled, they take priority over the variable-range MTRRs when overlaps in ranges occur. If the fixed-range MTRRs are disabled, the variable-range MTRRs can still be used and can map the range ordinarily covered by the fixed-range MTRRs.
- **E (MTRRs enabled) flag, bit 11** — MTRRs are enabled when set; all MTRRs are disabled when clear, and the UC memory type is applied to all of physical memory. When this flag is set, the FE flag can disable the fixed-range MTRRs; when the flag is clear, the FE flag has no affect. When the E flag is set, the type specified in the default memory type field is used for areas of memory not already mapped by either a fixed or variable MTRR.

Bits 8 and 9, and bits 12 through 63, in the IA32_MTRR_DEF_TYPE MSR are reserved; the processor generates a general-protection exception (#GP) if software attempts to write nonzero values to them.

13.11.2.2 Fixed Range MTRRs

The fixed memory ranges are mapped with 11 fixed-range registers of 64 bits each. Each of these registers is divided into 8-bit fields that are used to specify the memory type for each of the sub-ranges the register controls:

- **Register IA32_MTRR_FIX64K_00000** — Maps the 512-KByte address range from 0H to 7FFFFH. This range is divided into eight 64-KByte sub-ranges.
- **Registers IA32_MTRR_FIX16K_80000 and IA32_MTRR_FIX16K_A0000** — Maps the two 128-KByte address ranges from 80000H to BFFFFH. This range is divided into sixteen 16-KByte sub-ranges, 8 ranges per register.
- **Registers IA32_MTRR_FIX4K_C0000 through IA32_MTRR_FIX4K_F8000** — Maps eight 32-KByte address ranges from C0000H to FFFFFH. This range is divided into sixty-four 4-KByte sub-ranges, 8 ranges per register.

Table 13-9 shows the relationship between the fixed physical-address ranges and the corresponding fields of the fixed-range MTRRs; Table 13-8 shows memory type encoding for MTRRs.

For the P6 family processors, the prefix for the fixed range MTRRs is MTRRfix.

13.11.2.3 Variable Range MTRRs

The Pentium 4, Intel Xeon, and P6 family processors permit software to specify the memory type for m variable-size address ranges, using a pair of MTRRs for each range. The number m of ranges supported is given in bits 7:0 of the IA32_MTRRCAP MSR (see Figure 13-5 in Section 13.11.1).

The first entry in each pair (IA32_MTRR_PHYSBASEn) defines the base address and memory type for the range; the second entry (IA32_MTRR_PHYSMASKn) contains a mask used to determine the address range. The “n” suffix is in the range 0 through m–1 and identifies a specific register pair.

For P6 family processors, the prefixes for these variable range MTRRs are MTRRphysBase and MTRRphysMask.

Table 13-9. Address Mapping for Fixed-Range MTRRs

Address Range (hexadecimal)								MTRR
63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
70000-7FFFF	60000-6FFFF	50000-5FFFF	40000-4FFFF	30000-3FFFF	20000-2FFFF	10000-1FFFF	00000-0FFFF	IA32_MTRR_FIX64K_00000
9C000-9FFFF	98000-9BFFF	94000-97FFF	90000-93FFF	8C000-8FFFF	88000-8BFFF	84000-87FFF	80000-83FFF	IA32_MTRR_FIX16K_80000
BC000-BFFFF	B8000-BBFFF	B4000-B7FFF	B0000-B3FFF	AC000-AFFFF	A8000-ABFFF	A4000-A7FFF	A0000-A3FFF	IA32_MTRR_FIX16K_A0000
C7000-C7FFF	C6000-C6FFF	C5000-C5FFF	C4000-C4FFF	C3000-C3FFF	C2000-C2FFF	C1000-C1FFF	C0000-C0FFF	IA32_MTRR_FIX4K_C0000
CF000-CFFFF	CE000-CEFFF	CD000-CDFFF	CC000-CCFFF	CB000-CBFFF	CA000-CAFFF	C9000-C9FFF	C8000-C8FFF	IA32_MTRR_FIX4K_C8000
D7000-D7FFF	D6000-D6FFF	D5000-D5FFF	D4000-D4FFF	D3000-D3FFF	D2000-D2FFF	D1000-D1FFF	D0000-D0FFF	IA32_MTRR_FIX4K_D0000
DF000-DFFFF	DE000-DEFFF	DD000-DDFFF	DC000-DCFFF	DB000-DBFFF	DA000-DAFFF	D9000-D9FFF	D8000-D8FFF	IA32_MTRR_FIX4K_D8000
E7000-E7FFF	E6000-E6FFF	E5000-E5FFF	E4000-E4FFF	E3000-E3FFF	E2000-E2FFF	E1000-E1FFF	E0000-E0FFF	IA32_MTRR_FIX4K_E0000
EF000-EFFFF	EE000-EEFFF	ED000-EDFFF	EC000-ECFFF	EB000-EBFFF	EA000-EAFFF	E9000-E9FFF	E8000-E8FFF	IA32_MTRR_FIX4K_E8000
F7000-F7FFF	F6000-F6FFF	F5000-F5FFF	F4000-F4FFF	F3000-F3FFF	F2000-F2FFF	F1000-F1FFF	F0000-F0FFF	IA32_MTRR_FIX4K_F0000
FF000-FFFFF	FE000-FEFFF	FD000-FDFFF	FC000-FCFFF	FB000-FBFFF	FA000-FAFFF	F9000-F9FFF	F8000-F8FFF	IA32_MTRR_FIX4K_F8000

Figure 13-7 shows flags and fields in these registers. The functions of these flags and fields are:

- **Type field, bits 0 through 7** — Specifies the memory type for the range (see Table 13-8 for the encoding of this field).
- **PhysBase field, bits MAXPHYADDR-1:12** — Specifies the base address of the address range.¹ This 24-bit value, in the case where MAXPHYADDR is 36 bits, is extended by 12 bits at the low end to form the base address (this automatically aligns the address on a 4-KByte boundary).
- **PhysMask field, bits MAXPHYADDR-1:12** — Specifies a mask (24 bits if the maximum physical address size is 36 bits, 28 bits if the maximum physical address size is 40 bits). The mask determines the range of the region being mapped, according to the following relationships:
 - Address_Within_Range AND PhysMask = PhysBase AND PhysMask
 - This value is extended by 12 bits at the low end to form the mask value. For more information: see Section 13.11.3, “Example Base and Mask Calculations.”
 - The width of the PhysMask field depends on the maximum physical address size supported by the processor. CPUID.80000008H reports the maximum physical address size supported by the processor. If CPUID.80000008H is not available, software may assume that the processor supports a 36-bit physical address size (then PhysMask is 24 bits wide and the upper 28 bits of IA32_MTRR_PHYSMASKn are reserved). See the Note below.
- **V (valid) flag, bit 11** — Enables the register pair when set; disables register pair when clear.

1. MAXPHYADDR is derived from the value enumerated by CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. (For processors that do not support CPUID.80000008H, MAXPHYADDR is 36.)

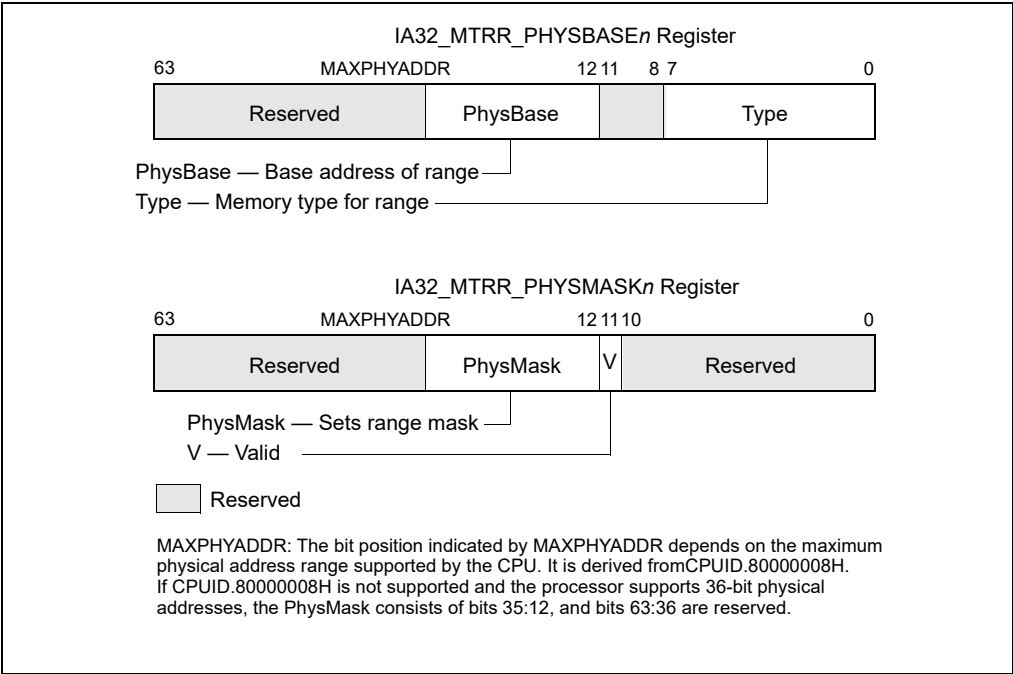


Figure 13-7. IA32_MTRR_PHYSBASE_n and IA32_MTRR_PHYSMASK_n Variable-Range Register Pair

All other bits in the IA32_MTRR_PHYSBASE_n and IA32_MTRR_PHYSMASK_n registers are reserved; the processor generates a general-protection exception (#GP) if software attempts to write to them.

Some mask values can result in ranges that are not continuous. In such ranges, the area not mapped by the mask value is set to the default memory type, unless some other MTRR specifies a type for that range. Intel does not encourage the use of “discontinuous” ranges.

NOTE

It is possible for software to parse the memory descriptions that BIOS provides by using the ACPI/INT15 e820 interface mechanism. This information then can be used to determine how MTRRs are initialized (for example: allowing the BIOS to define valid memory ranges and the maximum memory range supported by the platform, including the processor).

See Section 13.11.4.1, “MTRR Precedences,” for information on overlapping variable MTRR ranges.

13.11.2.4 System-Management Range Register Interface

If IA32_MTRRCAP[bit 11] is set, the processor supports the SMRR interface to restrict access to a specified memory address range used by system-management mode (SMM) software (see Section 33.4.2.1). If the SMRR interface is supported, SMM software is strongly encouraged to use it to protect the SMI code and data stored by SMI handler in the SMRAM region.

The system-management range registers consist of a pair of MSRs (see Figure 13-8). The IA32_SMRR_PHYSBASE MSR defines the base address for the SMRAM memory range and the memory type used to access it in SMM. The IA32_SMRR_PHYSMASK MSR contains a valid bit and a mask that determines the SMRAM address range protected by the SMRR interface. These MSRs may be written only in SMM; an attempt to write them outside of SMM causes a general-protection exception.¹

Figure 13-8 shows flags and fields in these registers. The functions of these flags and fields are the following:

1. For some processor models, these MSRs can be accessed by RDMSR and WRMSR only if the SMRR interface has been enabled using a model-specific bit in the IA32_FEATURE_CONTROL MSR.

- **Type field, bits 0 through 7** — Specifies the memory type for the range (see Table 13-8 for the encoding of this field).
- **PhysBase field, bits 12 through 31** — Specifies the base address of the address range. The address must be less than 4 GBytes and is automatically aligned on a 4-KByte boundary.
- **PhysMask field, bits 12 through 31** — Specifies a mask that determines the range of the region being mapped, according to the following relationships:
 - $\text{Address\_Within\_Range AND PhysMask} = \text{PhysBase AND PhysMask}$
 - This value is extended by 12 bits at the low end to form the mask value. For more information: see Section 13.11.3, “Example Base and Mask Calculations.”
- **V (valid) flag, bit 11** — Enables the register pair when set; disables register pair when clear.

Before attempting to access these SMRR registers, software must test bit 11 in the IA32_MTRRCAP register. If SMRR is not supported, reads from or writes to registers cause general-protection exceptions.

When the valid flag in the IA32_SMRR_PHYSMASK MSR is 1, accesses to the specified address range are treated as follows:

- If the logical processor is in SMM, accesses uses the memory type in the IA32_SMRR_PHYSBASE MSR.
- If the logical processor is not in SMM, write accesses are ignored and read accesses return a fixed value for each byte. The uncacheable memory type (UC) is used in this case.

The above items apply even if the address range specified overlaps with a range specified by the MTRRs.

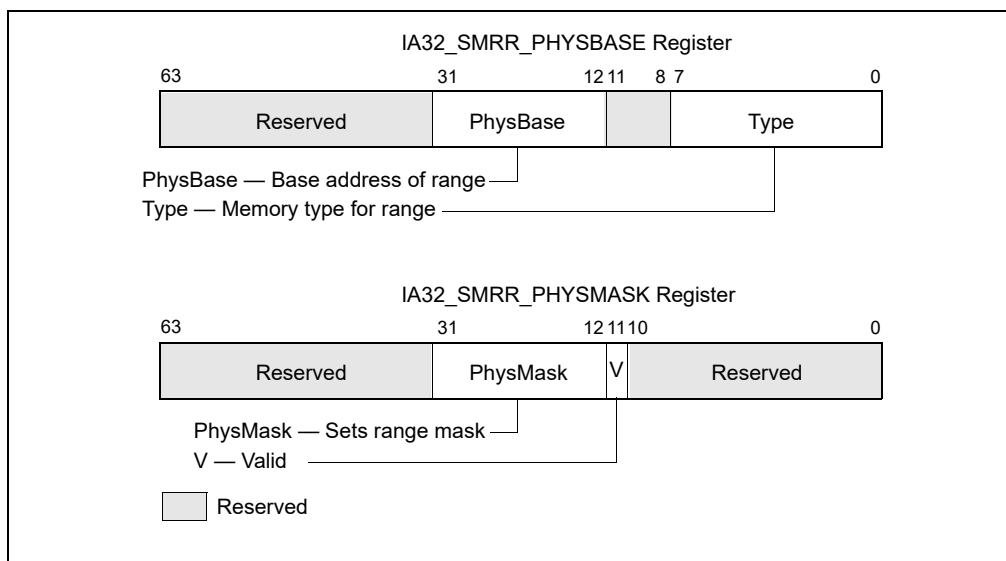


Figure 13-8. IA32_SMRR_PHYSBASE and IA32_SMRR_PHYSMASK SMRR Pair

13.11.3 Example Base and Mask Calculations

The examples in this section apply to processors that support a maximum physical address size of 36 bits. The base and mask values entered in variable-range MTRR pairs are 24-bit values that the processor extends to 36-bits.

For example, to enter a base address of 2 MBytes (200000H) in the IA32_MTRR_PHYSBASE3 register, the 12 least-significant bits are truncated and the value 000200H is entered in the PhysBase field. The same operation must be performed on mask values. For example, to map the address range from 200000H to 3FFFFFFH (2 MBytes to 4 MBytes), a mask value of FFFE0000H is required. Again, the 12 least-significant bits of this mask value are truncated, so that the value entered in the PhysMask field of IA32_MTRR_PHYSMASK3 is FFFE00H. This mask is chosen so that when any address in the 200000H to 3FFFFFFH range is ANDed with the mask value, it will return the same value as when the base address is ANDed with the mask value (which is 200000H).

To map the address range from 400000H to 7FFFFFFH (4 MBytes to 8 MBytes), a base value of 000400H is entered in the PhysBase field and a mask value of FFFC00H is entered in the PhysMask field.

Example 13-2. Setting-Up Memory for a System

Here is an example of setting up the MTRRs for an system. Assume that the system has the following characteristics:

- 96 MBytes of system memory is mapped as write-back memory (WB) for highest system performance.
- A custom 4-MByte I/O card is mapped to uncached memory (UC) at a base address of 64 MBytes. This restriction forces the 96 MBytes of system memory to be addressed from 0 to 64 MBytes and from 68 MBytes to 100 MBytes, leaving a 4-MByte hole for the I/O card.
- An 8-MByte graphics card is mapped to write-combining memory (WC) beginning at address A0000000H.
- The BIOS area from 15 MBytes to 16 MBytes is mapped to UC memory.

The following settings for the MTRRs will yield the proper mapping of the physical address space for this system configuration.

```
IA32_MTRR_PHYSBASE0 = 0000 0000 0000 0006H
IA32_MTRR_PHYSMASK0 = 0000 000F FC00 0800H
Caches 0-64 MByte as WB cache type.
```

```
IA32_MTRR_PHYSBASE1 = 0000 0000 0400 0006H
IA32_MTRR_PHYSMASK1 = 0000 000F FE00 0800H
Caches 64-96 MByte as WB cache type.
```

```
IA32_MTRR_PHYSBASE2 = 0000 0000 0600 0006H
IA32_MTRR_PHYSMASK2 = 0000 000F FFC0 0800H
Caches 96-100 MByte as WB cache type.
```

```
IA32_MTRR_PHYSBASE3 = 0000 0000 0400 0000H
IA32_MTRR_PHYSMASK3 = 0000 000F FFC0 0800H
Caches 64-68 MByte as UC cache type.
```

```
IA32_MTRR_PHYSBASE4 = 0000 0000 00F0 0000H
IA32_MTRR_PHYSMASK4 = 0000 000F FFF0 0800H
Caches 15-16 MByte as UC cache type.
```

```
IA32_MTRR_PHYSBASE5 = 0000 0000 A000 0001H
IA32_MTRR_PHYSMASK5 = 0000 000F FF80 0800H
Caches A0000000-A0800000 as WC type.
```

This MTRR setup uses the ability to overlap any two memory ranges (as long as the ranges are mapped to WB and UC memory types) to minimize the number of MTRR registers that are required to configure the memory environment. This setup also fulfills the requirement that two register pairs are left for operating system usage.

13.11.3.1 Base and Mask Calculations for Greater-Than 36-bit Physical Address Support

For Intel 64 and IA-32 processors that support greater than 36 bits of physical address size, software should query CPUID.80000008H to determine the maximum physical address. See the [footnote 1 on page 13-24](#).

Example 13-3. Setting-Up Memory for a System with a 40-Bit Address Size

If a processor supports 40-bits of physical address size, then the PhysMask field (in IA32_MTRR_PHYSMASK_n registers) is 28 bits instead of 24 bits. For this situation, Example 13-2 should be modified as follows:

```
IA32_MTRR_PHYSBASE0 = 0000 0000 0000 0006H
IA32_MTRR_PHYSMASK0 = 0000 00FF FC00 0800H
Caches 0-64 MByte as WB cache type.
```

IA32_MTRR_PHYSBASE1 = 0000 0000 0400 0006H

IA32_MTRR_PHYSMASK1 = 0000 00FF FE00 0800H

Caches 64-96 MByte as WB cache type.

IA32_MTRR_PHYSBASE2 = 0000 0000 0600 0006H

IA32_MTRR_PHYSMASK2 = 0000 00FF FFC0 0800H

Caches 96-100 MByte as WB cache type.

IA32_MTRR_PHYSBASE3 = 0000 0000 0400 0000H

IA32_MTRR_PHYSMASK3 = 0000 00FF FFC0 0800H

Caches 64-68 MByte as UC cache type.

IA32_MTRR_PHYSBASE4 = 0000 0000 00F0 0000H

IA32_MTRR_PHYSMASK4 = 0000 00FF FFF0 0800H

Caches 15-16 MByte as UC cache type.

IA32_MTRR_PHYSBASE5 = 0000 0000 A000 0001H

IA32_MTRR_PHYSMASK5 = 0000 00FF FF80 0800H

Caches A0000000-A0800000 as WC type.

13.11.4 Range Size and Alignment Requirement

A range that is to be mapped to a variable-range MTRR must meet the following “power of 2” size and alignment rules:

1. The minimum range size is 4 KBytes and the base address of the range must be on at least a 4-KByte boundary.
2. For ranges greater than 4 KBytes, each range must be of length 2^n and its base address must be aligned on a 2^n boundary, where n is a value equal to or greater than 12. The base-address alignment value cannot be less than its length. For example, an 8-KByte range cannot be aligned on a 4-KByte boundary. It must be aligned on at least an 8-KByte boundary.

13.11.4.1 MTRR Precedences

If the MTRRs are not enabled (by setting the E flag in the IA32_MTRR_DEF_TYPE MSR), then all memory accesses are of the UC memory type. If the MTRRs are enabled, then the memory type used for a memory access is determined as follows:

1. If the physical address falls within the first 1 MByte of physical memory and fixed MTRRs are enabled, the processor uses the memory type stored for the appropriate fixed-range MTRR.
2. Otherwise, the processor attempts to match the physical address with a memory type set by the variable-range MTRRs:
 - If one variable memory range matches, the processor uses the memory type stored in the IA32_MTRR_PHYSBASE n register for that range.
 - If two or more variable memory ranges match and the memory types are identical, then that memory type is used.
 - If two or more variable memory ranges match and one of the memory types is UC, the UC memory type is used.
 - If two or more variable memory ranges match and the memory types are WT and WB, the WT memory type is used.
 - For overlaps not defined by the above rules, processor behavior is undefined.
3. If no fixed or variable memory range matches, the processor uses the default memory type.

13.11.5 MTRR Initialization

On a hardware reset, the P6 and more recent processors clear the valid flags in variable-range MTRRs and clear the E flag in the IA32_MTRR_DEF_TYPE MSR to disable all MTRRs. All other bits in the MTRRs are undefined.

Prior to initializing the MTRRs, software (normally the system BIOS) must initialize all fixed-range and variable-range MTRR register fields to 0. Software can then initialize the MTRRs according to known types of memory, including memory on devices that it auto-configures. Initialization is expected to occur prior to booting the operating system.

See Section 13.11.8, “MTRR Considerations in MP Systems,” for information on initializing MTRRs in MP (multiple-processor) systems.

13.11.6 Remapping Memory Types

A system designer may re-map memory types to tune performance or because a future processor may not implement all memory types supported by the Pentium 4, Intel Xeon, and P6 family processors. The following rules support coherent memory-type re-mappings:

1. A memory type should not be mapped into another memory type that has a weaker memory ordering model. For example, the uncacheable type cannot be mapped into any other type, and the write-back, write-through, and write-protected types cannot be mapped into the weakly ordered write-combining type.
2. A memory type that does not delay writes should not be mapped into a memory type that does delay writes, because applications of such a memory type may rely on its write-through behavior. Accordingly, the write-back type cannot be mapped into the write-through type.
3. A memory type that views write data as not necessarily stored and read back by a subsequent read, such as the write-protected type, can only be mapped to another type with the same behavior (and there are no others for the Pentium 4, Intel Xeon, and P6 family processors) or to the uncacheable type.

In many specific cases, a system designer can have additional information about how a memory type is used, allowing additional mappings. For example, write-through memory with no associated write side effects can be mapped into write-back memory.

13.11.7 MTRR Maintenance Programming Interface

The operating system maintains the MTRRs after booting and sets up or changes the memory types for memory-mapped devices. The operating system should provide a driver and application programming interface (API) to access and set the MTRRs. The function calls `MemTypeGet()` and `MemTypeSet()` define this interface.

13.11.7.1 MemTypeGet() Function

The `MemTypeGet()` function returns the memory type of the physical memory range specified by the parameters base and size. The base address is the starting physical address and the size is the number of bytes for the memory range. The function automatically aligns the base address and size to 4-KByte boundaries. Pseudocode for the `MemTypeGet()` function is given in Example 13-4.

Example 13-4. MemTypeGet() Pseudocode

```

#define MIXED_TYPES -1    /* 0 < MIXED_TYPES || MIXED_TYPES > 256 */

IF CPU_FEATURES.MTRR /* processor supports MTRRs */
    THEN
        Align BASE and SIZE to 4-KByte boundary;
        IF (BASE + SIZE) wrap physical-address space
            THEN return INVALID;
        FI;
        IF MTRRdefType.E = 0
            THEN return UC;
        FI;
        FirstType := Get4KMemType (BASE);
        /* Obtains memory type for first 4-KByte range. */
        /* See Get4KMemType (4KByteRange) in Example 13-5. */
        FOR each additional 4-KByte range specified in SIZE
            NextType := Get4KMemType (4KByteRange);
            IF NextType != FirstType
                THEN return Mixed_Types;
            FI;
        ROF;
        return FirstType;
    ELSE return UNSUPPORTED;
FI;

```

If the processor does not support MTRRs, the function returns UNSUPPORTED. If the MTRRs are not enabled, then the UC memory type is returned. If more than one memory type corresponds to the specified range, a status of MIXED_TYPES is returned. Otherwise, the memory type defined for the range (UC, WC, WT, WB, or WP) is returned.

The pseudocode for the Get4KMemType() function in Example 13-5 obtains the memory type for a single 4-KByte range at a given physical address. The sample code determines whether an PHY_ADDRESS falls within a fixed range by comparing the address with the known fixed ranges: 0 to 7FFFFH (64-KByte regions), 80000H to BFFFFH (16-KByte regions), and C0000H to FFFFFH (4-KByte regions). If an address falls within one of these ranges, the appropriate bits within one of its MTRRs determine the memory type.

Example 13-5. Get4KMemType() Pseudocode

```

IF IA32_MTRRCAP.FIX AND MTRRdefType.FE /* fixed registers enabled */
    THEN IF PHY_ADDRESS is within a fixed range
        return IA32_MTRR_FIX.Type;
FI;
FOR each variable-range MTRR in IA32_MTRRCAP.VCNT
    IF IA32_MTRR_PHYSMASK.V = 0
        THEN continue;
    FI;
    IF (PHY_ADDRESS AND IA32_MTRR_PHYSMASK.Mask) =
        (IA32_MTRR_PHYSBASE.Base
         AND IA32_MTRR_PHYSMASK.Mask)
        THEN
            return IA32_MTRR_PHYSBASE.Type;
    FI;
ROF;
return MTRRdefType.Type;

```

13.11.7.2 MemTypeSet() Function

The MemTypeSet() function in Example 13-6 sets a MTRR for the physical memory range specified by the parameters base and size to the type specified by type. The base address and size are multiples of 4 KBytes and the size is not 0.

Example 13-6. MemTypeSet Pseudocode

```

IF CPU_FEATURES.MTRR (* processor supports MTRRs *)
  THEN
    IF BASE and SIZE are not 4-KByte aligned or size is 0
      THEN return INVALID;
    FI;
    IF (BASE + SIZE) wrap 4-GByte address space
      THEN return INVALID;
    FI;
    IF TYPE is invalid for Pentium 4, Intel Xeon, and P6 family
    processors
      THEN return UNSUPPORTED;
    FI;
    IF TYPE is WC and not supported
      THEN return UNSUPPORTED;
    FI;
    IF IA32_MTRRCAP.FIX is set AND range can be mapped using a
    fixed-range MTRR
      THEN
        pre_mtrr_change();
        update affected MTRR;
        post_mtrr_change();
      FI;

  ELSE (* try to map using a variable MTRR pair *)
    IF IA32_MTRRCAP.VCNT = 0
      THEN return UNSUPPORTED;
    FI;
    IF conflicts with current variable ranges
      THEN return RANGE_OVERLAP;
    FI;
    IF no MTRRs available
      THEN return VAR_NOT_AVAILABLE;
    FI;
    IF BASE and SIZE do not meet the power of 2 requirements for
    variable MTRRs
      THEN return INVALID_VAR_REQUEST;
    FI;
    pre_mtrr_change();
    Update affected MTRRs;
    post_mtrr_change();
  FI;

pre_mtrr_change()
  BEGIN
    disable interrupts;
    Save current value of CR4;
    disable and flush caches;
  
```

```

        flush TLBs;
        disable MTRRs;
        IF multiprocessing
            THEN maintain consistency through IPIs;
        FI;
    END
post_mtrr_change()
BEGIN
    flush caches and TLBs;
    enable MTRRs;
    enable caches;
    restore value of CR4;
    enable interrupts;
END

```

The physical address to variable range mapping algorithm in the MemTypeSet function detects conflicts with current variable range registers by cycling through them and determining whether the physical address in question matches any of the current ranges. During this scan, the algorithm can detect whether any current variable ranges overlap and can be concatenated into a single range.

The pre_mtrr_change() function disables interrupts prior to changing the MTRRs, to avoid executing code with a partially valid MTRR setup. The algorithm disables caching by setting the CD flag and clearing the NW flag in control register CR0. The caches are invalidated using the WBINVD instruction. The algorithm flushes all TLB entries either by clearing the page-global enable (PGE) flag in control register CR4 (if PGE was already set) or by updating control register CR3 (if PGE was already clear). Finally, it disables MTRRs by clearing the E flag in the IA32_MTRR_DEF_TYPE MSR.

After the memory type is updated, the post_mtrr_change() function re-enables the MTRRs and again invalidates the caches and TLBs. This second invalidation is required because of the processor's aggressive prefetch of both instructions and data. The algorithm restores interrupts and re-enables caching by setting the CD flag.

An operating system can batch multiple MTRR updates so that only a single pair of cache invalidations occur.

13.11.8 MTRR Considerations in MP Systems

In MP (multiple-processor) systems, the operating systems must maintain MTRR consistency between all the processors in the system. The Pentium 4, Intel Xeon, and P6 family processors provide no hardware support to maintain this consistency. In general, all processors must have the same MTRR values.

This requirement implies that when the operating system initializes an MP system, it must load the MTRRs of the boot processor while the E flag in register MTRRdefType is 0. The operating system then directs other processors to load their MTRRs with the same memory map. After all the processors have loaded their MTRRs, the operating system signals them to enable their MTRRs. Barrier synchronization is used to prevent further memory accesses until all processors indicate that the MTRRs are enabled. This synchronization is likely to be a shoot-down style algorithm, with shared variables and interprocessor interrupts.

Any change to the value of the MTRRs in an MP system requires the operating system to repeat the loading and enabling process to maintain consistency, using the following procedure:

1. Broadcast to all processors to execute the following code sequence.
2. Disable interrupts.
3. Wait for all processors to reach this point.
4. Enter the no-fill cache mode. (Set the CD flag in control register CR0 to 1 and the NW flag to 0.)
5. Flush all caches using the WBINVD instructions. On a processor that supports self-snooping (enumerating CPUID.01H:EDX.SELF_SNOOP[27] as 1), this step may be unnecessary. However, if there are changes for which self-snooping behavior would be problematic (e.g., changing the memory type of a cache line from WB to UC for memory-mapped I/O), execution of WBINVD would still be required.

6. If either the PGE or PCIDE flag is set in control register CR4, flush all TLBs by clearing one or both of these flags.
7. If the PGE and PCIDE flags are both clear in control register CR4, flush all TLBs by executing a MOV from control register CR3 to another register and then a MOV from that register back to CR3.
8. Disable all range registers (by clearing the E flag in register MTRRdefType). If only variable ranges are being modified, software may clear the valid bits for the affected register pairs instead.
9. Update the MTRRs.
10. Enable all range registers (by setting the E flag in register MTRRdefType). If only variable-range registers were modified and their individual valid bits were cleared, then set the valid bits for the affected ranges instead.
11. Flush all caches and all TLBs a second time. (The TLB flush is required for Pentium 4, Intel Xeon, and P6 family processors. Executing the WBINVD instruction is not needed when using Pentium 4, Intel Xeon, and P6 family processors, but it may be needed in future systems.)
12. Enter the normal cache mode to re-enable caching. (Set the CD and NW flags in control register CR0 to 0.)
13. Restore the values of the PGE and/or PCIDE flags in control register CR4, if cleared in Step 6 (above).
14. Wait for all processors to reach this point.
15. Enable interrupts.

13.11.9 Large Page Size Considerations

The MTRRs provide memory typing for a limited number of regions that have a 4 KByte granularity (the same granularity as 4-KByte pages). The memory type for a given page is cached in the processor's TLBs. When using large pages (2 MBytes, 4 MBytes, or 1 GBytes), a single page-table entry covers multiple 4-KByte granules, each with a single memory type. Because the memory type for a large page is cached in the TLB, the processor can behave in an undefined manner if a large page is mapped to a region of memory that MTRRs have mapped with multiple memory types.

Undefined behavior can be avoided by ensuring that all MTRR memory-type ranges within a large page are of the same type. If a large page maps to a region of memory containing different MTRR-defined memory types, the PCD and PWT flags in the page-table entry should be set for the most conservative memory type for that range. For example, a large page used for memory mapped I/O and regular memory is mapped as UC memory. Alternatively, the operating system can map the region using multiple 4-KByte pages each with its own memory type.

The requirement that all 4-KByte ranges in a large page are of the same memory type implies that large pages with different memory types may suffer a performance penalty, since they must be marked with the lowest common denominator memory type. The same consideration apply to 1 GByte pages, each of which may consist of multiple 2-Mbyte ranges.

The Pentium 4, Intel Xeon, and P6 family processors provide special support for the physical memory range from 0 to 4 MBytes, which is potentially mapped by both the fixed and variable MTRRs. This support is invoked when a Pentium 4, Intel Xeon, or P6 family processor detects a large page overlapping the first 1 MByte of this memory range with a memory type that conflicts with the fixed MTRRs. Here, the processor maps the memory range as multiple 4-KByte pages within the TLB. This operation ensures correct behavior at the cost of performance. To avoid this performance penalty, operating-system software should reserve the large page option for regions of memory at addresses greater than or equal to 4 MBytes.

13.12 PAGE ATTRIBUTE TABLE (PAT)

The Page Attribute Table (PAT) extends the IA-32 architecture's page-table format to allow memory types to be assigned to regions of physical memory based on linear address mappings. The PAT is a companion feature to the MTRRs; that is, the MTRRs allow mapping of memory types to regions of the physical address space, where the PAT allows mapping of memory types to pages within the linear address space. The MTRRs are useful for statically describing memory types for physical ranges, and are typically set up by the system BIOS. The PAT extends the functions of the PCD and PWT bits in page tables to allow all five of the memory types that can be assigned with the MTRRs (plus one additional memory type) to also be assigned dynamically to pages of the linear address space.

The PAT was introduced to IA-32 architecture on the Pentium III processor. It is also available in the Pentium 4 and Intel Xeon processors.

13.12.1 Detecting Support for the PAT Feature

An operating system or executive can detect the availability of the PAT by executing the CPUID instruction with a value of 1 in the EAX register. Support for the PAT is indicated by the PAT flag (bit 16 of the values returned to EDX register). If the PAT is supported, the operating system or executive can use the IA32_PAT MSR to program the PAT. When memory types have been assigned to entries in the PAT, software can then use of the PAT-index bit (PAT) in the page-table and page-directory entries along with the PCD and PWT bits to assign memory types from the PAT to individual pages.

Note that there is no separate flag or control bit in any of the control registers that enables the PAT. The PAT is always enabled on all processors that support it, and the table lookup always occurs whenever paging is enabled, in all paging modes.

13.12.2 IA32_PAT MSR

The IA32_PAT MSR is located at MSR address 277H (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4). Figure 13-9. shows the format of the 64-bit IA32_PAT MSR.

The IA32_PAT MSR contains eight page attribute fields: PA0 through PA7. The three low-order bits of each field are used to specify a memory type. The five high-order bits of each field are reserved, and must be set to all 0s. Each of the eight page attribute fields can contain any of the memory type encodings specified in Table 13-10.

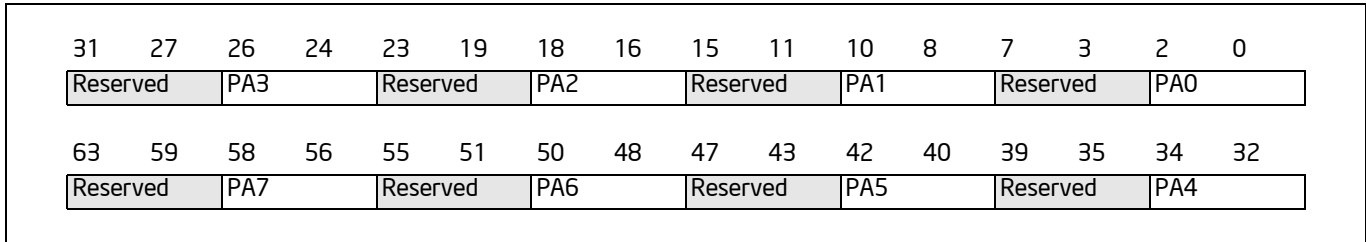


Figure 13-9. IA32_PAT MSR

Note that for the P6 family processors, the IA32_PAT MSR is named the PAT MSR.

Table 13-10. Memory Types That Can Be Encoded With PAT

Encoding	Mnemonic
00H	Uncacheable (UC)
01H	Write Combining (WC)
02H	Reserved*
03H	Reserved*
04H	Write Through (WT)
05H	Write Protected (WP)
06H	Write Back (WB)
07H	Uncached (UC-)
08H - FFH	Reserved*

NOTE:
* Using these encodings will result in a general-protection exception (#GP).

13.12.3 Selecting a Memory Type from the PAT

To select a memory type for a page from the PAT, a 3-bit index made up of the PAT, PCD, and PWT bits must be encoded in the page-table or page-directory entry for the page. Table 13-11 shows the possible encodings of the PAT, PCD, and PWT bits and the PAT entry selected with each encoding. The PAT bit is bit 7 in page-table entries that point to 4-KByte pages and bit 12 in paging-structure entries that point to larger pages. The PCD and PWT bits are bits 4 and 3, respectively, in paging-structure entries that point to pages of any size.

The PAT entry selected for a page is used in conjunction with the MTRR setting for the region of physical memory in which the page is mapped to determine the effective memory type for the page, as shown in Table 13-7.

Table 13-11. Selection of PAT Entries with PAT, PCD, and PWT Flags

PAT	PCD	PWT	PAT Entry
0	0	0	PAT0
0	0	1	PAT1
0	1	0	PAT2
0	1	1	PAT3
1	0	0	PAT4
1	0	1	PAT5
1	1	0	PAT6
1	1	1	PAT7

13.12.4 Programming the PAT

Table 13-12 shows the default setting for each PAT entry following a power up or reset of the processor. The setting remain unchanged following a soft reset (INIT reset).

Table 13-12. Memory Type Setting of PAT Entries Following a Power-up or Reset

PAT Entry	Memory Type Following Power-up or Reset
PAT0	WB
PAT1	WT
PAT2	UC-
PAT3	UC
PAT4	WB
PAT5	WT
PAT6	UC-
PAT7	UC

The values in all the entries of the PAT can be changed by writing to the IA32_PAT MSR using the WRMSR instruction. The IA32_PAT MSR is read and write accessible (use of the RDMSR and WRMSR instructions, respectively) to software operating at a CPL of 0. Table 13-10 shows the allowable encoding of the entries in the PAT. Attempting to write an undefined memory type encoding into the PAT causes a general-protection (#GP) exception to be generated.

The operating system (OS) is responsible for ensuring that changes to a PAT entry occur in a manner that maintains the consistency of the processor caches and translation lookaside buffers (TLB). It requires the OS to invalidate all affected TLB entries (including global entries) and all entries in all paging-structure caches. It may also require flushing of the processor caches in certain situations. This can be accomplished in various ways, including the sequence below or by following the procedure specified in Section 13.11.8, "MTRR Considerations in MP Systems." (See Section 5.10.4, "Invalidation of TLBs and Paging-Structure Caches" for additional background information.) Also note that in a multi-processor environment, it is the software's responsibility to resolve differences in conflicting memory types across logical processors that may arise from changes to the PAT (e.g., if two

logical processors map a linear address to the same physical address but have PATs that specify a different memory type for that physical address).

Example of a sequence to invalidate the processor TLBs and caches (if necessary):

1. If the PCIDE or PGE flag is set in CR4, flush TLBs by clearing one of those flags (then restore the flag via a subsequent CR4 write).
Otherwise, flush TLBs by executing a MOV from control register CR3 to another register and then a MOV from that register back to CR3.
2. In the case that there are changes to memory-type mappings for which cache self-snooping behavior would be problematic given the existing mappings (e.g., changing a cache line's memory type from WB to UC to be used for memory-mapped I/O), then cache flushing is also required. This can be done by executing CLFLUSH operations for all affected cache lines or by executing the WBINVD instruction (recommended only if there are a large number of affected mappings or if it is unknown which mappings are affected).

The PAT allows any memory type to be specified in the page tables, and therefore it is possible to have a single physical page mapped to two or more different linear addresses, each with different memory types. Intel does not support this practice because it may lead to undefined operations that can result in a system failure. In particular, a WC page must never be aliased to a cacheable page because WC writes may not check the processor caches.

When remapping a page that was previously mapped as a cacheable memory type to a WC page, an operating system can avoid this type of aliasing by doing the following:

1. Remove the previous mapping to a cacheable memory type in the page tables; that is, make them not present.
2. Flush the TLBs of processors that may have used the mapping, even speculatively.
3. Create a new mapping to the same physical address with a new memory type, for instance, WC.
4. Flush the caches on all processors that may have used the mapping previously. Note on processors that support self-snooping, CPUID feature flag bit 27, this step is unnecessary.

Operating systems that use a page directory as a page table (to map large pages) and enable page size extensions must carefully scrutinize the use of the PAT index bit for the 4-KByte page-table entries. The PAT index bit for a page-table entry (bit 7) corresponds to the page size bit in a page-directory entry. Therefore, the operating system can only use PAT entries PA0 through PA3 when setting the caching type for a page table that is also used as a page directory. If the operating system attempts to use PAT entries PA4 through PA7 when using this memory as a page table, it effectively sets the PS bit for the access to this memory as a page directory.

For compatibility with earlier IA-32 processors that do not support the PAT, care should be taken in selecting the encodings for entries in the PAT (see Section 13.12.5, "PAT Compatibility with Earlier IA-32 Processors").

13.12.5 PAT Compatibility with Earlier IA-32 Processors

For IA-32 processors that support the PAT, the IA32_PAT MSR is always active. That is, the PCD and PWT bits in page-table entries and in page-directory entries (that point to pages) are always select a memory type for a page indirectly by selecting an entry in the PAT. They never select the memory type for a page directly as they do in earlier IA-32 processors that do not implement the PAT (see Table 13-6).

To allow compatibility for code written to run on earlier IA-32 processor that do not support the PAT, the PAT mechanism has been designed to allow backward compatibility to earlier processors. This compatibility is provided through the ordering of the PAT, PCD, and PWT bits in the 3-bit PAT entry index. For processors that do not implement the PAT, the PAT index bit (bit 7 in the page-table entries and bit 12 in the page-directory entries) is reserved and set to 0. With the PAT bit reserved, only the first four entries of the PAT can be selected with the PCD and PWT bits. At power-up or reset (see Table 13-12), these first four entries are encoded to select the same memory types as the PCD and PWT bits would normally select directly in an IA-32 processor that does not implement the PAT. So, if encodings of the first four entries in the PAT are left unchanged following a power-up or reset, code written to run on earlier IA-32 processors that do not implement the PAT will run correctly on IA-32 processors that do implement the PAT.

15. Updates to Chapter 16, Volume 3B

Change bars and violet text show changes to Chapter 16 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Updated footnote for bits 63:MAXPHYADDR in Section 16.6.4, "Hardware Feedback Interface Pointer".

This chapter describes facilities of Intel 64 and IA-32 architecture used for power management and thermal monitoring.

16.1 ENHANCED INTEL SPEEDSTEP® TECHNOLOGY

Enhanced Intel SpeedStep® Technology was introduced in the Pentium M processor. The technology enables the management of processor power consumption via performance state transitions. These states are defined as discrete operating points associated with different voltages and frequencies.

Enhanced Intel SpeedStep Technology differs from previous generations of Intel SpeedStep® Technology in two ways:

- Centralization of the control mechanism and software interface in the processor by using model-specific registers.
- Reduced hardware overhead; this permits more frequent performance state transitions.

Previous generations of the Intel SpeedStep Technology require processors to be in a deep sleep state, holding off bus master transfers for the duration of a performance state transition. Performance state transitions under the Enhanced Intel SpeedStep Technology are discrete transitions to a new target frequency.

Support is indicated by CPUID, using ECX feature bit 07. Enhanced Intel SpeedStep Technology is enabled by setting IA32_MISC_ENABLE MSR, bit 16. On reset, bit 16 of IA32_MISC_ENABLE MSR is cleared.

16.1.1 Software Interface For Initiating Performance State Transitions

State transitions are initiated by writing a 16-bit value to the IA32_PERF_CTL register, see Figure 16-2. If a transition is already in progress, transition to a new value will subsequently take effect.

Reads of IA32_PERF_CTL determine the last targeted operating point. The current operating point can be read from IA32_PERF_STATUS. IA32_PERF_STATUS is updated dynamically.

The 16-bit encoding that defines valid operating points is model-specific. Applications and performance tools are not expected to use either IA32_PERF_CTL or IA32_PERF_STATUS and should treat both as reserved. Performance monitoring tools can access model-specific events and report the occurrences of state transitions.

16.2 P-STATE HARDWARE COORDINATION

The Advanced Configuration and Power Interface (ACPI) defines performance states (P-states) that are used to facilitate system software's ability to manage processor power consumption. Different P-states correspond to different performance levels that are applied while the processor is actively executing instructions. Enhanced Intel SpeedStep Technology supports P-states by providing software interfaces that control the operating frequency and voltage of a processor.

With multiple processor cores residing in the same physical package, hardware dependencies may exist for a subset of logical processors on a platform. These dependencies may impose requirements that impact the coordination of P-state transitions. As a result, multi-core processors may require an OS to provide additional software support for coordinating P-state transitions for those subsets of logical processors.

ACPI firmware can choose to expose P-states as dependent and hardware-coordinated to OS power management (OSPM) policy. To support OSPMs, multi-core processors must have additional built-in support for P-state hardware coordination and feedback.

Intel 64 and IA-32 processors with dependent P-states amongst a subset of logical processors permit hardware coordination of P-states and provide a hardware-coordination feedback mechanism using IA32_MPERF MSR and

IA32_APERF MSR. See Figure 16-1 for an overview of the two 64-bit MSRs and the bullets below for a detailed description.

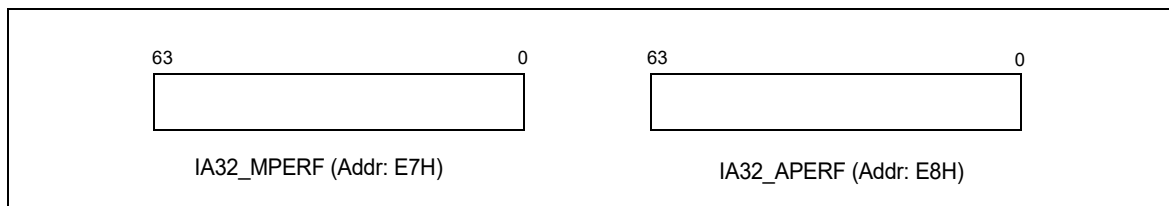


Figure 16-1. IA32_MPERF MSR and IA32_APERF MSR for P-state Coordination

- Use CPUID to check the P-State hardware coordination feedback capability bit. CPUID.06H:ECX[0] = 1 indicates IA32_MPERF MSR and IA32_APERF MSR are present.
- The IA32_MPERF MSR (E7H) increments in proportion to a fixed frequency, which is configured when the processor is booted.
- The IA32_APERF MSR (E8H) increments in proportion to actual performance, while accounting for hardware coordination of P-state and TM1/TM2; or software initiated throttling.
- The MSRs are per logical processor; they measure performance only when the targeted processor is in the C0 state.
- Only the IA32_APERF/IA32_MPERF ratio is architecturally defined; software should not attach meaning to the content of the individual bits of the IA32_APERF or IA32_MPERF MSRs.
- When either MSR overflows, both MSRs are reset to zero and continue to increment.
- Both MSRs are full 64-bits counters. Each MSR can be written to independently. However, software should follow the guidelines illustrated in Example 16-1.

If P-states are exposed by the BIOS as hardware coordinated, software is expected to confirm processor support for P-state hardware coordination feedback and use the feedback mechanism to make P-state decisions. The OSPM is expected to either save away the current MSR values (for determination of the delta of the counter ratio at a later time) or reset both MSRs (execute WRMSR with 0 to these MSRs individually) at the start of the time window used for making the P-state decision. When not resetting the values, overflow of the MSRs can be detected by checking whether the new values read are less than the previously saved values.

Example 16-1 demonstrates steps for using the hardware feedback mechanism provided by IA32_APERF MSR and IA32_MPERF MSR to determine a target P-state.

Example 16-1. Determine Target P-state From Hardware Coordinated Feedback

```

DWORD PercentBusy; // Percentage of processor time not idle.
// Measure "PercentBusy" during previous sampling window.
// Typically, "PercentBusy" is measure over a time scale suitable for
// power management decisions
//
// RDMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between
// the two RDMSRs (for example, interrupts).
MCNT = RDMSR(IA32_MPERF);
ACNT = RDMSR(IA32_APERF);

// PercentPerformance indicates the percentage of the processor
// that is in use. The calculation is based on the PercentBusy,
// that is the percentage of processor time not idle and the P-state
// hardware coordinated feedback using the ACNT/MCNT ratio.
// Note that both values need to be calculated over the same

```

```

// time window.
PercentPerformance = PercentBusy * (ACNT/MCNT);

// This example does not cover the additional logic or algorithms
// necessary to coordinate multiple logical processors to a target P-state.

TargetPstate = FindPstate(PercentPerformance);

if (TargetPstate != currentPstate) {
    SetPstate(TargetPstate);
}
// WRMSR of MCNT and ACNT should be performed without delay.
// Software needs to exercise care to avoid delays between the two WRMSRs (for example, interrupts).
WRMSR(IA32_MPERF, 0);
WRMSR(IA32_APERF, 0);

```

16.3 SYSTEM SOFTWARE CONSIDERATIONS AND OPPORTUNISTIC PROCESSOR PERFORMANCE OPERATION

An Intel 64 processor may support a form of processor operation that takes advantage of design headroom to opportunistically increase performance. The Intel® Turbo Boost Technology can convert thermal headroom into higher performance across multi-threaded and single-threaded workloads. The Intel® Dynamic Acceleration Technology feature can convert thermal headroom into higher performance if only one thread is active.

16.3.1 Intel® Dynamic Acceleration Technology

The Intel Core 2 Duo processor T7700 introduces Intel Dynamic Acceleration Technology. Intel Dynamic Acceleration Technology takes advantage of thermal design headroom and opportunistically allows a single core to operate at a higher performance level when the operating system requests increased performance.

16.3.2 System Software Interfaces for Opportunistic Processor Performance Operation

Opportunistic processor performance operation, applicable to Intel Dynamic Acceleration Technology and Intel® Turbo Boost Technology, has the following characteristics:

- A transition from a normal state of operation (e.g., Intel Dynamic Acceleration Technology/Turbo mode disengaged) to a target state is not guaranteed, but may occur opportunistically after the corresponding enable mechanism is activated, the headroom is available and certain criteria are met.
- The opportunistic processor performance operation is generally transparent to most application software.
- System software (BIOS and Operating system) must be aware of hardware support for opportunistic processor performance operation and may need to temporarily disengage opportunistic processor performance operation when it requires more predictable processor operation.
- When opportunistic processor performance operation is engaged, the OS should use hardware coordination feedback mechanisms to prevent un-intended policy effects if it is activated during inappropriate situations.

16.3.2.1 Discover Hardware Support and Enabling of Opportunistic Processor Performance Operation

If an Intel 64 processor has hardware support for opportunistic processor performance operation, the power-on default state of IA32_MISC_ENABLE[38] indicates the presence of such hardware support. For Intel 64 processors that support opportunistic processor performance operation, the default value is 1, indicating its presence. For processors that do not support opportunistic processor performance operation, the default value is 0. The power-

on default value of IA32_MISC_ENABLE[38] allows BIOS to detect the presence of hardware support of opportunistic processor performance operation.

IA32_MISC_ENABLE[38] is shared across all logical processors in a physical package. It is written by BIOS during platform initiation to enable/disable opportunistic processor performance operation in conjunction of OS power management capabilities, see Section 16.3.2.2. BIOS can set IA32_MISC_ENABLE[38] with 1 to disable opportunistic processor performance operation; it must clear the default value of IA32_MISC_ENABLE[38] to 0 to enable opportunistic processor performance operation. OS and applications must use CPUID.06H if it needs to detect processors that have opportunistic processor performance operation enabled.

When CPUID is executed with EAX = 06H on input, bit 1 of EAX in Leaf 06H (i.e., CPUID.06H:EAX[1]) indicates opportunistic processor performance operation, such as Intel Dynamic Acceleration Technology, has been enabled by BIOS.

Opportunistic processor performance operation can be disabled by setting bit 38 of IA32_MISC_ENABLE. This mechanism is intended for BIOS only. If IA32_MISC_ENABLE[38] is set, CPUID.06H:EAX[1] will return 0.

16.3.2.2 OS Control of Opportunistic Processor Performance Operation

There may be phases of software execution in which system software cannot tolerate the non-deterministic aspects of opportunistic processor performance operation. For example, when calibrating a real-time workload to make a CPU reservation request to the OS, it may be undesirable to allow the possibility of the processor delivering increased performance that cannot be sustained after the calibration phase.

System software can temporarily disengage opportunistic processor performance operation by setting bit 32 of the IA32_PERF_CTL MSR (0199H), using a read-modify-write sequence on the MSR. The opportunistic processor performance operation can be re-engaged by clearing bit 32 in IA32_PERF_CTL MSR, using a read-modify-write sequence. The DISENGAGE bit in IA32_PERF_CTL is not reflected in bit 32 of the IA32_PERF_STATUS MSR (0198H), and it is not shared between logical processors in a physical package. In order for OS to engage Intel Dynamic Acceleration Technology/Turbo mode, the BIOS must:

- Enable opportunistic processor performance operation, as described in Section 16.3.2.1.
- Expose the operating points associated with Intel Dynamic Acceleration Technology/Turbo mode to the OS.

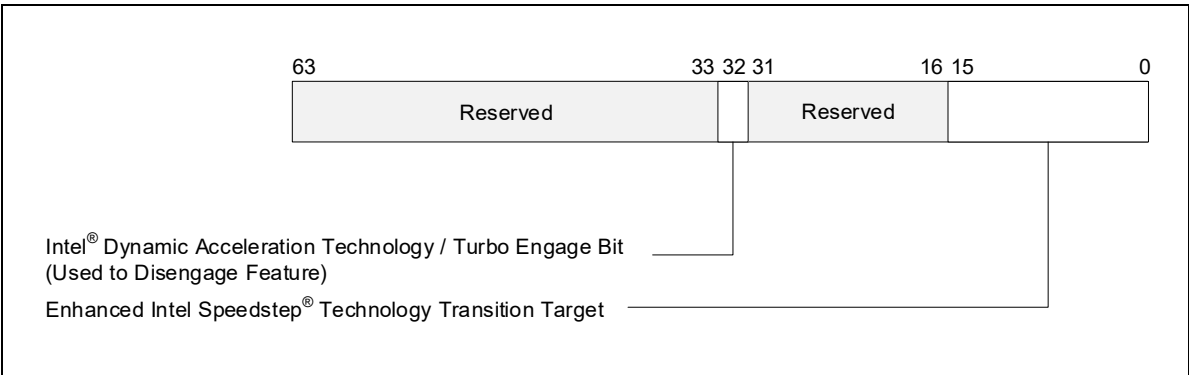


Figure 16-2. IA32_PERF_CTL Register

16.3.2.3 Required Changes to OS Power Management P-State Policy

Intel Dynamic Acceleration Technology and Intel Turbo Boost Technology can provide opportunistic performance greater than the performance level corresponding to the Processor Base frequency of the processor (see CPUID’s processor frequency information). System software can use a pair of MSRs to observe performance feedback. Software must query for the presence of IA32_APERF and IA32_MPERF (see Section 16.2). The ratio between IA32_APERF and IA32_MPERF is architecturally defined and a value greater than unity indicates performance increase occurred during the observation period due to Intel Dynamic Acceleration Technology. Without incorporating such performance feedback, the target P-state evaluation algorithm can result in a non-optimal P-state target.

There are other scenarios under which OS power management may want to disable Intel Dynamic Acceleration Technology, some of these are listed below:

- When engaging ACPI defined passive thermal management, it may be more effective to disable Intel Dynamic Acceleration Technology for the duration of passive thermal management.
- When the user has indicated a policy preference of power savings over performance, OS power management may want to disable Intel Dynamic Acceleration Technology while that policy is in effect.

16.3.3 Intel® Turbo Boost Technology

Intel Turbo Boost Technology is supported in Intel Core i7 processors and Intel Xeon processors based on Nehalem microarchitecture. It uses the same principle of leveraging thermal headroom to dynamically increase processor performance for single-threaded and multi-threaded/multi-tasking environment. The programming interface described in Section 16.3.2 also applies to Intel Turbo Boost Technology.

16.3.4 Performance and Energy Bias Hint Support

Intel 64 processors may support additional software hint to guide the hardware heuristic of power management features to favor increasing dynamic performance or conserve energy consumption.

Software can detect the processor's capability to support the performance-energy bias preference hint by examining bit 3 of ECX in CPUID.06H. The processor supports this capability if CPUID.06H:ECX.ENERGY_PERF_BIAS[3] is set and it also implies the presence of a new architectural MSR called IA32_ENERGY_PERF_BIAS (1B0H).

Software can program the lowest four bits of IA32_ENERGY_PERF_BIAS MSR with a value from 0-15. The values represent a sliding scale, where a value of 0 (the default reset value) corresponds to a hint preference for highest performance and a value of 15 corresponds to the maximum energy savings. A value of 7 roughly translates into a hint to balance performance with energy consumption.

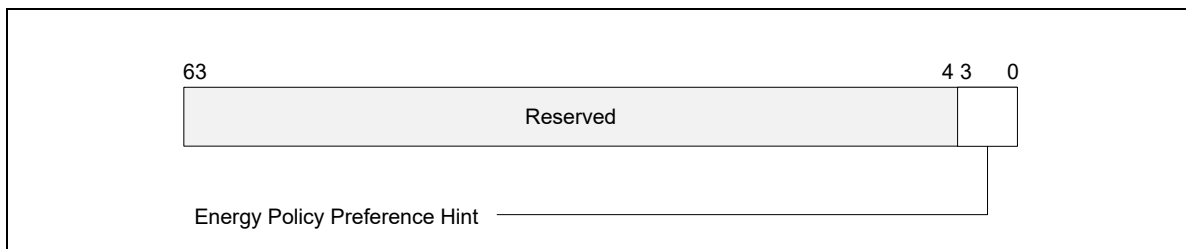


Figure 16-3. IA32_ENERGY_PERF_BIAS Register

The layout of IA32_ENERGY_PERF_BIAS is shown in Figure 16-3. The scope of IA32_ENERGY_PERF_BIAS is per logical processor, which means that each of the logical processors in the package can be programmed with a different value. This may be especially important in virtualization scenarios, where the performance / energy requirements of one logical processor may differ from the other. Conflicting “hints” from various logical processors at higher hierarchy level will be resolved in favor of performance over energy savings.

Software can use whatever criteria it sees fit to program the MSR with an appropriate value. However, the value only serves as a hint to the hardware and the actual impact on performance and energy savings is model specific.

16.4 HARDWARE-CONTROLLED PERFORMANCE STATES (HWP)

Intel processors may contain support for Hardware-Controlled Performance States (HWP), which autonomously selects performance states while utilizing OS supplied performance guidance hints. The Enhanced Intel Speed-Step® Technology provides a means for the OS to control and monitor discrete frequency-based operating points via the IA32_PERF_CTL and IA32_PERF_STATUS MSRs.

In contrast, HWP is an implementation of the ACPI-defined Collaborative Processor Performance Control (CPPC), which specifies that the platform enumerates a continuous, abstract unit-less, performance value scale that is not tied to a specific performance state / frequency by definition. While the enumerated scale is roughly linear in terms of a delivered integer workload performance result, the OS is required to characterize the performance value range to comprehend the delivered performance for an applied workload.

When HWP is enabled, the processor autonomously selects performance states as deemed appropriate for the applied workload and with consideration of constraining hints that are programmed by the OS. These OS-provided hints include minimum and maximum performance limits, preference towards energy efficiency or performance, and the specification of a relevant workload history observation time window. The means for the OS to override HWP's autonomous selection of performance state with a specific desired performance target is also provided, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations.

16.4.1 HWP Programming Interfaces

The programming interfaces provided by HWP include the following:

- The CPUID instruction allows software to discover the presence of HWP support in an Intel processor. Specifically, execute CPUID.06H will return 5 bit flags covering the following aspects in bits 7 through 11 of CPUID.06H:EAX:
 - Availability of HWP baseline resource and capability, CPUID.06H:EAX[7]: If this bit is set, HWP provides several new architectural MSRs: IA32_PM_ENABLE, IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS.
 - Availability of HWP Notification upon dynamic Guaranteed Performance change, CPUID.06H:EAX[8]: If this bit is set, HWP provides IA32_HWP_INTERRUPT MSR to enable interrupt generation due to dynamic Performance changes and excursions.
 - Availability of HWP Activity window control, CPUID.06H:EAX[9]: If this bit is set, HWP allows software to program activity window in the IA32_HWP_REQUEST MSR.
 - Availability of HWP energy/performance preference control, CPUID.06H:EAX[10]: If this bit is set, HWP allows software to set an energy/performance preference hint in the IA32_HWP_REQUEST MSR.
 - Availability of HWP package level control, CPUID.06H:EAX[11]: If this bit is set, HWP provides the IA32_HWP_REQUEST_PKG MSR to convey OS Power Management's control hints for all logical processors in the physical package.

Table 16-1. Architectural and Non-Architectural MSRs Related to HWP

Address	Architectural	Register Name	Description
770H	Y	IA32_PM_ENABLE	Enable/Disable HWP.
771H	Y	IA32_HWP_CAPABILITIES	Enumerates the HWP performance range (static and dynamic).
772H	Y	IA32_HWP_REQUEST_PKG	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for all logical processor in the physical package.
773H	Y	IA32_HWP_INTERRUPT	Controls HWP native interrupt generation (Guaranteed Performance changes, excursions).
774H	Y	IA32_HWP_REQUEST	Conveys OSPM's control hints (Min, Max, Activity Window, Energy Performance Preference, Desired) for a single logical processor.
775H	Y	IA32_HWP_PECI_REQUEST_INFO	Conveys embedded system controller requests to override some of the OS HWP Request settings via the Peci mechanism.
777H	Y	IA32_HWP_STATUS	Status bits indicating changes to Guaranteed Performance and excursions to Minimum Performance.
19CH	Y	IA32_THERM_STATUS[bits 15:12]	Conveys reasons for performance excursions.
64EH	N	MSR_PPERF	Productive Performance Count.

- Additionally, HWP may provide a non-architectural MSR, MSR_PPERF, which provides a quantitative metric to software of hardware's view of workload scalability. This hardware's view of workload scalability is implementation specific.

16.4.2 Enabling HWP

The layout of the IA32_PM_ENABLE MSR is shown in Figure 16-4. The bit fields are described below:

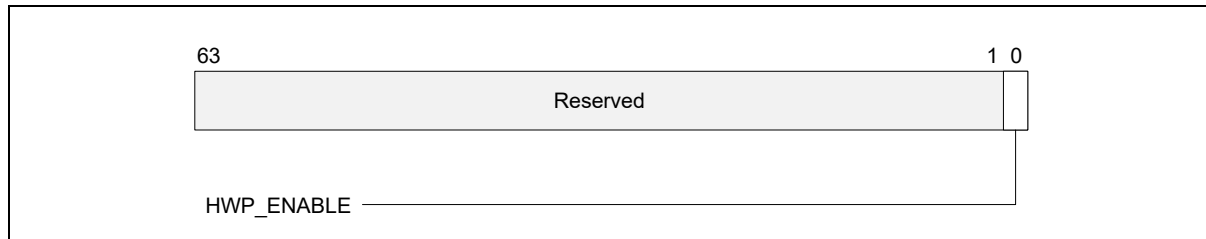


Figure 16-4. IA32_PM_ENABLE MSR

- **HWP_ENABLE (bit 0, R/W1Once)** — Software sets this bit to enable HWP with autonomous selection of processor P-States. When set, the processor will disregard input from the legacy performance control interface (IA32_PERF_CTL). Note that this bit can only be enabled once from the default value. Once set, writes to the HWP_ENABLE bit are ignored. Only RESET will clear this bit. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After software queries CPUID and verifies the processor's support of HWP, system software can write 1 to IA32_PM_ENABLE.HWP_ENABLE (bit 0) to enable hardware controlled performance states. The default value of IA32_PM_ENABLE MSR at power-on is 0, i.e., HWP is disabled.

Additional MSRs associated with HWP may only be accessed after HWP is enabled, with the exception of IA32_HWP_INTERRUPT and MSR_PPERF. Accessing the IA32_HWP_INTERRUPT MSR requires only HWP is present as enumerated by CPUID but does not require enabling HWP.

IA32_PM_ENABLE is a package level MSR, i.e., writing to it from any logical processor within a package affects all logical processors within that package.

16.4.3 HWP Performance Range and Dynamic Capabilities

The OS reads the IA32_HWP_CAPABILITIES MSR to comprehend the limits of the HWP-managed performance range as well as the dynamic capability, which may change during processor operation. The enumerated performance range values reported by IA32_HWP_CAPABILITIES directly map to initial frequency targets (prior to workload-specific frequency optimizations of HWP). However the mapping is processor family specific.

The layout of the IA32_HWP_CAPABILITIES MSR is shown in Figure 16-5. The bit fields are described below this figure.

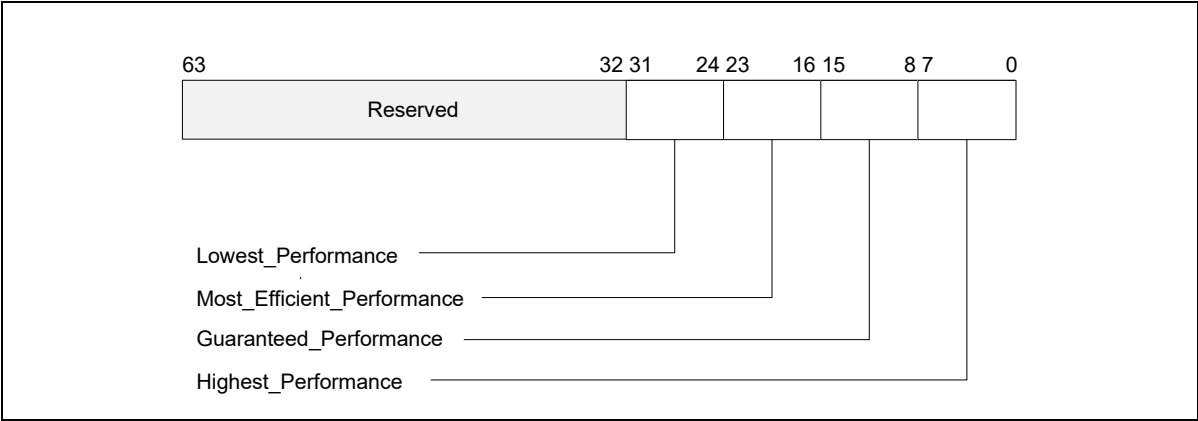


Figure 16-5. IA32_HWP_CAPABILITIES Register

- **Highest_Performance (bits 7:0, RO)** — Value for the maximum non-guaranteed performance level.
- **Guaranteed_Performance (bits 15:8, RO)** — Current value for the guaranteed performance level. This value can change dynamically as a result of internal or external constraints, e.g., thermal or power limits.
- **Most_Efficient_Performance (bits 23:16, RO)** — Current value of the most efficient performance level. This value can change dynamically as a result of workload characteristics.
- **Lowest_Performance (bits 31:24, RO)** — Value for the lowest performance level that software can program to IA32_HWP_REQUEST.
- Bits 63:32 are reserved and must be zero.

The value returned in the **Guaranteed_Performance** field is hardware's best-effort approximation of the available performance given current operating constraints. Changes to the Guaranteed_Performance value will primarily occur due to a shift in operational mode. This includes a power or other limit applied by an external agent, e.g., RAPL (see Figure 16.10.1), or the setting of a Configurable TDP level (see model-specific controls related to Programmable TDP Limit in Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.). Notification of a change to the Guaranteed_Performance occurs via interrupt (if configured) and the IA32_HWP_Status MSR. Changes to Guaranteed_Performance are indicated when a macroscopically meaningful change in performance occurs i.e., sustained for greater than one second. Consequently, notification of a change in Guaranteed Performance will typically occur no more frequently than once per second. Rapid changes in platform configuration, e.g., docking/undocking, with corresponding changes to a Configurable TDP level could potentially cause more frequent notifications.

The value returned by the **Most_Efficient_Performance** field provides the OS with an indication of the practical lower limit for the IA32_HWP_REQUEST. The processor may not honor IA32_HWP_REQUEST.Maximum Performance settings below this value.

16.4.4 Managing HWP

16.4.4.1 IA32_HWP_REQUEST MSR (Address: 774H Logical Processor Scope)

Typically, the operating system controls HWP operation for each logical processor via the writing of control hints / constraints to the IA32_HWP_REQUEST MSR. The layout of the IA32_HWP_REQUEST MSR is shown in Figure 16-6. The bit fields are described below Figure 16-6.

Operating systems can control HWP by writing both IA32_HWP_REQUEST and IA32_HWP_REQUEST_PKG MSRs (see Section 16.4.4.2). Five valid bits within the IA32_HWP_REQUEST MSR let the operating system flexibly select which of its five hint / constraint fields should be derived by the processor from the IA32_HWP_REQUEST MSR and which should be derived from the IA32_HWP_REQUEST_PKG MSR. These five valid bits are supported if CPUID.06H:EAX[17] is set.

When the IA32_HWP_REQUEST MSR Package Control bit is set, any valid bit that is NOT set indicates to the processor to use the respective field value from the IA32_HWP_REQUEST_PKG MSR. Otherwise, the values are derived from the IA32_HWP_REQUEST MSR. The valid bits are ignored when the IA32_HWP_REQUEST MSR Package Control bit is zero.

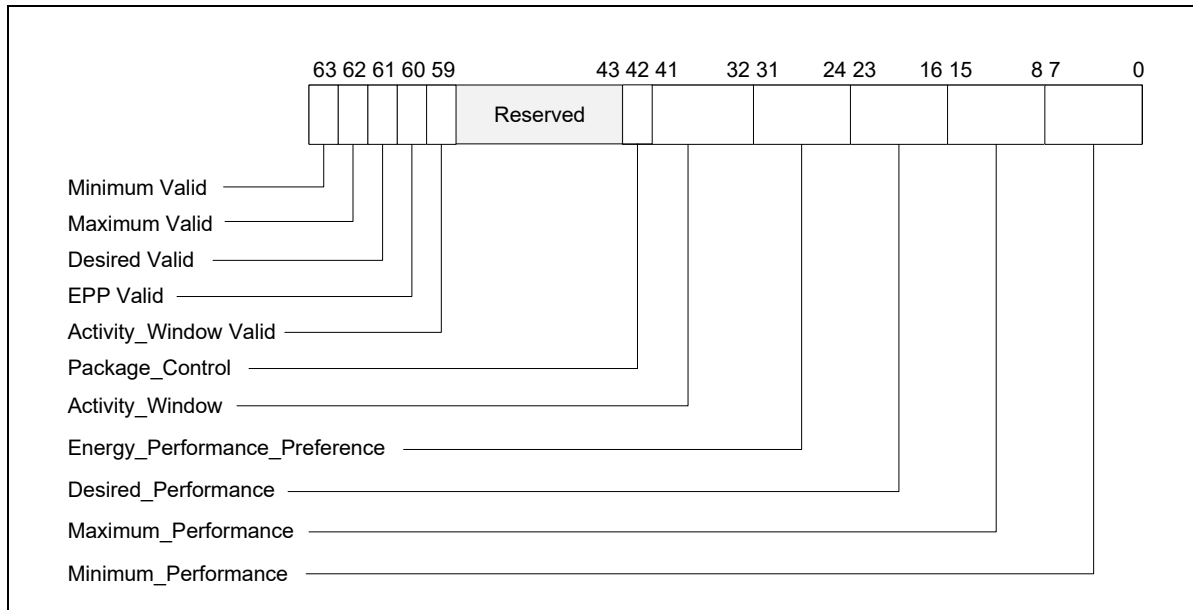


Figure 16-6. IA32_HWP_REQUEST Register

- **Minimum_Performance (bits 7:0, RW)** — Conveys a hint to the HWP hardware. The OS programs the minimum performance hint to achieve the required quality of service (QOS) or to meet a service level agreement (SLA) as needed. Note that an excursion below the level specified is possible due to hardware constraints. The default value of this field is IA32_HWP_CAPABILITIES.Lowest_Performance.
- **Maximum_Performance (bits 15:8, RW)** — Conveys a hint to the HWP hardware. The OS programs this field to limit the maximum performance that is expected to be supplied by the HWP hardware. Excursions above the limit requested by OS are possible due to hardware coordination between the processor cores and other components in the package. The default value of this field is IA32_HWP_CAPABILITIES.Highest_Performance.
- **Desired_Performance (bits 23:16, RW)** — Conveys a hint to the HWP hardware. When set to zero, hardware autonomous selection determines the performance target. When set to a non-zero value (between the range of Lowest_Performance and Highest_Performance of IA32_HWP_CAPABILITIES) conveys an explicit performance request hint to the hardware; effectively disabling HW Autonomous selection. The Desired_Performance input is non-constraining in terms of Performance and Energy Efficiency optimizations, which are independently controlled. The default value of this field is 0.
- **Energy_Performance_Preference (bits 31:24, RW)** — Conveys a hint to the HWP hardware. The OS may write a range of values from 0 (performance preference) to 0FFH (energy efficiency preference) to influence the rate of performance increase /decrease and the result of the hardware's energy efficiency and performance optimizations. The default value of this field is 80H. Note: If CPUID.06H:EAX[10] indicates that this field is not supported, HWP uses the value of the IA32_ENERGY_PERF_BIAS MSR to determine the energy efficiency / performance preference.
- **Activity_Window (bits 41:32, RW)** — Conveys a hint to the HWP hardware specifying a moving workload history observation window for performance/frequency optimizations. If 0, the hardware will determine the appropriate window size. When writing a non-zero value to this field, this field is encoded in the format of bits 38:32 as a 7-bit mantissa and bits 41:39 as a 3-bit exponent value in powers of 10. The resultant value is in microseconds. Thus, the minimal/maximal activity window size is 1 microsecond/1270 seconds. Combined with the Energy_Performance_Preference input, Activity_Window influences the rate of performance increase

/ decrease. This non-zero hint only has meaning when Desired_Performance = 0. The default value of this field is 0.

- **Package_Control (bit 42, RW)** — When set, causes this logical processor's IA32_HWP_REQUEST control inputs to be derived from the IA32_HWP_REQUEST_PKG MSR.
- Bits 58:43 are reserved and must be zero.
- **Activity_Window_Valid (bit 59, RW)** — When set, indicates to the processor to derive the Activity Window field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **EPP_Valid (bit 60, RW)** — When set, indicates to the processor to derive the EPP field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Desired_Valid (bit 61, RW)** — When set, indicates to the processor to derive the Desired Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Maximum_Valid (bit 62, RW)** — When set, indicates to the processor to derive the Maximum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.
- **Minimum_Valid (bit 63, RW)** — When set, indicates to the processor to derive the Minimum Performance field value from the IA32_HWP_REQUEST MSR even if the package control bit is set. Otherwise, derive it from the IA32_HWP_REQUEST_PKG MSR. The default value of this field is 0.

The HWP hardware clips and resolves the field values as necessary to the valid range. Reads return the last value written not the clipped values.

Processors may support a subset of IA32_HWP_REQUEST fields as indicated by CPUID. Reads of non-supported fields will return 0. Writes to non-supported fields are ignored.

The OS may override HWP's autonomous selection of performance state with a specific performance target by setting the Desired_Performance field to a non-zero value, however, the effective frequency delivered is subject to the result of energy efficiency and performance optimizations, which are influenced by the Energy Performance Preference field.

Software may disable all hardware optimizations by setting Minimum_Performance = Maximum_Performance (subject to package coordination).

Note: The processor may run below the Minimum_Performance level due to hardware constraints including: power, thermal, and package coordination constraints. The processor may also run below the Minimum_Performance level for short durations (few milliseconds) following C-state exit, and when Hardware Duty Cycling (see Section 16.5) is enabled.

When the IA32_HWP_REQUEST MSR is set to fast access mode, writes of this MSR are posted, i.e., the WRMSR instruction retires before the data reaches its destination within the processor. It may retire even before all preceding IA stores are globally visible, i.e., it is not an architecturally serializing instruction anymore (no store fence). A new CPUID bit indicates this new characteristic of the IA32_HWP_REQUEST MSR (see Section 16.4.8 for additional details).

16.4.4.2 IA32_HWP_REQUEST_PKG MSR (Address: 772H Package Scope)

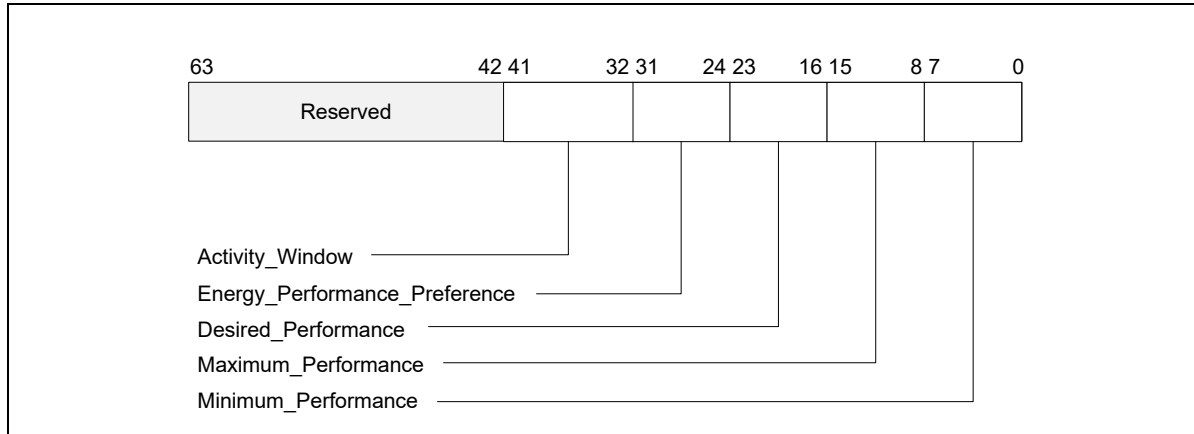


Figure 16-7. IA32_HWP_REQUEST_PKG Register

The structure of the IA32_HWP_REQUEST_PKG MSR (package-level) is identical to the IA32_HWP_REQUEST MSR with the exception of the the Package Control bit field and the five valid bit fields, which do not exist in the IA32_HWP_REQUEST_PKG MSR. Field values written to this MSR apply to all logical processors within the physical package with the exception of logical processors whose IA32_HWP_REQUEST.Package Control field is clear (zero). Single P-state Control mode is only supported when IA32_HWP_REQUEST_PKG is not supported.

16.4.4.3 IA32_HWP_PECI_REQUEST_INFO MSR (Address 775H Package Scope)

When an embedded system controller is integrated in the platform, it can override some of the OS HWP Request settings via the Peci mechanism. Peci initiated settings take precedence over the relevant fields in the IA32_HWP_REQUEST MSR and in the IA32_HWP_REQUEST_PKG MSR, irrespective of the Package Control bit or the Valid Bit values described above. Peci can independently control each of: Minimum Performance, Maximum Performance, and EPP fields. This MSR contains both the Peci induced values and the control bits that indicate whether the embedded controller actually set the processor to use the respective value.

PECI override is supported if CPUID.06H:EAX[bit 16] is set.

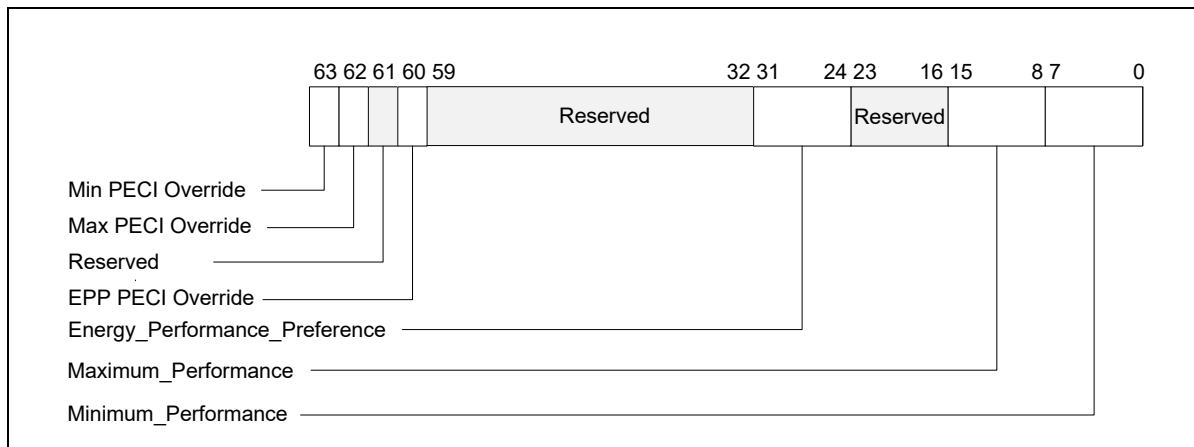


Figure 16-8. IA32_HWP_PECI_REQUEST_INFO MSR

The layout of the IA32_HWP_PECI_REQUEST_INFO MSR is shown in Figure 16-8. This MSR is writable by the embedded controller but is read-only by software executing on the CPU. This MSR has Package scope. The bit fields are described below:

- **Minimum_Performance (bits 7:0, RO)** — Used by the OS to read the latest value of Peci minimum performance input.
- **Maximum_Performance (bits 15:8, RO)** — Used by the OS to read the latest value of Peci maximum performance input.
- Bits 23:16 are reserved and must be zero.
- **Energy_Performance_Preference (bits 31:24, RO)** — Used by the OS to read the latest value of Peci energy performance preference input.
- Bits 59:32 are reserved and must be zero.
- **EPP_PECI_Override (bit 60, RO)** — Indicates whether Peci is currently overriding the Energy Performance Preference input. If set(1), Peci is overriding the Energy Performance Preference input. If clear(0), OS has control over Energy Performance Preference input.
- Bit 61 is reserved and must be zero.
- **Max_PECI_Override (bit 62, RO)** — Indicates whether Peci is currently overriding the Maximum Performance input. If set(1), Peci is overriding the Maximum Performance input. If clear(0), OS has control over Maximum Performance input.
- **Min_PECI_Override (bit 63, RO)** — Indicates whether Peci is currently overriding the Minimum Performance input. If set(1), Peci is overriding the Minimum Performance input. If clear(0), OS has control over Minimum Performance input.

HWP Request Field Hierarchical Resolution

HWP Request field resolution is fed by three MSRs: IA32_HWP_REQUEST, IA32_HWP_REQUEST_PKG, and IA32_HWP_PECI_REQUEST_INFO. The flow that the processor goes through to resolve which field value is chosen is shown below.

For each of the two HWP Request fields; Desired and Activity Window:

```
If IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
    Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
Else
    Resolved Field Value = IA32_HWP_REQUEST.<field>
```

For each of the three HWP Request fields; Min, Max, and EPP:

```
If IA32_HWP_PECI_REQUEST_INFO.<field> Peci Override bit = 1
    Resolved Field Value = IA32_HWP_PECI_REQUEST_INFO.<field>
Else if IA32_HWP_REQUEST.PACKAGE_CONTROL = 1 and IA32_HWP_REQUEST.<field> valid bit = 0
    Resolved Field Value = IA32_HWP_REQUEST_PKG.<field>
Else
    Resolved Field Value = IA32_HWP_REQUEST.<field>
```

16.4.4.4 IA32_HWP_CTL MSR (Address: 776H Logical Processor Scope)

IA32_HWP_CTL[0] controls the behavior of IA32_HWP_REQUEST Package Control [bit 42]. This control bit allows the IA32_HWP_REQUEST MSR to stay in INIT mode most of the time (Control Bit is equal to its RESET value of 0) thus avoiding actual saving/restoring of the MSR contents when the OS adds it to the register set saved and restored by XSAVES/XRSTORS.

- When IA32_HWP_CTL[0] = 0:
 - If IA32_HWP_REQUEST[42] = 0, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
 - If IA32_HWP_REQUEST[42] = 1, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according

to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 16.4.4.1 for additional details.

- When IA32_HWP_CTL[0] = 1, the behavior is reversed:
 - If IA32_HWP_REQUEST[42] = 1, the processor ignores all fields of the IA32_HWP_REQUEST_PKG MSR and selects all HWP values (Min, Max, EPP, Desired, Activity Window) from the IA32_HWP_REQUEST MSR.
 - If IA32_HWP_REQUEST[42] = 0, the processor selects the HWP values (Min, Max, EPP, Desired, Activity Window) either from the IA32_HWP_REQUEST MSR or from the IA32_HWP_REQUEST_PKG MSR according to the values contained in the IA32_HWP_REQUEST MSR bit fields [bits 63:59]. See Section 16.4.4.1 for additional details.

Section 16-2 summarizes the IA32_HWP_CTL MSR bit 0 control behavior.

Table 16-2. IA32_HWP_CTL MSR Bit 0 Behavior

Field	Description		
Thread request PKG CTL meaning	Defines which HWP Request MSR is used, whether thread level or package level. When the package MSR is used, the thread MSR valid bits define which thread MSR fields override the package (default 0).		
	IA32_HWP_CTL[PKG_CTL_PLR]	IA32_HWP_REQUEST[PKG_CTL]	HWP Request MSR Used
	0	0	IA32_HWP_REQUEST MSR
	0	1	IA32_HWP_REQUEST_PKG MSR
	1	0	IA32_HWP_REQUEST_PKG MSR
	1	1	IA32_HWP_REQUEST MSR

This MSR is supported if CPUID.06H:EAX[bit 22] is set.

If the IA32_PM_ENABLE[HWP_ENABLE] (bit 0) is not set, access to this MSR will generate a #GP fault.

16.4.5 HWP Feedback

The processor provides several types of feedback to the OS during HWP operation.

The IA32_MPERF MSR and IA32_APERF MSR mechanism (see Section 16.2) allows the OS to calculate the resultant effective frequency delivered over a time period. Energy efficiency and performance optimizations directly impact the resultant effective frequency delivered.

The layout of the IA32_HWP_STATUS MSR is shown in Figure 16-9. It provides feedback regarding changes to IA32_HWP_CAPABILITIES.Guaranteed_Performance, IA32_HWP_CAPABILITIES.Highest_Performance, excursions to IA32_HWP_CAPABILITIES.Minimum_Performance, and PECI_Override entry/exit events. The bit fields are described below:

- **Guaranteed_Performance_Change (bit 0, RWC0)** — If set (1), a change to Guaranteed_Performance has occurred. Software should query IA32_HWP_CAPABILITIES.Guaranteed_Performance value to ascertain the new Guaranteed Performance value and to assess whether to re-adjust HWP hints via IA32_HWP_REQUEST. Software must clear this bit by writing a zero (0).
- Bit 1 is reserved and must be zero.
- **Excursion_To_Minimum (bit 2, RWC0)** — If set (1), an excursion to Minimum_Performance of IA32_HWP_REQUEST has occurred. Software must clear this bit by writing a zero (0).
- **Highest_Change (bit 3, RWC0)** — If set (1), a change to Highest Performance has occurred. Software should query IA32_HWP_CAPABILITIES to ascertain the new Highest Performance value. Software must clear this bit by writing a zero (0). Interrupts upon Highest Performance change are supported if CPUID.06H:EAX[bit 15] is set.
- **PECI_Override_Entry (bit 4, RWC0)** — If set (1), an embedded/management controller has started a PECI override of one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are now overridden via the PECI mechanism and what their values are (see Section 16.4.4.3 for

- additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override entry are supported if CPUID.06H:EAX[bit 16] is set.
- **PECI_Override_Exit (bit 5, RWC0)** — If set (1), an embedded/management controller has stopped overriding one or more OS control hints (Min, Max, EPP) specified in IA32_HWP_REQUEST or IA32_HWP_REQUEST_PKG. Software may query IA32_HWP_PECI_REQUEST_INFO MSR to ascertain which fields are still overridden via the PECI mechanism and which fields are now back under software control (see Section 16.4.4.3 for additional details). Software must clear this bit by writing a zero (0). Interrupts upon PECI override exit are supported if CPUID.06H:EAX[bit 16] is set.
 - Bits 63:6 are reserved and must be zero.

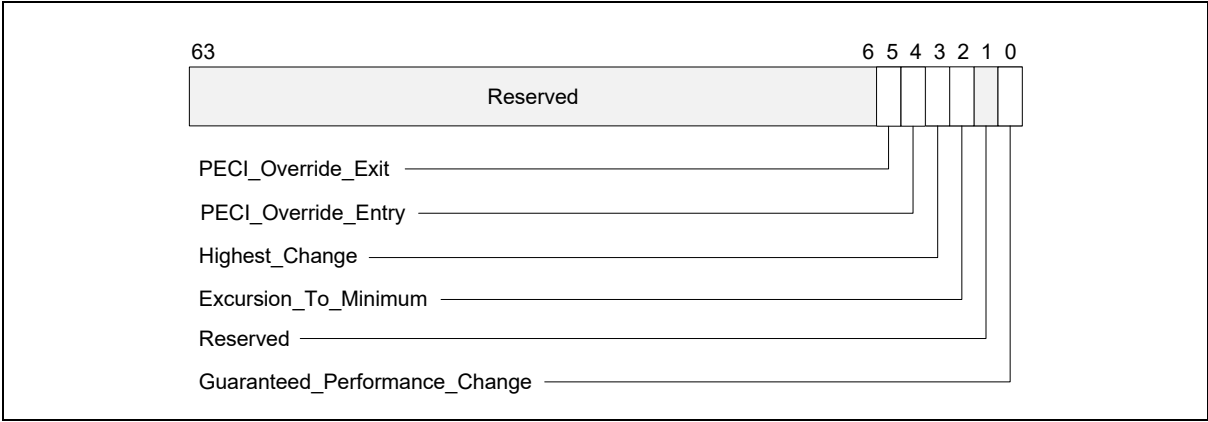


Figure 16-9. IA32_HWP_STATUS MSR

The status bits of IA32_HWP_STATUS must be cleared (0) by software so that a new status condition change will cause the hardware to set the bit again and issue the notification. Status bits are not set for “normal” excursions, e.g., running below Minimum Performance for short durations during C-state exit. Changes to Guaranteed_Performance, Highest_Performance, excursions to Minimum_Performance, or PECI_Override entry/exit will occur no more than once per second.

The OS can determine the specific reasons for a Guaranteed_Performance change or an excursion to Minimum_Performance in IA32_HWP_REQUEST by examining the associated status and log bits reported in the IA32_THERM_STATUS MSR. The layout of the IA32_HWP_STATUS MSR that HWP uses to support software query of HWP feedback is shown in Figure 16-10. The bit fields of IA32_THERM_STATUS associated with HWP feedback are described below (Bit fields of IA32_THERM_STATUS unrelated to HWP can be found in Section 16.8.5.2).

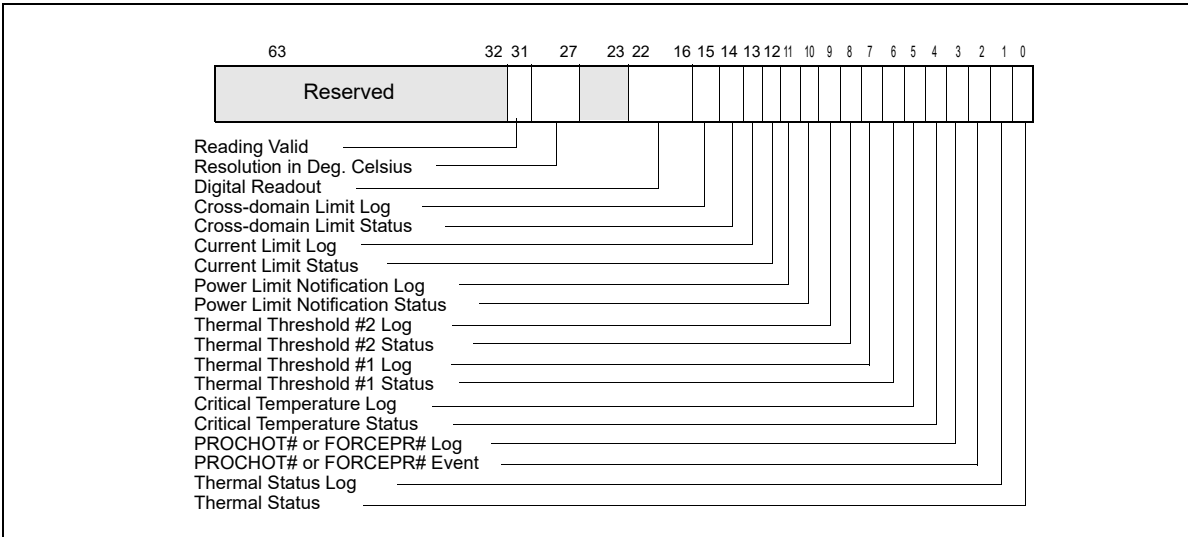


Figure 16-10. IA32_THERM_STATUS Register With HWP Feedback

- Bits 11:0, See Section 16.8.5.2.
- **Current Limit Status (bit 12, RO)** — If set (1), indicates an electrical current limit (e.g., Electrical Design Point/IccMax) is being exceeded and is adversely impacting energy efficiency optimizations.
- **Current Limit Log (bit 13, RWC0)** — If set (1), an electrical current limit has been exceeded that has adversely impacted energy efficiency optimizations since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- **Cross-domain Limit Status (bit 14, RO)** — If set (1), indicates another hardware domain (e.g., processor graphics) is currently limiting energy efficiency optimizations in the processor core domain.
- **Cross-domain Limit Log (bit 15, RWC0)** — If set (1), indicates another hardware domain (e.g., processor graphics) has limited energy efficiency optimizations in the processor core domain since the last clearing of this bit or a reset. This bit is sticky, software may clear this bit by writing a zero (0).
- Bits 63:16, See Section 16.8.5.2.

16.4.5.1 Non-Architectural HWP Feedback

The Productive Performance (MSR_PPERF) MSR (non-architectural) provides hardware's view of workload scalability, which is a rough assessment of the relationship between frequency and workload performance, to software. The layout of the MSR_PPERF is shown in Figure 16-11.

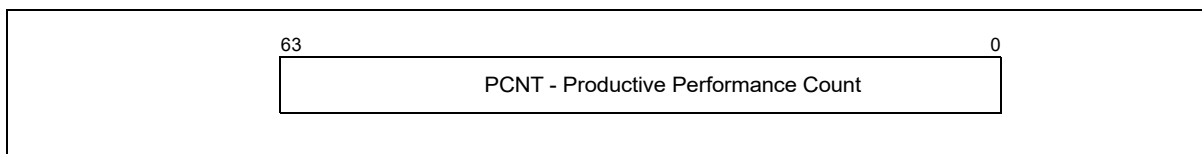


Figure 16-11. MSR_PPERF MSR

- **PCNT (bits 63:0, RO)** — Similar to IA32_APERF but only counts cycles perceived by hardware as contributing to instruction execution (e.g., unhalted and unstalled cycles). This counter increments at the same rate as IA32_APERF, where the ratio of ($\Delta\text{PCNT}/\Delta\text{ACNT}$) is an indicator of workload scalability (0% to 100%). Note that values in this register are valid even when HWP is not enabled.

16.4.6 HWP Notifications

Processors may support interrupt-based notification of changes to HWP status as indicated by CPUID. If supported, the IA32_HWP_INTERRUPT MSR is used to enable interrupt-based notifications. Notification events, when enabled, are delivered using the existing thermal LVT entry. The layout of the IA32_HWP_INTERRUPT is shown in Figure 16-12. The bit fields are described below:

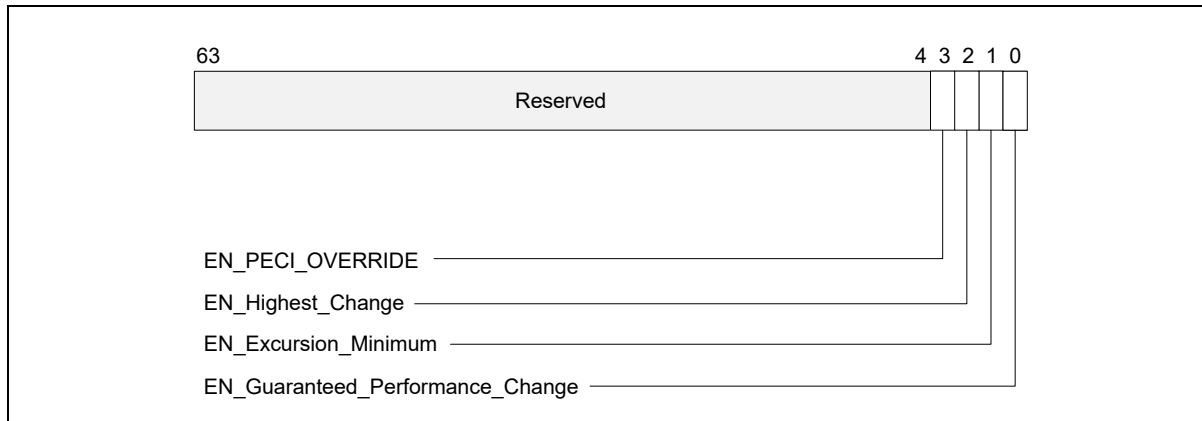


Figure 16-12. IA32_HWP_INTERRUPT MSR

- **EN_Guaranteed_Performance_Change (bit 0, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Guaranteed_Performance occurs. The default value is 0 (Interrupt generation is disabled).
- **EN_Excursion_Minimum (bit 1, RW)** — When set (1), an HWP Interrupt will be generated whenever the HWP hardware is unable to meet the IA32_HWP_REQUEST.Minimum_Performance setting. The default value is 0 (Interrupt generation is disabled).
- **EN_Highest_Change (bit 2, RW)** — When set (1), an HWP Interrupt will be generated whenever a change to the IA32_HWP_CAPABILITIES.Highest_Performance occurs. The default value is 0 (interrupt generation is disabled). Interrupts upon Highest Performance change are supported if CPUID.06H:EAX[bit 15] is set.
- **EN_PECI_OVERRIDE (bit 3, RW)** — When set (1), an HWP Interrupt will be generated whenever PECI starts or stops overriding any of the three HWP fields described in Section 16.4.4.3. The default value is 0 (interrupt generation is disabled). See Section 16.4.5 and Section 16.4.4.3 for details on how the OS learns what is the current set of HWP fields that are overridden by PECI. Interrupts upon PECI override change are supported if CPUID.06H:EAX[bit 16] is set.
- Bits 63:4 are reserved and must be zero.

16.4.7 Idle Logical Processor Impact on Core Frequency

Intel processors use one of two schemes for setting core frequency:

1. All cores share same frequency.
2. Each physical core is set to a frequency of its own.

In both cases the two logical processors that share a single physical core are set to the same frequency, so the processor accounts for the IA32_HWP_REQUEST MSR fields of both logical processors when defining the core frequency or the whole package frequency.

When **CPUID.06H:EAX[bit 20]** is set and only one logical processor of the two is active, while the other is idle (in any **C1 sub-state** or in a deeper sleep state), only the **active logical processor's** IA32_HWP_REQUEST MSR fields are considered, i.e., the HWP Request fields of a logical processor in the C1E sub-state or in a deeper sleep state are ignored.

Note: when a logical processor is in **C1 state** its HWP Request fields are accounted for.

16.4.8 Fast Write of Uncore MSR (Model Specific Feature)

There are a few logical processor scope MSRs whose values need to be observed outside the logical processor. The WRMSR instruction takes over 1000 cycles to complete (retire) for those MSRs. This overhead forces operating systems to avoid writing them too often whereas in many cases it is preferable that the OS writes them quite frequently for optimal power/performance operation of the processor.

The model specific “Fast Write MSR” feature reduces this overhead by an order of magnitude to a level of 100 cycles for a selected subset of MSRs.

Note: Writes to Fast Write MSRs are posted, i.e., when the WRMSR instruction completes, the data may still be “in transit” within the processor. Software can check the status by querying the processor to ensure data is already visible outside the logical processor (see Section 16.4.8.3 for additional details). Once the data is visible outside the logical processor, software is ensured that later writes by the same logical processor to the same MSR will be visible later (will not bypass the earlier writes).

MSRs that are selected for Fast Write are specified in a special capability MSR (see Section 16.4.8.1). Architectural MSRs that existed prior to the introduction of this feature and are selected for Fast Write, thus turning from slow to fast write MSRs, will be noted as such via a new CPUID bit. New MSRs that are fast upon introduction will be documented as such without an additional CPUID bit.

Three model specific MSRs are associated with the feature itself. They enable enumerating, controlling, and monitoring it. All three are logical processor scope.

16.4.8.1 FAST_UNCORE_MSRS_CAPABILITY (Address: 65FH, Logical Processor Scope)

Operating systems or BIOS can read the FAST_UNCORE_MSRS_CAPABILITY MSR to enumerate those MSRs that are Fast Write MSRs.

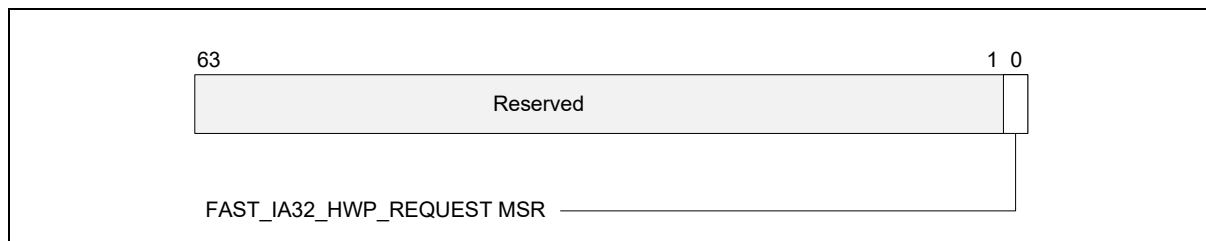


Figure 16-13. FAST_UNCORE_MSRS_CAPABILITY MSR

- **FAST_IA32_HWP_REQUEST MSR (bit 0, RO)** — When set (1), indicates that the IA32_HWP_REQUEST MSR is supported as a Fast Write MSR. A value of 0 indicates the IA32_HWP_REQUEST MSR is not supported as a Fast Write MSR.
- Bits 63:1 are reserved and must be zero.

16.4.8.2 FAST_UNCORE_MSRS_CTL (Address: 657H, Logical Processor Scope)

Operating Systems or BIOS can use the FAST_UNCORE_MSRS_CTL MSR to opt-in or opt-out for fast write of specific MSRs that are enabled for Fast Write by the processor.

Note: Not all MSRs that are selected for this feature will necessarily have this opt-in/opt-out option. They may be supported in fast write mode only.

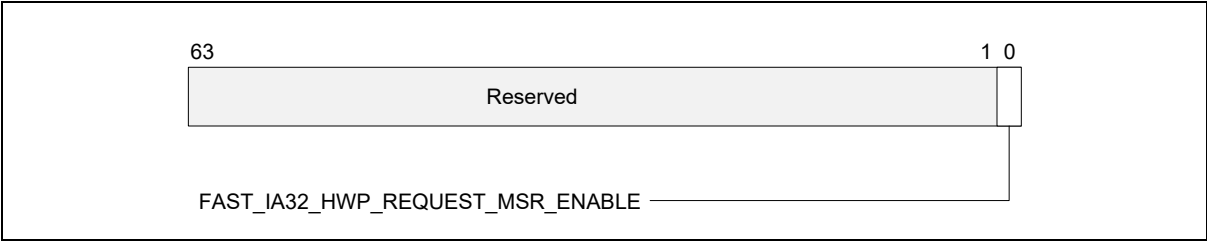


Figure 16-14. FAST_UNCORE_MSRS_CTL MSR

- **FAST_IA32_HWP_REQUEST_MSR_ENABLE (bit 0, R/W)** — When set (1), enables the fast access mode, low latency, posted IA32_HWP_REQUEST MSR feature and sets the related CPUID.06H:EAX[18] bit. The default value is 0. Note that this bit can only be enabled once from the default value. Once set, writes to this bit are ignored. Only CPU RESET will clear this bit.
- Bits 63:1 are reserved and must be zero.

16.4.8.3 FAST_UNCORE_MSRS_STATUS (Address: 65EH, Logical Processor Scope)

Software that executes the WRMSR instruction of a Fast Write MSR can check whether the data is already visible outside the logical processor by reading the FAST_UNCORE_MSRS_STATUS MSR. For each Fast Write MSR there is a status bit that indicates whether the data is already visible outside the logical processor or is still in “transit”.

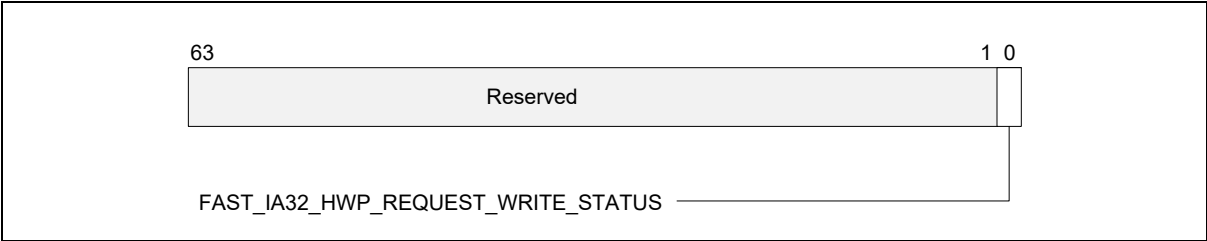


Figure 16-15. FAST_UNCORE_MSRS_STATUS MSR

- **FAST_IA32_HWP_REQUEST_WRITE_STATUS (bit 0, R/O)** — Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor.
- Bits 63:1 are reserved and must be zero.

16.4.9 Fast_IA32_HWP_REQUEST CPUID

IA32_HWP_REQUEST is an architectural MSR that exists in processors whose CPUID.06H:EAX[7] is set (HWP BASE is enabled). This MSR has logical processor scope, but after its contents are written the contents become visible outside the logical processor. When the FAST_IA32_HWP_REQUEST bit is set (CPUID.06H:EAX[18]), writes to the IA32_HWP_REQUEST MSR are visible outside the logical processor via the “Fast Write” feature described in Section 16.4.8.

16.4.10 Recommendations for OS use of HWP Controls

Common Cases of Using HWP

The default HWP control field values are expected to be suitable for many applications. The OS can enable autonomous HWP for these common cases by

- Setting `IA32_HWP_REQUEST.Desired_Performance = 0` (hardware autonomous selection determines the performance target). Set `IA32_HWP_REQUEST.Activity_Window = 0` (enable HW dynamic selection of window size).

To maximize HWP benefit for the common cases, the OS should set

- `IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance` and
- `IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance`.

Setting `IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance` is functionally equivalent to using of the `IA32_PERF_CTL` interface and is therefore not recommended (bypassing HWP).

Calibrating HWP for Application-Specific HWP Optimization

In some applications, the OS may have Quality of Service requirements that may not be met by the default values. The OS can characterize HWP by:

- keeping `IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance` to prevent non-linearity in the characterization process,
- utilizing the range values enumerated from the `IA32_HWP_CAPABILITIES` MSR to program `IA32_HWP_REQUEST` while executing workloads of interest and observing the power and performance result.

The power and performance result of characterization is also influenced by the `IA32_HWP_REQUEST.Energy_Performance_Preference` field, which must also be characterized.

Characterization can be used to set `IA32_HWP_REQUEST.Minimum_Performance` to achieve the required QOS in terms of performance. If `IA32_HWP_REQUEST.Minimum_Performance` is set higher than `IA32_HWP_CAPABILITIES.Guaranteed_Performance` then notification of excursions to Minimum Performance may be continuous.

If autonomous selection does not deliver the required workload performance, the OS should assess the current delivered effective frequency and for the duration of the specific performance requirement set `IA32_HWP_REQUEST.Desired_Performance ≠ 0` and adjust `IA32_HWP_REQUEST.Energy_Performance_Preference` as necessary to achieve the required workload performance. The `MSR_PPERF.PCNT` value can be used to better comprehend the potential performance result from adjustments to `IA32_HWP_REQUEST.Desired_Performance`. The OS should set `IA32_HWP_REQUEST.Desired_Performance = 0` to re-enable autonomous selection.

Tuning for Maximum Performance or Lowest Power Consumption

Maximum performance will be delivered by setting `IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Highest_Performance` and setting `IA32_HWP_REQUEST.Energy_Performance_Preference = 0` (performance preference).

Lowest power will be achieved by setting `IA32_HWP_REQUEST.Minimum_Performance = IA32_HWP_REQUEST.Maximum_Performance = IA32_HWP_CAPABILITIES.Lowest_Performance` and setting `IA32_HWP_REQUEST.Energy_Performance_Preference = 0FFH` (energy efficiency preference).

Mixing Logical Processor and Package Level HWP Field Settings

Using the `IA32_HWP_REQUEST.Package_Control` bit and the five valid bits in that MSR, the OS can mix and match between selecting the Logical Processor scope fields and the Package level fields. For example, the OS can set all logical cores' `IA32_HWP_REQUEST.Package_Control` bit to '1', and for those logical processors if it prefers a different EPP value than the one set in the `IA32_HWP_REQUEST_PKG` MSR, the OS can set the desired EPP value and the EPP valid bit. This overrides the package EPP value for only a subset of the logical processors in the package.

Additional Guidelines

Set `IA32_HWP_REQUEST.Energy_Performance_Preference` as appropriate for the platform's current mode of operation. For example, a mobile platforms' setting may be towards performance preference when on AC power and more towards energy efficiency when on DC power.

The use of the Running Average Power Limit (RAPL) processor capability (see section 14.7.1) is highly recommended when HWP is enabled. Use of `IA32_HWP_Request.Maximum_Performance` for thermal control is subject to limitations and can adversely impact the performance of other processor components, e.g., graphics

If default values deliver undesirable performance latency in response to events, the OS should set IA32_HWP_REQUEST.Activity_Window to a low (non-zero) value and IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) for the event duration.

Similarly, for “real-time” threads, set IA32_HWP_REQUEST.Energy_Performance_Preference towards performance (0) and IA32_HWP_REQUEST.Activity_Window to a low value, e.g., 01H, for the duration of their execution.

When executing low priority work that may otherwise cause the hardware to deliver high performance, set IA32_HWP_REQUEST.Activity_Window to a longer value and reduce the IA32_HWP_Request.Maximum_Performance value as appropriate to control energy efficiency. Adjustments to IA32_HWP_REQUEST.Energy_Performance_Preference may also be necessary.

16.5 HARDWARE DUTY CYCLING (HDC)

Intel processors may contain support for Hardware Duty Cycling (HDC), which enables the processor to autonomously force its components inside the physical package into idle state. For example, the processor may selectively force only the processor cores into an idle state.

HDC is disabled by default on processors that support it. System software can dynamically enable or disable HDC to force one or more components into an idle state or wake up those components previously forced into an idle state. Forced Idling (and waking up) of multiple components in a physical package can be done with one WRMSR to a packaged-scope MSR from any logical processor within the same package.

HDC does not delay events such as timer expiration, but it may affect the latency of short (less than 1 msec) software threads, e.g., if a thread is forced to idle state just before completion and entering a “natural idle”.

HDC forced idle operation can be thought of as operating at a lower effective frequency. The effective average frequency computed by software will include the impact of HDC forced idle.

The primary use of HDC is enable system software to manage low active workloads to increase the package level C6 residency. Additionally, HDC can lower the effective average frequency in case of power or thermal limitation.

When HDC forces a logical processor, a processor core or a physical package to enter an idle state, its C-State is set to C3 or deeper. The deep “C-states” referred to in this section are processor-specific C-states.

16.5.1 Hardware Duty Cycling Programming Interfaces

The programming interfaces provided by HDC include the following:

- The CPUID instruction allows software to discover the presence of HDC support in an Intel processor. Specifically, CPUID.06H:EAX[13] indicates the processor’s support of the following aspects of HDC.
 - Availability of HDC baseline resource, CPUID.06H:EAX[13]: If this bit is set, HDC provides the following architectural MSRs: IA32_PKG_HDC_CTL, IA32_PM_CTL1, and the IA32_THREAD_STALL MSRs.
- Additionally, HDC may provide several non-architectural MSR.

Table 16-3. Architectural and non-Architecture MSRs Related to HDC

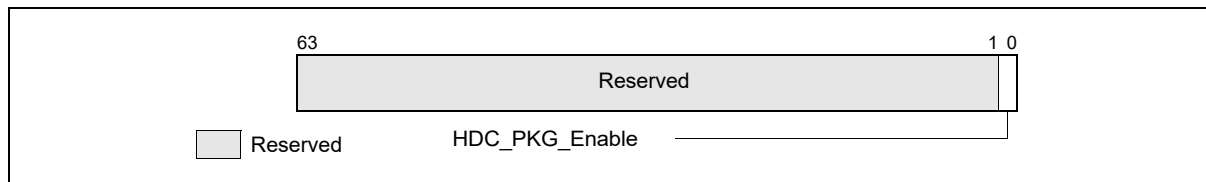
Address	Architectural	Register Name	Description
DB0H	Y	IA32_PKG_HDC_CTL	Package Enable/Disable HDC.
DB1H	Y	IA32_PM_CTL1	Per-logical-processor select control to allow/block HDC forced idling.
DB2H	Y	IA32_THREAD_STALL	Accumulate stalled cycles on this logical processor due to HDC forced idling.
653H	N	MSR_CORE_HDC_RESIDENCY	Core level stalled cycle counter due to HDC forced idling on one or more logical processor.
655H	N	MSR_PKG_HDC_SHALLOW_RESIDENCY	Accumulate the cycles the package was in C2 ¹ state and at least one logical processor was in forced idle
656H	N	MSR_PKG_HDC_DEEP_RESIDENCY	Accumulate the cycles the package was in the software specified Cx ¹ state and at least one logical processor was in forced idle. Cx is specified in MSR_PKG_HDC_CONFIG_CTL.
652H	N	MSR_PKG_HDC_CONFIG_CTL	HDC configuration controls

NOTES:

1. The package “C-states” referred to in this section are processor-specific C-states.

16.5.2 Package level Enabling HDC

The layout of the IA32_PKG_HDC_CTL MSR is shown in Figure 16-16. IA32_PKG_HDC_CTL is a writable MSR from any logical processor in a package. The bit fields are described below:

**Figure 16-16. IA32_PKG_HDC_CTL MSR**

- **HDC_PKG_Enable (bit 0, R/W)** — Software sets this bit to enable HDC operation by allowing the processor to force to idle all “HDC-allowed” (see Figure 16.5.3) logical processors in the package. Clearing this bit disables HDC operation in the package by waking up all the processor cores that were forced into idle by a previous ‘0’-to-‘1’ transition in IA32_PKG_HDC_CTL.HDC_PKG_Enable. This bit is writable only if CPUID.06H:EAX[13] = 1. Default = zero (0).
- Bits 63:1 are reserved and must be zero.

After processor support is determined via CPUID, system software can enable HDC operation by setting IA32_PKG_HDC_CTL.HDC_PKG_Enable to 1. At reset, IA32_PKG_HDC_CTL.HDC_PKG_Enable is cleared to 0. A ‘0’-to-‘1’ transition in HDC_PKG_Enable allows the processor to force to idle all HDC-allowed (indicated by the non-zero state of IA32_PM_CTL1[bit 0]) logical processors in the package. A ‘1’-to-‘0’ transition wakes up those HDC force-idled logical processors.

Software can enable or disable HDC using this package level control multiple times from any logical processor in the package. Note the latency of writing a value to the package-visible IA32_PKG_HDC_CTL.HDC_PKG_Enable is longer than the latency of a WRMSR operation to a Logical Processor MSR (as opposed to package level MSR) such as: IA32_PM_CTL1 (described in Section 16.5.3). Propagation of the change in IA32_PKG_HDC_CTL.HDC_PKG_Enable and reaching all HDC idled logical processor to be woken up may take on the order of core C6 exit latency.

16.5.3 Logical-Processor Level HDC Control

The layout of the IA32_PM_CTL1 MSR is shown in Figure 16-17. Each logical processor in a package has its own IA32_PM_CTL1 MSR. The bit fields are described below:

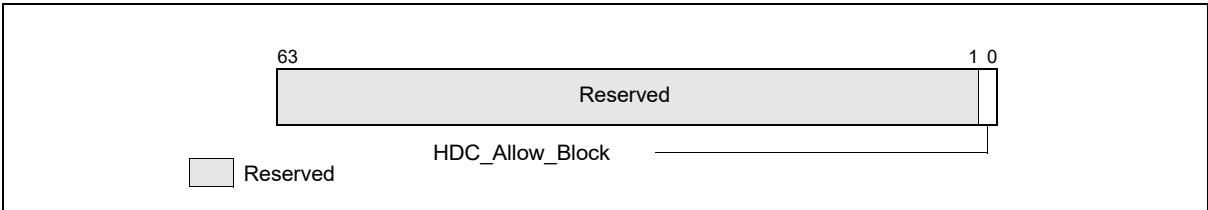


Figure 16-17. IA32_PM_CTL1 MSR

- **HDC_Allow_Block (bit 0, R/W)** — Software sets this bit to allow this logical processors to honor the package-level IA32_PKG_HDC_CTL.HDC_PKG_Enable control. Clearing this bit prevents this logical processor from using the HDC. This bit is writable only if CPUID.06H:EAX[13] = 1. Default = one (1).
- Bits 63:1 are reserved and must be zero.

Fine-grain OS control of HDC operation at the granularity of per-logical-processor is provided by IA32_PM_CTL1. At RESET, all logical processors are allowed to participate in HDC operation such that OS can manage HDC using the package-level IA32_PKG_HDC_CTL.

Writes to IA32_PM_CTL1 complete with the latency that is typical to WRMSR to a Logical Processor level MSR. When the OS chooses to manage HDC operation at per-logical-processor granularity, it can write to IA32_PM_CTL1 on one or more logical processors as desired. Each write to IA32_PM_CTL1 must be done by code that executes on the logical processor targeted to be allowed into or blocked from HDC operation.

Blocking one logical processor for HDC operation may have package level impact. For example, the processor may decide to stop duty cycling of all other Logical Processors as well.

The propagation of IA32_PKG_HDC_CTL.HDC_PKG_Enable in a package takes longer than a WRMSR to IA32_P-M_CTL1. The last completed write to IA32_PM_CTL1 on a logical processor will be honored when a '0'-to-'1' transition of IA32_PKG_HDC_CTL.HDC_PKG_Enable arrives to a logical processor.

16.5.4 HDC Residency Counters

There is a collection of counters available for software to track various residency metrics related to HDC operation. In general, HDC residency time is defined as the time in HDC forced idle state at the granularity of per-logical-processor, per-core, or package. At the granularity of per-core/package-level HDC residency, at least one of the logical processor in a core/package must be in the HDC forced idle state.

16.5.4.1 IA32_THREAD_STALL

Software can track per-logical-processor HDC residency using the architectural MSR IA32_THREAD_STALL. The layout of the IA32_THREAD_STALL MSR is shown in Figure 16-18. Each logical processor in a package has its own IA32_THREAD_STALL MSR. The bit fields are described below:

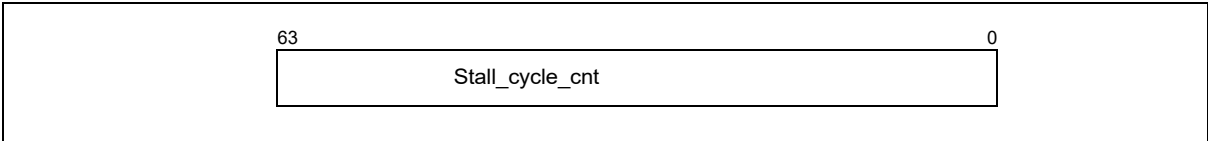


Figure 16-18. IA32_THREAD_STALL MSR

- **Stall_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after the logical processor exits from the forced idled C-state. At each update, the number of cycles that the logical processor was stalled due to forced-idle will be added to the counter. This counter is available only if CPUID.06H:EAX[13] = 1. Default = zero (0).

A value of zero in IA32_THREAD_STALL indicates either HDC is not supported or the logical processor never serviced any forced HDC idle. A non-zero value in IA32_THREAD_STALL indicates the HDC forced-idle residency times of the logical processor. It also indicates the forced-idle cycles due to HDC that could appear as C0 time to traditional OS accounting mechanisms (e.g., time-stamping OS idle/exit events).

Software can read IA32_THREAD_STALL irrespective of the state of IA32_PKG_HDC_CTL and IA32_PM_CTL1, as long as CPUID.06H:EAX[13] = 1.

16.5.4.2 Non-Architectural HDC Residency Counters

Processors that support HDC operation may provide the following model-specific HDC residency counters.

MSR_CORE_HDC_RESIDENCY

Software can track per-core HDC residency using the counter MSR_CORE_HDC_RESIDENCY. This counter increments when the core is in C3 state or deeper (all logical processors in this core are idle due to either HDC or other mechanisms) and at least one of the logical processors is in HDC forced idle state. The layout of the MSR_CORE_HDC_RESIDENCY is shown in Figure 16-19. Each processor core in a package has its own MSR_CORE_HDC_RESIDENCY MSR. The bit fields are described below:

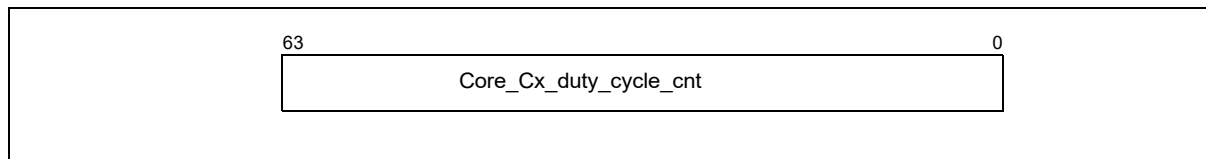


Figure 16-19. MSR_CORE_HDC_RESIDENCY MSR

- **Core_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after core C-state exit from a forced idled C-state. At each update, the increment counts cycles when the core is in a Cx state (all its logical processor are idle) and at least one logical processor in this core was forced into idle state due to HDC. If CPUID.06H:EAX[13] = 0, attempt to access this MSR will cause a #GP fault. Default = zero (0).

A value of zero in MSR_CORE_HDC_RESIDENCY indicates either HDC is not supported or this processor core never serviced any forced HDC idle.

MSR_PKG_HDC_SHALLOW_RESIDENCY

The counter MSR_PKG_HDC_SHALLOW_RESIDENCY allows software to track HDC residency time when the package is in C2 state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. The layout of the MSR_PKG_HDC_SHALLOW_RESIDENCY is shown in Figure 16-20. There is one MSR_PKG_HDC_SHALLOW_RESIDENCY per package. The bit fields are described below:

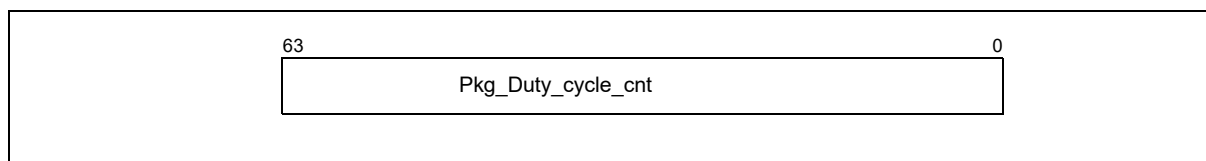


Figure 16-20. MSR_PKG_HDC_SHALLOW_RESIDENCY MSR

- **Pkg_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. Package shallow residency may be implementation specific. In the initial implementation, the threshold is package C2-state. The count is updated only after package C2-state exit from a forced idled C-state. At each update, the increment counts cycles when the package is in C2 state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_DEEP_RESIDENCY

The counter MSR_PKG_HDC_DEEP_RESIDENCY allows software to track HDC residency time when the package is in a software-specified package Cx state, all processor cores in the package are not active and at least one logical processor was forced into idle state due to HDC. Selection of a specific package Cx state can be configured using MSR_PKG_HDC_CONFIG. The layout of the MSR_PKG_HDC_DEEP_RESIDENCY is shown in Figure 16-21. There is one MSR_PKG_HDC_DEEP_RESIDENCY per package. The bit fields are described below:

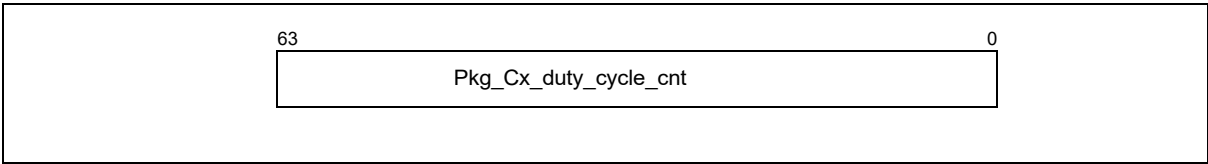


Figure 16-21. MSR_PKG_HDC_DEEP_RESIDENCY MSR

- **Pkg_Cx_Duty_Cycle_Cnt (bits 63:0, R/O)** — Stores accumulated HDC forced-idle cycle count of this processor core since last RESET. This counter increments at the same rate of the TSC. The count is updated only after package C-state exit from a forced idle state. At each update, the increment counts cycles when the package is in the software-configured Cx state and at least one processor core in this package was forced into idle state due to HDC. If CPUID.06H:EAX[13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).

A value of zero in MSR_PKG_HDC_SHALLOW_RESIDENCY indicates either HDC is not supported or this processor package never serviced any forced HDC idle.

MSR_PKG_HDC_CONFIG

MSR_PKG_HDC_CONFIG allows software to configure the package Cx state that the counter MSR_PKG_HDC_DEEP_RESIDENCY monitors. The layout of the MSR_PKG_HDC_CONFIG is shown in Figure 16-22. There is one MSR_PKG_HDC_CONFIG per package. The bit fields are described below:

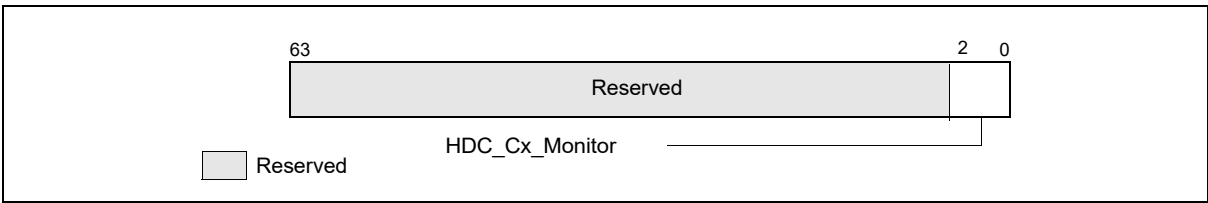


Figure 16-22. MSR_PKG_HDC_CONFIG MSR

- **Pkg_Cx_Monitor (bits 2:0, R/W)** — Selects which package C-state the MSR_HDC_DEEP_RESIDENCY counter will monitor. The encoding of the HDC_Cx_Monitor field are: **0**: no-counting; **1**: count package C2 only; **2**: count package C3 and deeper; **3**: count package C6 and deeper; **4**: count package C7 and deeper; other encodings are reserved. If CPUID.06H:EAX[13] = 0, attempt to access this MSR may cause a #GP fault. Default = zero (0).
- Bits 63:3 are reserved and must be zero.

16.5.5 MPERF and APERF Counters Under HDC

HDC operation can be thought of as an average effective frequency drop due to all or some of the Logical Processors enter an idle state period.

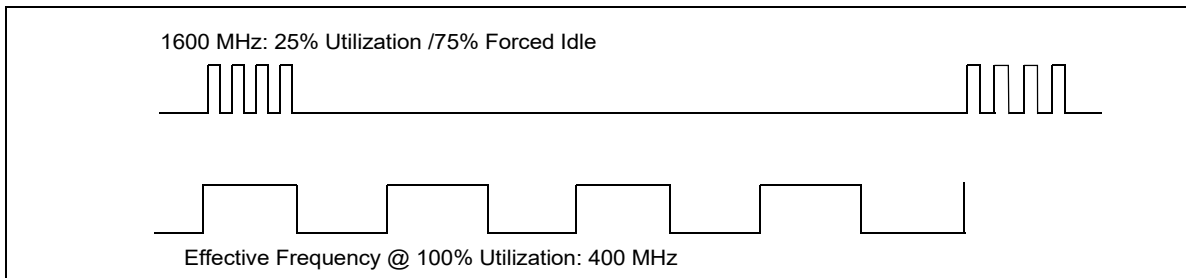


Figure 16-23. Example of Effective Frequency Reduction and Forced Idle Period of HDC

By default, the IA32_MPERF counter counts during forced idle periods as if the logical processor was active. The IA32_APERF counter does not count during forced idle state. This counting convention allows the OS to compute the average effective frequency of the Logical Processor between the last MWAIT exit and the next MWAIT entry (OS visible C0) by $\Delta\text{ACNT}/\Delta\text{MCNT} * \text{TSC Frequency}$.

16.6 HARDWARE FEEDBACK INTERFACE AND INTEL® THREAD DIRECTOR

Intel processors that enumerate CPUID.06H.00H:EAX.HW_FEEDBACK[19] as 1 support Hardware Feedback Interface (HFI). Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a hardware feedback interface structure in memory. Details on this table structure are described in Section 16.6.1.

Intel processors that enumerate CPUID.06H.00H:EAX[23] as 1 support Intel® Thread Director. Hardware provides guidance to the Operating System (OS) scheduler to perform optimal workload scheduling through a memory resident table and software thread specific index (Class ID) that points into that table and selects which data to use for that software thread. Details on this table structure are described in Section 16.6.2.

16.6.1 Hardware Feedback Interface Table Structure

This structure has a global header that is 16 bytes in size. Following this global header, there is one 8 byte entry per logical processor in the socket. The structure is designed as follows.

Table 16-4. Hardware Feedback Interface Structure

Byte Offset	Size (Bytes)	Description
0	16	Global Header
16	8	Per Logical Processor Entry
24	8	Per Logical Processor Entry
...	...	...
16 + n*8	8	Per Logical Processor Entry

The global header is structured as shown in Table 16-5.

Table 16-5. Hardware Feedback Interface Global Header Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	8	Timestamp	Timestamp of when the table was last updated by hardware. This is a timestamp in crystal clock units. Initialized by the OS to 0.
8	1	Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
9	1	Energy Efficiency Capability Changed	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
10	6	Reserved	Initialized by the OS to 0.

The per logical processor scheduler feedback entry is structured as follows. The operating system can determine the index of the logical processor feedback entry for a logical processor using CPUID.06H.00H:EDX[31:16] by executing CPUID on that logical processor.

Table 16-6. Hardware Feedback Interface Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.00H:EDX[0] enumerates support for Performance capability reporting.
1	1	Energy Efficiency Capability	Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback. CPUID.06H.00H:EDX[1] enumerates support for Energy Efficiency capability reporting.
2	6	Reserved	The OS scheduler is expected to initialize the Hardware Feedback Interface Structure to 0 prior to enabling Hardware Feedback.

16.6.2 Intel® Thread Director Table Structure

This structure has a global header that is at least 16 bytes in size. Its size depends on the number of classes and capabilities enumerated by the CPUID instruction (see notes below Table 16-7). Following this global header there are multiple Logical Processor related entries. The structure is designed as follows.

Table 16-7. Intel® Thread Director Table Structure

Byte Offset ^{1,2,3}	Size (Bytes)	Description
0	$8 + CP^4 * CL^4 + R8^5$	Global Header
$8 + CP * CL + R8$	$CL * CP + R8$	Per Logical Processor Entry ₀ ⁶
$8 + 2 * (CP * CL + R8)$	$CL * CP + R8$	Per Logical Processor Entry ₁
...	...	...
$8 + (N^7 - 1) * (CP * CL + R8)$	$CL * CP + R8$	Logical Processor Entry _{N-1}

NOTES:

1. Byte offset of Capability_{cp} of Class_{cl} change indication: $8 + CP * cl + cp$.
2. Byte offset of LP Entry_i: $8 + (i+1) * (CP * CL + R8)$.
3. Byte offset of capability_{cp} of class_{cl} of LP Entry_i: $8 + (i+1) * (CP * CL + R8) + CP * cl + cp$.
4. Both upper case CL and CP denote total number of classes and capabilities defined for the processor. Lower case cl and cp denote one instance of a class or capability. cl and cp are counted starting at zero. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A for the number of classes (CL) and the number of supported capabilities (CP). CP (# of capabilities): number of enumerated bits in CPUID.06H.00H:EDX[7:0] and CL (# of classes): CPUID.06H.00H:ECX[15:8].
5. R8 is the number of bytes necessary to round up the Capability Change Indication array and the Logical Processor Entry to whole multiple of 8 bytes.
6. Table size: $8 + (N+1) * (CP * CL + R8)$.
7. N is the number of Logical Processor Entries in the table. It is not greater than the number of Logical Processors on the socket, but may be lower.
8. The Operating System can determine the index for the Logical Processor Entry within the Intel Thread Director table using CPUID.06H.00H:EDX[31:16] by executing the CPUID instruction on that Logical Processor.
9. The Operating System should allocate space to accommodate for one such structure per socket in the system.
10. The Intel Thread Director table structure extends the Hardware Feedback Interface table structure without breaking backward compatibility. The Hardware Feedback Interface can be viewed as having two capabilities and a single class.

The global header is structured as shown in Table 16-8.

Table 16-8. Intel® Thread Director Global Header Structure

Byte Offset	Size (Bytes)	Description	
0	8	Time-stamp of when the table was last updated by hardware. This is a time-stamp in crystal clock units. Initialized by the OS to 0.	
8	1	Class 0 Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + 1	1	Class 0 Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
...			
8 + CP - 1	1	Class 0 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
8 + CP	1	Class 1 Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + CP + 1	1	Class 1 Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
...			
8 + 2*CP - 1	1	Class 1 change indication for Capability #(CP-1) if exists	Unavailable for capabilities that are not enumerated.
...			Change indication for Capabilities of additional Classes if exist.
8 + (CL-1)*CP	1	Class #(CL-1) Performance Capability Flags	If bit 0 is set to 1, indicates the performance capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.
8 + (CL-1)*CP + 1	1	Class #(CL-1) Energy Efficiency Capability Flags	If bit 0 is set to 1, indicates the energy efficiency capability field for one or more logical processors was updated in the table and/or another bit in this field is updated. If bit 1 is set to 1, indicates a force idle/inject idle request to the OS for one or more logical processors. Initialized by the OS to 0.

Table 16-8. Intel® Thread Director Global Header Structure (Contd.)

Byte Offset	Size (Bytes)	Description
...		
8 + CL*CP - 1	1	Class #(CL-1) change indication for Capability #(CP-1) if exists
8 + CL*CP	R8	Padding
		Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

The logical processor capability entry in the Intel Thread Director table is structured as follows.

Table 16-9. Intel® Thread Director Logical Processor Entry Structure

Byte Offset	Size (Bytes)	Field Name	Description
0	1	Performance Capability	Class 0 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
1	1	Energy Efficiency Capability	Class 0 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
CP - 1	1	Capability #(CP-1)	Class 0 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.
CP + R8	1	Performance Capability	Class 1 Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
CP + 1	1	Energy Efficiency Capability	Class 1 Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
2*CP - 1	1	Capability #(CP-1)	Class 1 Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).

Table 16-9. Intel® Thread Director Logical Processor Entry Structure (Contd.)

Byte Offset	Size (Bytes)	Field Name	Description
2*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.
...			
(CL-1)*CP	1	Performance Capability	Class #(CL-1) Performance capability is an 8-bit value (0 ... 255) specifying the relative performance level of a single logical processor. Higher values indicate higher performance; the lowest performance level of 0 indicates a recommendation to the OS to not schedule any software threads on it for performance reasons. Initialized by the OS to 0.
(CL-1)*CP + 1	1	Energy Efficiency Capability	Class #(CL-1) Energy Efficiency capability is an 8-bit value (0 ... 255) specifying the relative energy efficiency level of a logical processor. Higher values indicate higher energy efficiency; the lowest energy efficiency capability of 0 indicates a recommendation to the OS to not schedule any software threads on it for efficiency reasons. An Energy Efficiency capability of 255 indicates which logical processors have the highest relative energy efficiency capability. In addition, the value 255 is an explicit recommendation for the OS to consolidate work on those logical processors for energy efficiency reasons. Initialized by the OS to 0.
...			
CL*CP - 1	1	Capability #(CP-1)	Class #(CL-1) Capability #(CP-1) if exists. If the capability does not exist (is not enumerated in the CPUID), the entry is unavailable (no space is reserved for future use here).
CL*CP	R8	Padding	Padding to nearest multiple of 8 bytes. Initialized by the OS to 0 prior to enabling Intel Thread Director.

16.6.3 Intel® Thread Director Usage Model

When the OS Scheduler needs to decide which one of multiple free logical processors to assign to a software thread that is ready to execute, it can choose one of the following options:

1. The free logical processor with the highest performance value of that software thread class, if the system is scheduling for performance.
2. The free logical processor with the highest energy efficiency value of that software thread class, if the system is scheduling for energy efficiency.

When the OS Scheduler needs to decide which of two logical processors (i,j) to assign to which of two software threads whose Class IDs are k1 and k2, it can compute the two performance ratios: $\text{Perf Ratio}_1 = \text{Perf}_{ik1} / \text{Perf}_{jk1}$ and $\text{Perf Ratio}_2 = \text{Perf}_{ik2} / \text{Perf}_{jk2}$, or two energy efficiency ratios: $\text{Energy Eff. Ratio}_1 = \text{Energy Eff}_{ik1} / \text{Energy Eff}_{jk1}$ and $\text{Energy Eff. Ratio}_2 = \text{Energy Eff}_{ik2} / \text{Energy Eff}_{jk2}$ between the two logical processors for each of the two classes, depending on whether the OS is scheduling for performance or for energy efficiency.

For example, assume that the system is scheduling for performance and that $\text{Perf Ratio}_1 > \text{Perf Ratio}_2$. The OS Scheduler will assign the software thread whose Class ID is k1 to logical processor i, and the one whose Class ID is k2 to logical processor j.

When the two software threads in question belong to the same Class ID, the OS Scheduler can schedule to higher performance logical processors within that class when scheduling for performance and to higher energy efficiency logical processors within that class when scheduling for energy efficiency.

The highest to lowest ordering may be different between classes across cores and between the performance column and the energy efficiency column of the same class across cores.

16.6.4 Hardware Feedback Interface Pointer

The physical address of the HFI/Intel Thread Director structure is programmed by the OS into a package scoped MSR named `IA32_HW_FEEDBACK_PTR`. The MSR is structured as follows:

- Bits 63:MAXPHYADDR¹ — Reserved.
- Bits MAXPHYADDR-1:12 — ADDR. This is the physical address of the page frame of the first page of this structure.
- Bits 11:1 — Reserved.
- Bit 0 — Valid. When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.

The address of this MSR is 17D0H. This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

CPUID.06H.00H:EDX[11:8] enumerates the size of memory that must be allocated by the OS for this structure.

16.6.5 Hardware Feedback Interface Configuration

The operating system enables HFI/Intel Thread Director using a package scoped MSR named `IA32_HW_FEEDBACK_CONFIG` (address 17D1H). This MSR is cleared on processor reset to its default value of 0. It retains its value upon INIT.

The MSR is structured as follows:

- Bits 63:2 — Reserved.
- Bit 1 — Enable Intel Thread Director (or multi-class support). Both bits 0 and 1 must be set for Intel Thread Director to be enabled. The extra class columns in the Intel Thread Director table are updated by hardware immediately following setting those two bits, as well as during run time as necessary.
- Bit 0 — Enable. When set to 1, enables HFI.

Before enabling HFI, the OS must set a valid hardware feedback interface structure using `IA32_HW_FEEDBACK_PTR`.

When the OS sets bit 0 only, the hardware populates class 0 capabilities only in the HFI structure. When bit 1 is set after or together with bit 0, the Intel Thread Director multi-class structure is populated.

When either the HFI structure or the Intel Thread Director structure are ready to use by the OS, the hardware sets `IA32_PACKAGE_THERM_STATUS`[bit 26]. An interrupt is generated by the hardware if `IA32_PACKAGE_THERM_INTERRUPT`[bit 25] is set.

When the OS clears bit 1 but leaves bit 0 set, Intel Thread Director is disabled, but HFI is kept operational. `IA32_PACKAGE_THERM_STATUS`[bit 26] is NOT set in this case.

Clearing bit 0 disables both HFI and Intel Thread Director, independent of the bit 1 state. Setting bit 1 to '1' while keeping bit 0 at '0' is an invalid combination which is quietly ignored.

When the OS clears bit 0, hardware sets the `IA32_PACKAGE_THERM_STATUS`[bit 26] to 1 to acknowledge disabling of the interface. The OS should wait for this bit to be set to 1 to reclaim the memory of the Intel Thread Director structure, as by setting `IA32_PACKAGE_THERM_STATUS`[bit 26] hardware guarantees not to write into the Intel Thread Director structure anymore.

The OS may clear bit 0 only after receiving an indication from the hardware that the structure initialization is complete via the same `IA32_PACKAGE_THERM_STATUS`[bit 26], following enabling of HFI/Intel Thread Director, thus avoiding a race condition between OS and hardware.

Bit 1 is valid only if CPUID.06H:EAX[bit 23] is set. When setting this bit while support is not enumerated, the hardware generates #GP.

Table 16-10 summarizes the control options described above.

See Section 16.6.9 for details on scenarios where `IA32_HW_FEEDBACK_CONFIG` bits are implicitly reset by the hardware.

1. MAXPHYADDR is derived from the value enumerated in CPUID.80000008H:EAX[7:0]. If `IA32_TME_ACTIVATE`[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of `IA32_TME_ACTIVATE`[39:36] when a logical processor is out-side secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.

Table 16-10. IA32_HW_FEEDBACK_CONFIG Control Options

Pre-Bit 1	Pre-Bit 0	Post-Bit 1	Post-Bit 0	Action	IA32_PACKAGE_THERM_STATUS [bit 26] and Interrupt
0	0	0	0	Reset value.	Both Hardware Feedback Interface and Intel Thread Director are disabled, no status bit set, no interrupt is generated.
0	0	0	1	Enable HFI structure.	Set the status bit and generate interrupt if enabled.
0	0	1	0	Invalid option; quietly ignored by the hardware.	No action (no update in the table).
0	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	0	0	Disable HFI support.	Set the status bit and generate interrupt if enabled.
0	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
0	1	1	1	Enable Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	0	0	0	No action; keeps HFI and Intel Thread Director disabled.	No action (no update in the table).
1	0	0	1	Enable HFI.	Set the status bit and generate interrupt if enabled.
1	0	1	1	Enable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.
1	1	0	1	Disable Intel Thread Director; keep HFI enabled.	No action (no update in the table).
1	1	1	0	Disable HFI and Intel Thread Director.	Set the status bit and generate interrupt if enabled.

16.6.6 Hardware Feedback Interface Notifications

The IA32_PACKAGE_THERM_STATUS MSR is extended with a new bit, hardware feedback interface structure change status (bit 26, R/WC0), to indicate that the hardware has updated the HFI/Intel Thread Director structure. This is a sticky bit and once set, indicates that the OS should read the structure to determine the change and adjust its scheduling decisions. Once set, the hardware will not generate any further updates to this structure until the OS clears this bit by writing 0.

The OS can enable interrupt-based notifications when the structure is updated by hardware through a new enable bit, hardware feedback interrupt enable (bit 25, R/W), in the IA32_PACKAGE_THERM_INTERRUPT MSR. When this bit is set to 1, it enables the generation of an interrupt when the HFI/Intel Thread Director structure is updated by hardware. When the enable bit transitions from 0 to 1, hardware will generate an initial notify, with the IA32_PACKAGE_THERM_STATUS bit 26 set to 1, to indicate that the OS should read the current HFI/Intel Thread Director structure.

16.6.7 Hardware Feedback Interface and Intel® Thread Director Structure Dynamic Update

The HFI/Intel Thread Director structure can be updated dynamically during run time. Changes to the structure may occur to one or more of its cells. Such changes may occur for one or more logical processors. The hardware sets a non-zero value in the “capability change” field of the HFI/Intel Thread Director structure as an indication for the OS to read that capability for all logical processors. A thermal interrupt is delivered to indicate to the OS that the structure has just changed. Section 16.6.6 contains more details on this notification mechanism. The hardware clears all “capability change” fields after the OS resets IA32_PACKAGE_THERM_STATUS[bit 26].

Zeroing a performance or energy efficiency cell hints to the OS that it is beneficial not to schedule software threads of that class on the associated logical processor for performance or energy efficiency reasons, respectively. If SMT is supported, it may be the case that the hardware zeroes one of the core's logical processors only. Zeroing the performance and energy efficiency cells of all classes for a logical processor implies that the hardware provides a hint to the OS to completely avoid scheduling work on that logical processor.

Zeroing a performance and energy efficiency cell hint of a logical processor across all classes along with Capability Flag bit 1 set to 1 across all capabilities and classes, indicates to the OS to force idle logical processor(s), and if affinity activity occurs on those logical processor(s), the OS should inject idle periods such that overall utilization of those idled cores has a minimal-to-no impact to power. Capability Flag bit 1 will be set to 1 while this hint persists.

When EE=255 is set on one or more logical processors, it represents a request that the OS attempt to consolidate work to those logical processors with EE=255. These requests are made when the SOC has knowledge that consolidating the work to a subset of cores will result in significantly better platform energy efficiency. Examples of consolidating work would include, but not limited to, delaying less important work as needed to provide compute bandwidth for more important work, and routing interrupts to the logical processors with EE=255. When the cumulative workload requires performance greater than that which is available on the subset of cores with EE=255, it is expected that the OS will scale the work out to additional logical processors.

A few example reasons for runtime changes in the HGS/Intel Thread Director Table:

- Over clocking run time update that changes the capability values.
- Change in run time physical constraints.
- Run time performance or energy efficiency optimization.
- Change in core frequency, voltage, or power budget.

16.6.8 Logical Processor Scope Intel® Thread Director Configuration

The operating system enables Intel Thread Director at the logical processor scope using a logical processor scope MSR named IA32_HW_FEEDBACK_THREAD_CONFIG (address 17D4H).

The MSR is read/write and is structured as follows:

- Bits 63:1 – Reserved.
- Bit 0 – Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).

Bit 0 of the logical processor scope configuration MSR can be cleared or set regardless of the state of the HFI/Intel Thread Director package configuration MSR state. Even when bit 0 of all logical processor configuration MSRs is clear, the processor can still update the Intel Thread Director structure if it is still enabled in the IA32_HW_FEEDBACK_CONFIG package scope MSR. When the operating system clears IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0], hardware clears the history accumulated on that logical processor which otherwise drives assigning the Class ID to the software thread that executes on that logical processor. As long as IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set, the Class ID is available for the operating system to read, independent of the state of the package scope IA32_HW_FEEDBACK_CONFIG[1:0] bits.

See Section 16.6.9 for details on scenarios where IA32_HW_FEEDBACK_CONFIG bits are implicitly reset by the hardware.

16.6.9 Implicit Reset of Package and Logical Processor Scope Configuration MSRs

HFI/Intel Thread Director enable bits are reset by hardware in the following scenarios:

1. When GETSEC[SENDER] is executed:
 - a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on all sockets (packages) in the system.
 - b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors in the system across all sockets.
 - c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR across all sockets.
2. When GETSEC[ENTERACCS] is executed:
 - a. The processor implicitly resets the HFI/Intel Thread Director enable bits in the IA32_HW_FEEDBACK_CONFIG MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.
 - b. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on all logical processors on the socket where the GETSEC[ENTERACCS] instruction was executed.
 - c. The processor implicitly clears the HFI/Intel Thread Director table structure pointer in the IA32_HW_FEEDBACK_PTR package MSR on the socket where the GETSEC[ENTERACCS] instruction was executed.
3. When an INIT or a wait-for-SIPI state are processed by a logical processor:
 - a. The processor implicitly resets the Intel Thread Director enable bit in the IA32_HW_FEEDBACK_THREAD_CONFIG MSR on that logical processor, whether the signal was in the context of GETSEC[ENTERACCS] or not.

If the OS requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

16.6.10 Logical Processor Scope Intel® Thread Director Run Time Characteristics

The processor provides the operating system with run time feedback about the execution characteristics of the software thread executing on logical processors whose IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] is set.

The run time feedback is communicated via a read-only MSR named IA32_THREAD_FEEDBACK_CHAR. This is a logical processor scope MSR whose address is 17D2H. This MSR is structured as follows:

- Bit 63 – Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.
- Bits 62:8 – Reserved.
- Bits 7:0 – Application Class ID, pointing into the Intel Thread Director structure described in Table 16-8.

This MSR is valid only if CPUID.06H:EAX[23] is set.

The valid bit is cleared by the hardware in the following cases:

- The hardware does not have enough information to provide the operating system with a reliable Class ID.
- The operating system cleared the logical processor's IA32_HW_FEEDBACK_THREAD_CONFIG[bit 0] bit.

The HRESET instruction is executed while configured to reset the Intel Thread Director history.

16.6.11 Logical Processor Scope History

The operating system can reset the Intel Thread Director related history accumulated on the current logical processor it is executing on by issuing the HRESET instruction. See “CPUID—CPU Identification” in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A, for enumeration of the HRESET

instruction. See also the “HRESET—History Reset” instruction description in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

16.6.11.1 Enabling Intel® Thread Director History Reset

The IA32_HRESET_ENABLE MSR is a read/write MSR and is structured as follows:

- Bits 63:32 – Reserved.
- Bits 31:1 – Reserved for other capabilities that can be reset by the HRESET instruction.
- Bit 0 – Enable reset of the Intel Thread Director history.

The operating system should set IA32_HRESET_ENABLE[bit 0] to enable Intel Thread Director history reset via the HRESET instruction.

16.6.11.2 Implicit Intel® Thread Director History Reset

The Intel Thread Director history is implicitly reset in the following scenarios:

1. When the processor enters or exits SMM mode and IA32_DEBUGCTL MSR.FREEZE_WHILE_SMM (bit 14) is set, the Intel Thread Director history is implicitly reset by the processor.
2. When GETSEC[SENDER] is executed, the processor resets the Intel Thread Director history on all logical processors in the system, including logical processors on other sockets (other than the one GETSEC[SENDER] is executed).
3. When GETSEC[ENTERACCS] is executed, the processor resets the Intel Thread Director history on the logical processor it is executed on.
4. When an INIT or a wait-for-SIPI state are processed by a logical processor, the Intel Thread Director history is reset whether the signal was a result of GETSEC[ENTERACCS] or not.

If the operating system requires HFI/Intel Thread Director to be active after exiting the measured environment or when processing a SIPI event, it should re-enable HFI/Intel Thread Director.

16.7 MWAIT EXTENSIONS FOR ADVANCED POWER MANAGEMENT

IA-32 processors may support a number of C-states¹ that reduce power consumption for inactive states. Intel Core Solo and Intel Core Duo processors support both deeper C-state and MWAIT extensions that can be used by OS to implement power management policy.

Software should use CPUID to discover if a target processor supports the enumeration of MWAIT extensions. If CPUID.05H:ECX[0] = 1, the target processor supports MWAIT extensions and their enumeration (see Chapter 4, “Instruction Set Reference, M-U,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B).

If CPUID.05H:ECX[1] = 1, the target processor supports using interrupts as break-events for MWAIT, even when interrupts are disabled. Use this feature to measure C-state residency as follows:

- Software can write to bit 0 in the MWAIT Extensions register (ECX) when issuing an MWAIT to enter into a processor-specific C-state or sub C-state.
- When a processor comes out of an inactive C-state or sub C-state, software can read a timestamp before an interrupt service routine (ISR) is potentially executed.

CPUID.05H:EDX allows software to enumerate processor-specific C-states and sub C-states available for use with MWAIT extensions. IA-32 processors may support more than one C-state of a given C-state type. These are called sub C-states. Numerically higher C-state have higher power savings and latency (upon entering and exiting) than lower-numbered C-state.

1. The processor-specific C-states defined in MWAIT extensions can map to ACPI defined C-state types (C0, C1, C2, C3). The mapping relationship depends on the definition of a C-state by processor implementation and is exposed to OSPM by the BIOS using the ACPI defined _CST table.

At CPL = 0, system software can specify desired C-state and sub C-state by using the MWAIT hints register (EAX). Processors will not go to C-state and sub C-state deeper than what is specified by the hint register. If CPL > 0 and if MONITOR/MWAIT is supported at CPL > 0, the processor will only enter C1-state (regardless of the C-state request in the hints register).

Executing MWAIT generates an exception on processors operating at a privilege level where MONITOR/MWAIT are not supported.

NOTE

If MWAIT is used to enter a C-state (including sub C-state) that is numerically higher than C1, a store to the address range armed by MONITOR instruction will cause the processor to exit MWAIT if the store was originated by other processor agents. A store from non-processor agent may not cause the processor to exit MWAIT.

16.8 THERMAL MONITORING AND PROTECTION

The IA-32 architecture provides the following mechanisms for monitoring temperature and controlling thermal power:

1. The **catastrophic shutdown detector** forces processor execution to stop if the processor's core temperature rises above a preset limit.
2. **Automatic and adaptive thermal monitoring mechanisms** force the processor to reduce its power consumption in order to operate within predetermined temperature limits.
3. The **software controlled clock modulation mechanism** permits operating systems to implement power management policies that reduce power consumption; this is in addition to the reduction offered by automatic thermal monitoring mechanisms.
4. **On-die digital thermal sensor and interrupt mechanisms** permit the OS to manage thermal conditions natively without relying on BIOS or other system board components.

The first mechanism is not visible to software. The other three mechanisms are visible to software using processor feature information returned by executing CPUID with EAX = 1.

The second mechanism includes:

- **Automatic thermal monitoring** provides two modes of operation. One mode modulates the clock duty cycle; the second mode changes the processor's frequency. Both modes are used to control the core temperature of the processor.
- **Adaptive thermal monitoring** can provide flexible thermal management on processors made of multiple cores.

The third mechanism modulates the clock duty cycle of the processor. As shown in Figure 16-24, the phrase 'duty cycle' does not refer to the actual duty cycle of the clock signal. Instead it refers to the time period during which the clock signal is allowed to drive the processor chip. By using the stop clock mechanism to control how often the processor is clocked, processor power consumption can be modulated.

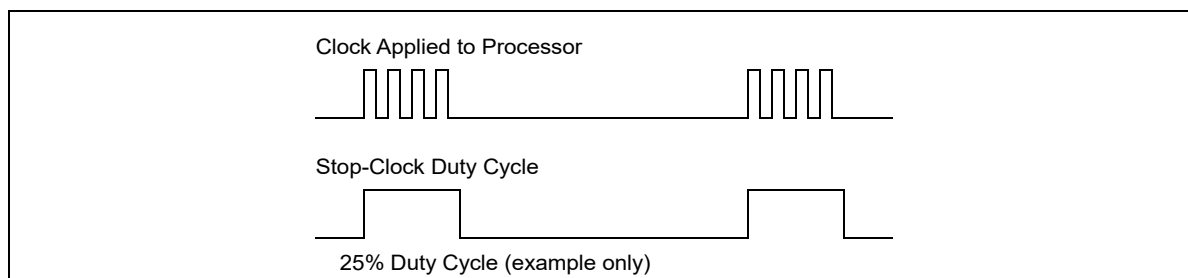


Figure 16-24. Processor Modulation Through Stop-Clock Mechanism

For previous automatic thermal monitoring mechanisms, software controlled mechanisms that changed processor operating parameters to impact changes in thermal conditions. Software did not have native access to the native thermal condition of the processor; nor could software alter the trigger condition that initiated software program control.

The fourth mechanism (listed above) provides access to an on-die digital thermal sensor using a model-specific register and uses an interrupt mechanism to alert software to initiate digital thermal monitoring.

16.8.1 Catastrophic Shutdown Detector

P6 family processors introduced a thermal sensor that acts as a catastrophic shutdown detector. This catastrophic shutdown detector was also implemented in Pentium 4, Intel Xeon and Pentium M processors. It is always enabled. When processor core temperature reaches a factory preset level, the sensor trips and processor execution is halted until after the next reset cycle.

16.8.2 Thermal Monitor

Pentium 4, Intel Xeon and Pentium M processors introduced a second temperature sensor that is factory-calibrated to trip when the processor's core temperature crosses a level corresponding to the recommended thermal design envelop. The trip-temperature of the second sensor is calibrated below the temperature assigned to the catastrophic shutdown detector.

16.8.2.1 Thermal Monitor 1

The Pentium 4 processor uses the second temperature sensor in conjunction with a mechanism called Thermal Monitor 1 (TM1) to control the core temperature of the processor. TM1 controls the processor's temperature by modulating the duty cycle of the processor clock. Modulation of duty cycles is processor model specific. Note that the processors STPCLK# pin is not used here; the stop-clock circuitry is controlled internally.

Support for TM1 is indicated by CPUID.01H:EDX.TM[29] = 1.

TM1 is enabled by setting the thermal-monitor enable flag (bit 3) in IA32_MISC_ENABLE; see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. Following a power-up or reset, the flag is cleared, disabling TM1. BIOS is required to enable only one automatic thermal monitoring modes. Operating systems and applications must not disable the operation of these mechanisms.

16.8.2.2 Thermal Monitor 2

An additional automatic thermal protection mechanism, called Thermal Monitor 2 (TM2), was introduced in the Intel Pentium M processor and also incorporated in newer models of the Pentium 4 processor family. Intel Core Duo and Solo processors, and Intel Core 2 Duo processor family all support TM1 and TM2. TM2 controls the core temperature of the processor by reducing the operating frequency and voltage of the processor and offers a higher performance level for a given level of power reduction than TM1.

TM2 is triggered by the same temperature sensor as TM1. The mechanism to enable TM2 may be implemented differently across various IA-32 processor families with different CPUID signatures in the family encoding value, but will be uniform within an IA-32 processor family.

Support for TM2 is indicated by CPUID.01H:ECX.TM2[8] = 1.

16.8.2.3 Two Methods for Enabling TM2

On processors with CPUID family/model/stepping signature encoded as 0x69n or 0x6Dn (early Pentium M processors), TM2 is enabled if the TM_SELECT flag (bit 16) of the MSR_THERM2_CTL register is set to 1 (Figure 16-25) and bit 3 of the IA32_MISC_ENABLE register is set to 1.

Following a power-up or reset, the TM_SELECT flag may be cleared. BIOS is required to enable either TM1 or TM2. Operating systems and applications must not disable mechanisms that enable TM1 or TM2. If bit 3 of the IA32_MISC_ENABLE register is set and TM_SELECT flag of the MSR_THERM2_CTL register is cleared, TM1 is enabled.

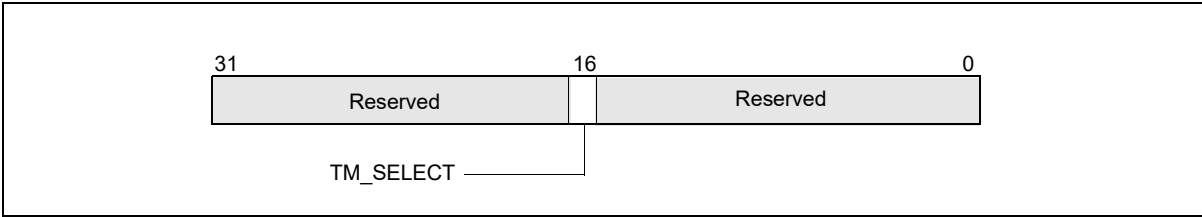


Figure 16-25. MSR_THERM2_CTL Register On Processors with CUID Family/Model/Stepping Signature Encoded as 0x69n or 0x6Dn

On processors introduced after the Pentium 4 processor (this includes most Pentium M processors), the method used to enable TM2 is different. TM2 is enable by setting bit 13 of IA32_MISC_ENABLE register to 1. This applies to Intel Core Duo, Core Solo, and Intel Core 2 processor family.

The target operating frequency and voltage for the TM2 transition after TM2 is triggered is specified by the value written to MSR_THERM2_CTL, bits 15:0 (Figure 16-26). Following a power-up or reset, BIOS is required to enable at least one of these two thermal monitoring mechanisms. If both TM1 and TM2 are supported, BIOS may choose to enable TM2 instead of TM1. Operating systems and applications must not disable the mechanisms that enable TM1or TM2; and they must not alter the value in bits 15:0 of the MSR_THERM2_CTL register.

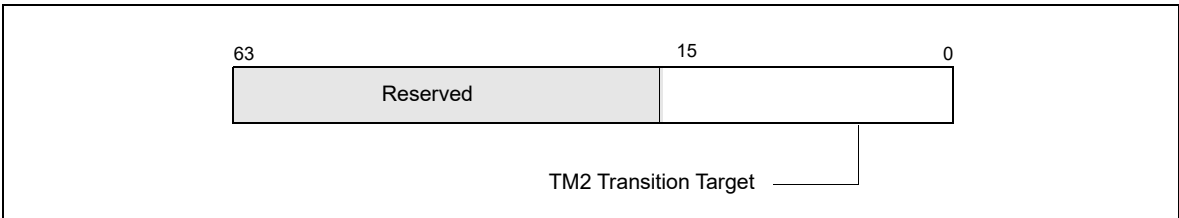


Figure 16-26. MSR_THERM2_CTL Register for Supporting TM2

16.8.2.4 Performance State Transitions and Thermal Monitoring

If the thermal control circuitry (TCC) for thermal monitor (TM1/TM2) is active, writes to the IA32_PERF_CTL will effect a new target operating point as follows:

- If TM1 is enabled and the TCC is engaged, the performance state transition can commence before the TCC is disengaged.
- If TM2 is enabled and the TCC is engaged, the performance state transition specified by a write to the IA32_PERF_CTL will commence after the TCC has disengaged.

16.8.2.5 Thermal Status Information

The status of the temperature sensor that triggers the thermal monitor (TM1/TM2) is indicated through the thermal status flag and thermal status log flag in the IA32_THERM_STATUS MSR (see Figure 16-27).

The functions of these flags are:

- **Thermal Status flag, bit 0** — When set, indicates that the processor core temperature is currently at the trip temperature of the thermal monitor and that the processor power consumption is being reduced via either TM1 or TM2, depending on which is enabled. When clear, the flag indicates that the core temperature is below the thermal monitor trip temperature. This flag is read only.

- **Thermal Status Log flag, bit 1** — When set, indicates that the thermal sensor has tripped since the last power-up or reset or since the last time that software cleared this flag. This flag is a sticky bit; once set it remains set until cleared by software or until a power-up or reset of the processor. The default state is clear.

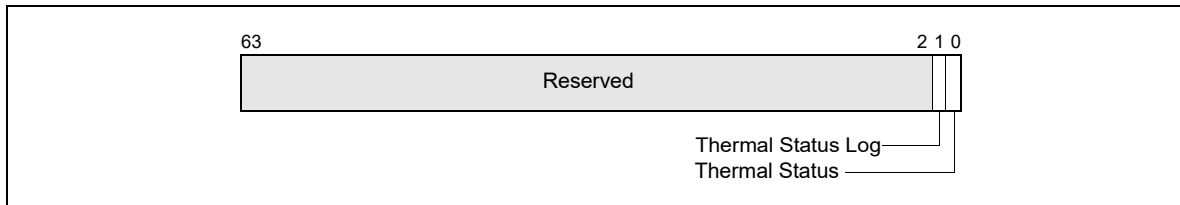


Figure 16-27. IA32_THERM_STATUS MSR

After the second temperature sensor has been tripped, the thermal monitor (TM1/TM2) will remain engaged for a minimum time period (on the order of 1 ms). The thermal monitor will remain engaged until the processor core temperature drops below the preset trip temperature of the temperature sensor, taking hysteresis into account.

While the processor is in a stop-clock state, interrupts will be blocked from interrupting the processor. This holding off of interrupts increases the interrupt latency, but does not cause interrupts to be lost. Outstanding interrupts remain pending until clock modulation is complete.

The thermal monitor can be programmed to generate an interrupt to the processor when the thermal sensor is tripped; this is called a thermal interrupt. The delivery mode, mask, and vector for this interrupt can be programmed through the thermal entry in the local APIC's LVT (see Section 12.5.1, "Local Vector Table"). The low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR (see Figure 16-28) control when the interrupt is generated; that is, on a transition from a temperature below the trip point to above and/or vice-versa.

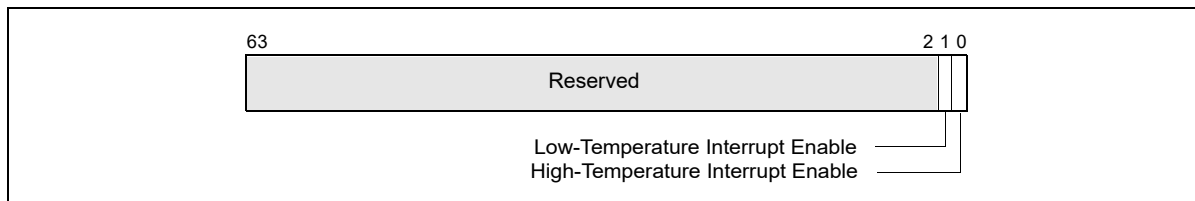


Figure 16-28. IA32_THERM_INTERRUPT MSR

- **High-Temperature Interrupt Enable flag, bit 0** — Enables an interrupt to be generated on the transition from a low-temperature to a high-temperature when set; disables the interrupt when clear.(R/W).
- **Low-Temperature Interrupt Enable flag, bit 1** — Enables an interrupt to be generated on the transition from a high-temperature to a low-temperature when set; disables the interrupt when clear.

The thermal interrupt can be masked by the thermal LVT entry. After a power-up or reset, the low-temperature interrupt enable and high-temperature interrupt enable flags in the IA32_THERM_INTERRUPT MSR are cleared (interrupts are disabled) and the thermal LVT entry is set to mask interrupts. This interrupt should be handled either by the operating system or system management mode (SMM) code.

Note that the operation of the thermal monitoring mechanism has no effect upon the clock rate of the processor's internal high-resolution timer (time stamp counter).

16.8.2.6 Adaptive Thermal Monitor

The Intel Core 2 Duo processor family supports enhanced thermal management mechanism, referred to as Adaptive Thermal Monitor (Adaptive TM).

Unlike TM2, Adaptive TM is not limited to one TM2 transition target. During a thermal trip event, Adaptive TM (if enabled) selects an optimal target operating point based on whether or not the current operating point has effectively cooled the processor.

Similar to TM2, Adaptive TM is enable by BIOS. The BIOS is required to test the TM1 and TM2 feature flags and enable all available thermal control mechanisms (including Adaptive TM) at platform initiation.

Adaptive TM is available only to a subset of processors that support TM2.

In each chip-multiprocessing (CMP) silicon die, each core has a unique thermal sensor that triggers independently. These thermal sensor can trigger TM1 or TM2 transitions in the same manner as described in Section 16.8.2.1 and Section 16.8.2.2. The trip point of the thermal sensor is not programmable by software since it is set during the fabrication of the processor.

Each thermal sensor in a processor core may be triggered independently to engage thermal management features. In Adaptive TM, both cores will transition to a lower frequency and/or lower voltage level if one sensor is triggered. Triggering of this sensor is visible to software via the thermal interrupt LVT entry in the local APIC of a given core.

16.8.3 Software Controlled Clock Modulation

Pentium 4, Intel Xeon and Pentium M processors also support software-controlled clock modulation. This provides a means for operating systems to implement a power management policy to reduce the power consumption of the processor. Here, the stop-clock duty cycle is controlled by software through the IA32_CLOCK_MODULATION MSR (see Figure 16-29).

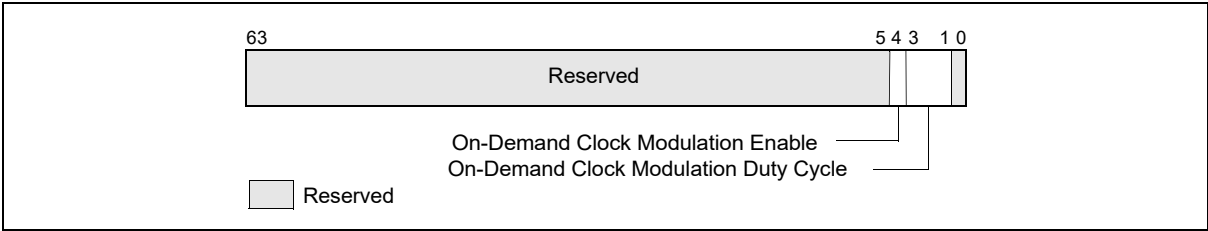


Figure 16-29. IA32_CLOCK_MODULATION MSR

The IA32_CLOCK_MODULATION MSR contains the following flag and field used to enable software-controlled clock modulation and to select the clock modulation duty cycle:

- **On-Demand Clock Modulation Enable, bit 4** — Enables on-demand software controlled clock modulation when set; disables software-controlled clock modulation when clear.
- **On-Demand Clock Modulation Duty Cycle, bits 1 through 3** — Selects the on-demand clock modulation duty cycle (see Table 16-11). This field is only active when the on-demand clock modulation enable flag is set.

Note that the on-demand clock modulation mechanism (like the thermal monitor) controls the processor’s stop-clock circuitry internally to modulate the clock signal. The STPCLK# pin is not used in this mechanism.

Table 16-11. On-Demand Clock Modulation Duty Cycle Field Encoding

Duty Cycle Field Encoding	Duty Cycle
000B	Reserved
001B	12.5% (Default)
010B	25.0%
011B	37.5%
100B	50.0%
101B	63.5%
110B	75%
111B	87.5%

The on-demand clock modulation mechanism can be used to control processor power consumption. Power management software can write to the IA32_CLOCK_MODULATION MSR to enable clock modulation and to select a modulation duty cycle. If on-demand clock modulation and TM1 are both enabled and the thermal status of the processor is hot (bit 0 of the IA32_THERM_STATUS MSR is set), clock modulation at the duty cycle specified by TM1 takes precedence, regardless of the setting of the on-demand clock modulation duty cycle.

For Hyper-Threading Technology enabled processors, the IA32_CLOCK_MODULATION register is duplicated for each logical processor. In order for the On-demand clock modulation feature to work properly, the feature must be enabled on all the logical processors within a physical processor. If the programmed duty cycle is not identical for all the logical processors, the processor core clock will modulate to the highest duty cycle programmed for processors with any of the following CPUID DisplayFamily_DisplayModel signatures (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A): 06_1A, 06_1C, 06_1E, 06_1F, 06_25, 06_26, 06_27, 06_2C, 06_2E, 06_2F, 06_35, 06_36, and 0F_xx. For all other processors, if the programmed duty cycle is not identical for all logical processors in the same core, the processor core will modulate at the lowest programmed duty cycle.

For multiple processor cores in a physical package, each processor core can modulate to a programmed duty cycle independently.

For the P6 family processors, on-demand clock modulation was implemented through the chipset, which controlled clock modulation through the processor's STPCLK# pin.

16.8.3.1 Extension of Software Controlled Clock Modulation

Extension of the software controlled clock modulation facility supports on-demand clock modulation duty cycle with 4-bit dynamic range (increased from 3-bit range). Granularity of clock modulation duty cycle is increased to 6.25% (compared to 12.5%).

Four bit dynamic range control is provided by using bit 0 in conjunction with bits 3:1 of the IA32_CLOCK_MODULATION MSR (see Figure 16-30).

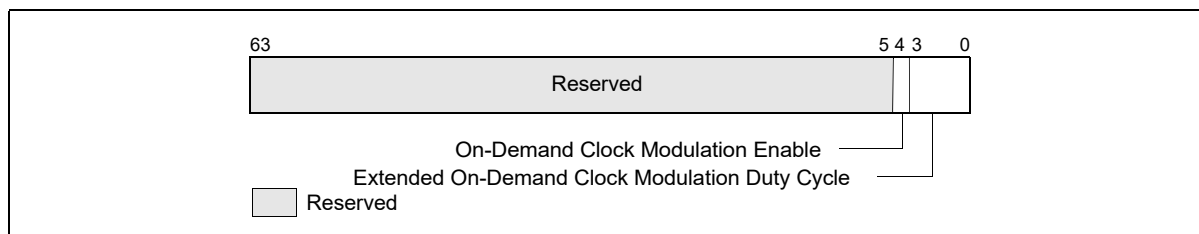


Figure 16-30. IA32_CLOCK_MODULATION MSR with Clock Modulation Extension

Extension to software controlled clock modulation is supported only if CPUID.06H:EAX[5] = 1. If CPUID.06H:EAX[5] = 0, then bit 0 of IA32_CLOCK_MODULATION is reserved.

16.8.4 Detection of Thermal Monitor and Software Controlled Clock Modulation Facilities

The ACPI flag (bit 22) of the CPUID feature flags indicates the presence of the IA32_THERM_STATUS, IA32_THERM_INTERRUPT, IA32_CLOCK_MODULATION MSRs, and the xAPIC thermal LVT entry.

The TM1 flag (bit 29) of the CPUID feature flags indicates the presence of the automatic thermal monitoring facilities that modulate clock duty cycles.

16.8.4.1 Detection of Software Controlled Clock Modulation Extension

Processor's support of software controlled clock modulation extension is indicated by CPUID.06H:EAX[5] = 1.

16.8.5 On Die Digital Thermal Sensors

On die digital thermal sensor can be read using an MSR (no I/O interface). In Intel Core Duo processors, each core has a unique digital sensor whose temperature is accessible using an MSR. The digital thermal sensor is the preferred method for reading the die temperature because (a) it is located closer to the hottest portions of the die, (b) it enables software to accurately track the die temperature and the potential activation of thermal throttling.

16.8.5.1 Digital Thermal Sensor Enumeration

The processor supports a digital thermal sensor if CPUID.06H:EAX[0] = 1. If the processor supports digital thermal sensor, EBX[bits 3:0] determine the number of thermal thresholds that are available for use.

Software sets thermal thresholds by using the IA32_THERM_INTERRUPT MSR. Software reads output of the digital thermal sensor using the IA32_THERM_STATUS MSR.

16.8.5.2 Reading the Digital Sensor

Unlike traditional analog thermal devices, the output of the digital thermal sensor is a temperature relative to the maximum supported operating temperature of the processor.

Temperature measurements returned by digital thermal sensors are always at or below TCC activation temperature. Critical temperature conditions are detected using the “Critical Temperature Status” bit. When this bit is set, the processor is operating at a critical temperature and immediate shutdown of the system should occur. Once the “Critical Temperature Status” bit is set, reliable operation is not guaranteed.

See Figure 16-31 for the layout of IA32_THERM_STATUS MSR. Bit fields include:

- **Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (PROCHOT#) is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (PROCHOT#). Bit 1 = 1 if PROCHOT# has been asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.
- **PROCHOT# or FORCEPR# Event (bit 2, RO)** — Indicates whether PROCHOT# or FORCEPR# is being asserted by another agent on the platform.

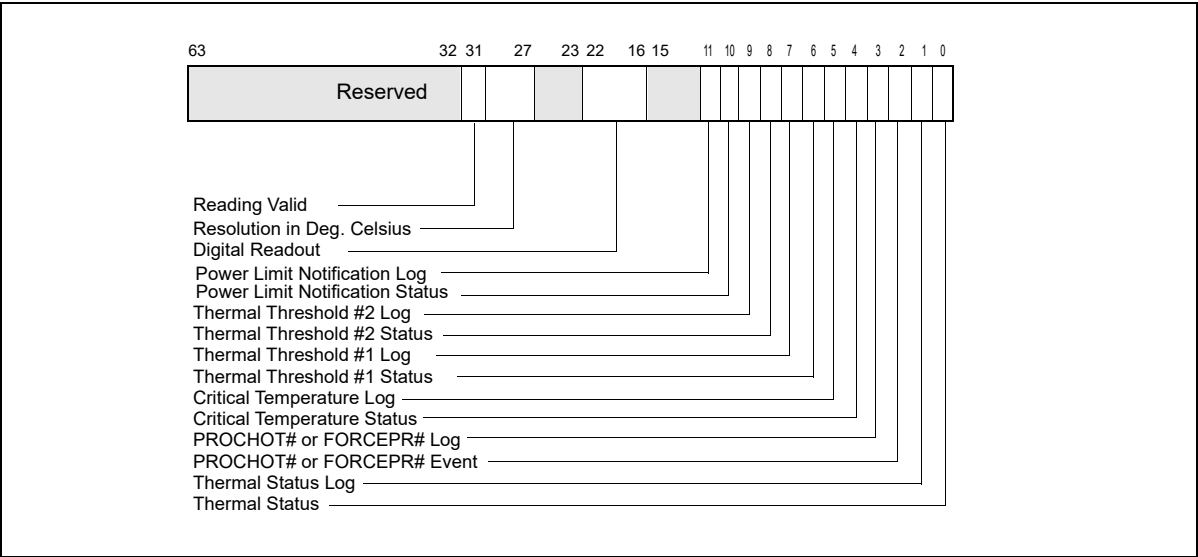


Figure 16-31. IA32_THERM_STATUS Register

- **PROCHOT# or FORCEPR# Log (bit 3, R/WC0)** — Sticky bit that indicates whether PROCHOT# or FORCEPR# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, PROCHOT# or FORCEPR# has been externally asserted. Software may clear this bit by writing a zero. External PROCHOT# assertions are only acknowledged if the Bidirectional Prochot feature is enabled.
- **Critical Temperature Status (bit 4, RO)** — Indicates whether the critical temperature detector output signal is currently active. If bit 4 = 1, the critical temperature detector output signal is currently active.
- **Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual temperature is currently higher than or equal to the value set in Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to TT#1. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- **Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual temperature is currently higher than or equal to the value set in Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to TT#2. Quantitative information of actual temperature can be inferred from Digital Readout, bits 22:16.
- **Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Power Limitation Status (bit 10, RO)** — Indicates whether the processor is currently operating below OS-requested P-state (specified in IA32_PERF_CTL) or OS-requested clock modulation duty cycle (specified in IA32_CLOCK_MODULATION). This field is supported only if CPUID.06H:EAX[4] = 1. Package level power limit notification can be delivered independently to IA32_PACKAGE_THERM_STATUS MSR.
- **Power Notification Log (bit 11, R/WC0)** — Sticky bit that indicates the processor went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET. This field is supported only if CPUID.06H:EAX[4] = 1. Package level power limit notification is indicated independently in IA32_PACKAGE_THERM_STATUS MSR.
- **Digital Readout (bits 22:16, RO)** — Digital temperature reading in 1 degree Celsius relative to the TCC activation temperature.
0: TCC Activation temperature,
1: (TCC Activation - 1) , etc. See the processor's data sheet for details regarding TCC activation.
A lower reading in the Digital Readout field (bits 22:16) indicates a higher actual temperature.
- **Resolution in Degrees Celsius (bits 30:27, RO)** — Specifies the resolution (or tolerance) of the digital thermal sensor. The value is in degrees Celsius. It is recommended that new threshold values be offset from the current temperature by at least the resolution + 1 in order to avoid hysteresis of interrupt generation.
- **Reading Valid (bit 31, RO)** — Indicates if the digital readout in bits 22:16 is valid. The readout is valid if bit 31 = 1.

Changes to temperature can be detected using two thresholds (see Figure 16-32); one is set above and the other below the current temperature. These thresholds have the capability of generating interrupts using the core's local APIC which software must then service. Note that the local APIC entries used by these thresholds are also used by the Intel® Thermal Monitor; it is up to software to determine the source of a specific interrupt.

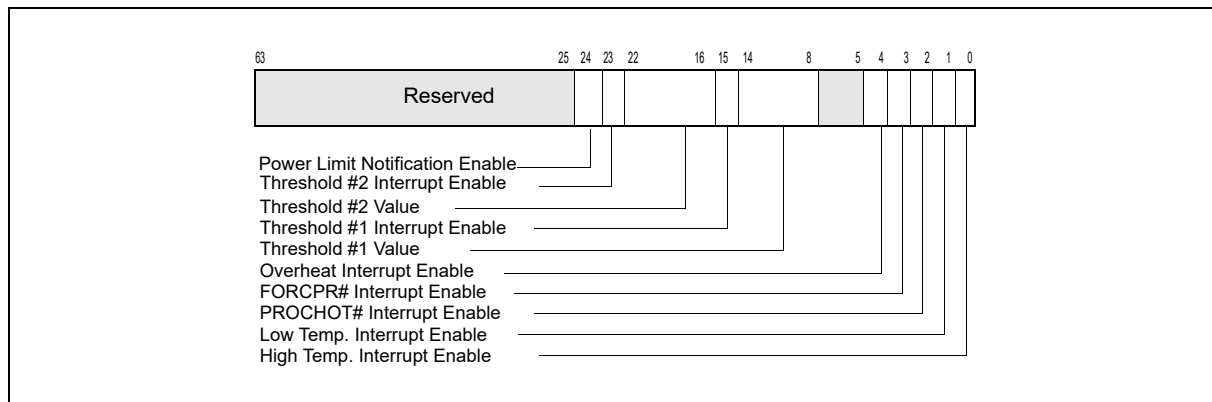


Figure 16-32. IA32_THERM_INTERRUPT Register

See Figure 16-32 for the layout of IA32_THERM_INTERRUPT MSR. Bit fields include:

- **High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **FORCEPR# Interrupt Enable (bit 3, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when FORCEPR# has been asserted by another agent on the platform. Bit 3 = 0 disables the interrupt; bit 3 = 1 enables the interrupt.
- **Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #1 Status and Log bits as well as the Threshold #1 thermal interrupt delivery.
- **Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, encoded relative to the TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Digital Readout and is used to generate the Thermal Threshold #2 Status and Log bits as well as the Threshold #2 thermal interrupt delivery.
- **Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of power notification events when the processor went below OS-requested P-state or OS-requested clock modulation duty cycle. This field is supported only if CPUID.06H:EAX[4] = 1. Package level power limit notification can be enabled independently by IA32_PACKAGE_THERM_INTERRUPT MSR.

16.8.6 Power Limit Notification

Platform firmware may be capable of specifying a power limit to restrict power delivered to a platform component, such as a physical processor package. This constraint imposed by platform firmware may occasionally cause the processor to operate below OS-requested P or T-state. A power limit notification event can be delivered using the existing thermal LVT entry in the local APIC.

Software can enumerate the presence of the processor's support for power limit notification by verifying `CPUID.06H:EAX[4] = 1`.

If `CPUID.06H:EAX[4] = 1`, then `IA32_THERM_INTERRUPT` and `IA32_THERM_STATUS` provides the following facility to manage power limit notification:

- Bits 10 and 11 in `IA32_THERM_STATUS` informs software of the occurrence of processor operating below OS-requested P-state or clock modulation duty cycle setting (see Figure 16-31).
- Bit 24 in `IA32_THERM_INTERRUPT` enables the local APIC to deliver a thermal event when the processor went below OS-requested P-state or clock modulation duty cycle setting (see Figure 16-32).

16.9 PACKAGE LEVEL THERMAL MANAGEMENT

The thermal management facilities like `IA32_THERM_INTERRUPT` and `IA32_THERM_STATUS` are often implemented with a processor core granularity. To facilitate software manage thermal events from a package level granularity, two architectural MSR is provided for package level thermal management. The `IA32_PACKAGE_THERM_STATUS` and `IA32_PACKAGE_THERM_INTERRUPT` MSRs use similar interfaces as `IA32_THERM_STATUS` and `IA32_THERM_INTERRUPT`, but are shared in each physical processor package.

Software can enumerate the presence of the processor's support for package level thermal management facility (`IA32_PACKAGE_THERM_STATUS` and `IA32_PACKAGE_THERM_INTERRUPT`) by verifying `CPUID.06H:EAX[6] = 1`.

The layout of `IA32_PACKAGE_THERM_STATUS` MSR is shown in Figure 16-33.

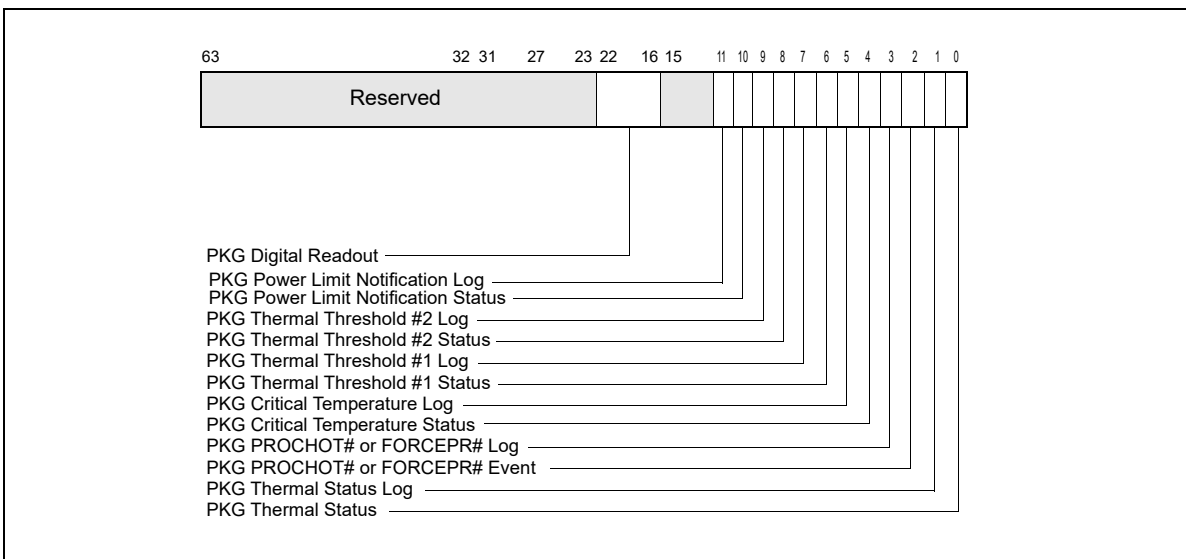


Figure 16-33. `IA32_PACKAGE_THERM_STATUS` Register

- **Package Thermal Status (bit 0, RO)** — This bit indicates whether the digital thermal sensor high-temperature output signal (`PROCHOT#`) for the package is currently active. Bit 0 = 1 indicates the feature is active. This bit may not be written by software; it reflects the state of the digital thermal sensor.
- **Package Thermal Status Log (bit 1, R/WC0)** — This is a sticky bit that indicates the history of the thermal sensor high temperature output signal (`PROCHOT#`) of the package. Bit 1 = 1 if package `PROCHOT#` has been

asserted since a previous RESET or the last time software cleared the bit. Software may clear this bit by writing a zero.

- **Package PROCHOT# Event (bit 2, RO)** — Indicates whether package PROCHOT# is being asserted by another agent on the platform.
- **Package PROCHOT# Log (bit 3, R/WC0)** — Sticky bit that indicates whether package PROCHOT# has been asserted by another agent on the platform since the last clearing of this bit or a reset. If bit 3 = 1, package PROCHOT# has been externally asserted. Software may clear this bit by writing a zero.
- **Package Critical Temperature Status (bit 4, RO)** — Indicates whether the package critical temperature detector output signal is currently active. If bit 4 = 1, the package critical temperature detector output signal is currently active.
- **Package Critical Temperature Log (bit 5, R/WC0)** — Sticky bit that indicates whether the package critical temperature detector output signal has been asserted since the last clearing of this bit or reset. If bit 5 = 1, the output signal has been asserted. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #1 Status (bit 6, RO)** — Indicates whether the actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #1. If bit 6 = 0, the actual temperature is lower. If bit 6 = 1, the actual temperature is greater than or equal to PTT#1. Quantitative information of actual package temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #1 Log (bit 7, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #1 has been reached since the last clearing of this bit or a reset. If bit 7 = 1, the Package Thermal Threshold #1 has been reached. Software may clear this bit by writing a zero.
- **Package Thermal Threshold #2 Status (bit 8, RO)** — Indicates whether actual package temperature is currently higher than or equal to the value set in Package Thermal Threshold #2. If bit 8 = 0, the actual temperature is lower. If bit 8 = 1, the actual temperature is greater than or equal to PTT#2. Quantitative information of actual temperature can be inferred from Package Digital Readout, bits 22:16.
- **Package Thermal Threshold #2 Log (bit 9, R/WC0)** — Sticky bit that indicates whether the Package Thermal Threshold #2 has been reached since the last clearing of this bit or a reset. If bit 9 = 1, the Package Thermal Threshold #2 has been reached. Software may clear this bit by writing a zero.
- **Package Power Limitation Status (bit 10, RO)** — Indicates package power limit is forcing one or more processors to operate below OS-requested P-state. Note that package power limit violation may be caused by processor cores or by devices residing in the uncore. Software can examine IA32_THERM_STATUS to determine if the cause originates from a processor core (see Figure 16-31).
- **Package Power Notification Log (bit 11, R/WC0)** — Sticky bit that indicates any processor in the package went below OS-requested P-state or OS-requested clock modulation duty cycle since the last clearing of this or RESET.
- **Package Digital Readout (bits 22:16, RO)** — Package digital temperature reading in 1 degree Celsius relative to the package TCC activation temperature.

0: Package TCC Activation temperature,

1: (PTCC Activation - 1) , etc. See the processor's data sheet for details regarding PTCC activation.

A lower reading in the Package Digital Readout field (bits 22:16) indicates a higher actual temperature.

The layout of IA32_PACKAGE_THERM_INTERRUPT MSR is shown in Figure 16-34.

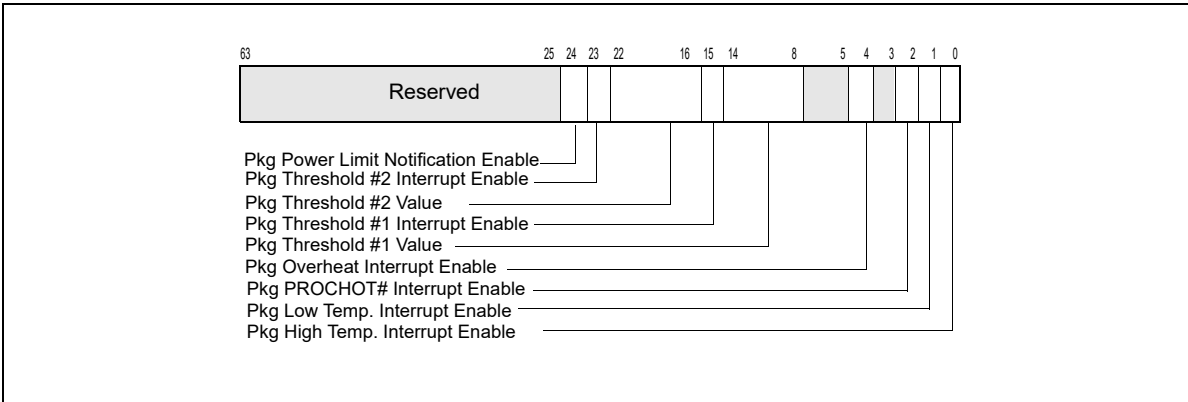


Figure 16-34. IA32_PACKAGE_THERM_INTERRUPT Register

- **Package High-Temperature Interrupt Enable (bit 0, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from low-temperature to a package high-temperature threshold. Bit 0 = 0 (default) disables interrupts; bit 0 = 1 enables interrupts.
- **Package Low-Temperature Interrupt Enable (bit 1, R/W)** — This bit allows the BIOS to enable the generation of an interrupt on the transition from high-temperature to a low-temperature (TCC de-activation). Bit 1 = 0 (default) disables interrupts; bit 1 = 1 enables interrupts.
- **Package PROCHOT# Interrupt Enable (bit 2, R/W)** — This bit allows the BIOS or OS to enable the generation of an interrupt when Package PROCHOT# has been asserted by another agent on the platform and the Bidirectional Prochot feature is enabled. Bit 2 = 0 disables the interrupt; bit 2 = 1 enables the interrupt.
- **Package Critical Temperature Interrupt Enable (bit 4, R/W)** — Enables the generation of an interrupt when the Package Critical Temperature Detector has detected a critical thermal condition. The recommended response to this condition is a system shutdown. Bit 4 = 0 disables the interrupt; bit 4 = 1 enables the interrupt.
- **Package Threshold #1 Value (bits 14:8, R/W)** — A temperature threshold, encoded relative to the Package TCC Activation temperature (using the same format as the Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #1 Status and Log bits as well as the Package Threshold #1 thermal interrupt delivery.
- **Package Threshold #1 Interrupt Enable (bit 15, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #1 setting in any direction. Bit 15 = 1 enables the interrupt; bit 15 = 0 disables the interrupt.
- **Package Threshold #2 Value (bits 22:16, R/W)** — A temperature threshold, encoded relative to the PTCC Activation temperature (using the same format as the Package Digital Readout). This threshold is compared against the Package Digital Readout and is used to generate the Package Thermal Threshold #2 Status and Log bits as well as the Package Threshold #2 thermal interrupt delivery.
- **Package Threshold #2 Interrupt Enable (bit 23, R/W)** — Enables the generation of an interrupt when the actual temperature crosses the Package Threshold #2 setting in any direction. Bit 23 = 1 enables the interrupt; bit 23 = 0 disables the interrupt.
- **Package Power Limit Notification Enable (bit 24, R/W)** — Enables the generation of package power notification events.

16.9.1 Support for Passive and Active cooling

Passive and active cooling may be controlled by the OS power management agent through ACPI control methods. On platforms providing package level thermal management facility described in the previous section, it is recommended that active cooling (FAN control) should be driven by measuring the package temperature using the IA32_PACKAGE_THERM_INTERRUPT MSR.

Passive cooling (frequency throttling) should be driven by measuring (a) the core and package temperatures, or (b) only the package temperature. If measured package temperature led the power management agent to choose which core to execute passive cooling, then all cores need to execute passive cooling. Core temperature is measured using the IA32_THERMAL_STATUS and IA32_THERMAL_INTERRUPT MSRs. The exact implementation details depend on the platform firmware and possible solutions include defining two different thermal zones (one for core temperature and passive cooling and the other for package temperature and active cooling).

16.10 PLATFORM SPECIFIC POWER MANAGEMENT SUPPORT

This section covers power management interfaces that are not architectural but addresses the power management needs of several platform specific components. Specifically, RAPL (Running Average Power Limit) interfaces provide mechanisms to enforce power consumption limit. Power limiting usages have specific usages in client and server platforms.

For client platform power limit control and for server platforms used in a data center, the following power and thermal related usages are desirable:

- Platform Thermal Management: Robust mechanisms to manage component, platform, and group-level thermals, either proactively or reactively (e.g., in response to a platform-level thermal trip point).
- Platform Power Limiting: More deterministic control over the system's power consumption, for example to meet battery life targets on rack-level or container-level power consumption goals within a datacenter.
- Power/Performance Budgeting: Efficient means to control the power consumed (and therefore the sustained performance delivered) within and across platforms.

The server and client usage models are addressed by RAPL interfaces, which expose multiple domains of power rationing within each processor socket. Generally, these RAPL domains may be viewed to include hierarchically:

- Package domain is the processor die.
- Memory domain includes the directly-attached DRAM; an additional power plane may constitute a separate domain.

In order to manage the power consumed across multiple sockets via RAPL, individual limits must be programmed for each processor complex. Programming specific RAPL domain across multiple sockets is not supported.

16.10.1 RAPL Interfaces

RAPL interfaces consist of non-architectural MSRs. Each RAPL domain supports the following set of capabilities, some of which are optional as stated below.

- Power limit - MSR interfaces to specify power limit, time window; lock bit, clamp bit etc.
- Energy Status - Power metering interface providing energy consumption information.
- Perf Status (Optional) - Interface providing information on the performance effects (regression) due to power limits. It is defined as a duration metric that measures the power limit effect in the respective domain. The meaning of duration is domain specific.
- Power Info (Optional) - Interface providing information on the range of parameters for a given domain, minimum power, maximum power etc.
- Policy (Optional) - 4-bit priority information that is a hint to hardware for dividing budget between sub-domains in a parent domain.

Each of the above capabilities requires specific units in order to describe them. Power is expressed in Watts, Time is expressed in Seconds, and Energy is expressed in Joules. Scaling factors are supplied to each unit to make the information presented meaningful in a finite number of bits. Units for power, energy, and time are exposed in the read-only MSR_RAPL_POWER_UNIT MSR.

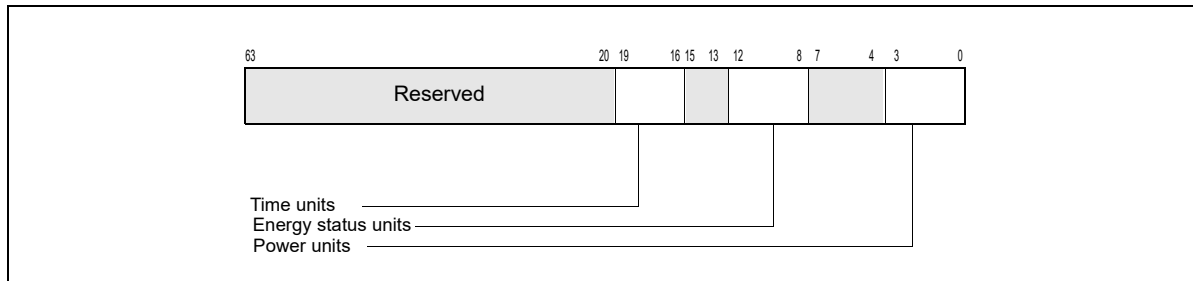


Figure 16-35. MSR_RAPL_POWER_UNIT Register

MSR_RAPL_POWER_UNIT (Figure 16-35) provides the following information across all RAPL domains:

- **Power Units** (bits 3:0): Power related information (in Watts) is based on the multiplier, $1/2^{\text{PU}}$; where PU is an unsigned integer represented by bits 3:0. Default value is 0011b, indicating power unit is in 1/8 Watts increment.
- **Energy Status Units** (bits 12:8): Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 10000b, indicating energy status unit is in 15.3 micro-Joules increment.
- **Time Units** (bits 19:16): Time related information (in Seconds) is based on the multiplier, $1/2^{\text{TU}}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating time unit is in 976 micro-seconds increment.

16.10.2 RAPL Domains and Platform Specificity

The specific RAPL domains available in a platform vary across product segments. Platforms targeting the client segment support the following RAPL domain hierarchy:

- Package
- Two power planes: PP0 and PP1 (PP1 may reflect to uncore devices)

Platforms targeting the server segment support the following RAPL domain hierarchy:

- Package
- Power plane: PP0
- DRAM

Each level of the RAPL hierarchy provides a respective set of RAPL interface MSRs. Table 16-12 lists the RAPL MSR interfaces available for each RAPL domain. The power limit MSR of each RAPL domain is located at offset 0 relative to an MSR base address which is non-architectural; see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4. The energy status MSR of each domain is located at offset 1 relative to the MSR base address of respective domain.

Table 16-12. RAPL MSR Interfaces and RAPL Domains

Domain	Power Limit (Offset 0)	Energy Status (Offset 1)	Policy (Offset 2)	Perf Status (Offset 3)	Power Info (Offset 4)
PKG	MSR_PKG_POWER_LIMIT	MSR_PKG_ENERGY_STATUS	RESERVED	MSR_PKG_PERF_STATUS	MSR_PKG_POWER_INFO
DRAM	MSR_DRAM_POWER_LIMIT	MSR_DRAM_ENERGY_STATUS	RESERVED	MSR_DRAM_PERF_STATUS	MSR_DRAM_POWER_INFO
PP0	MSR_PP0_POWER_LIMIT	MSR_PP0_ENERGY_STATUS	MSR_PP0_POLICY	MSR_PP0_PERF_STATUS	RESERVED
PP1	MSR_PP1_POWER_LIMIT	MSR_PP1_ENERGY_STATUS	MSR_PP1_POLICY	RESERVED	RESERVED

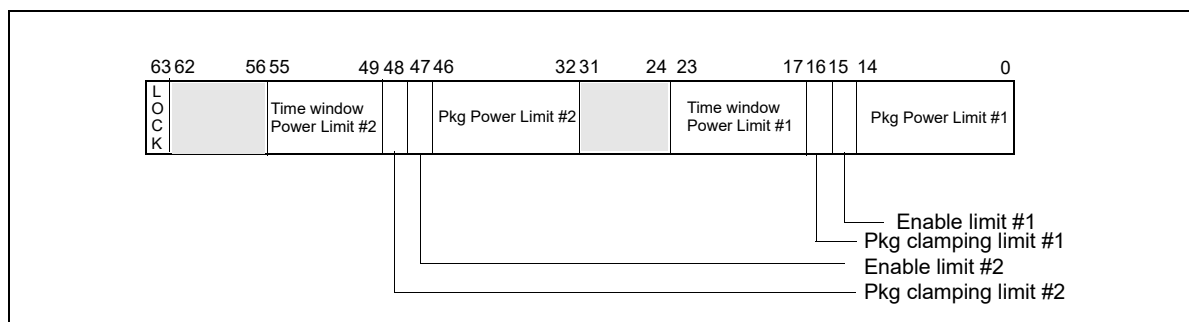
The presence of the optional MSR interfaces (the three right-most columns of Table 16-12) may be model-specific. See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for details.

16.10.3 Package RAPL Domain

The MSR interfaces defined for the package RAPL domain are:

- MSR_PKG_POWER_LIMIT allows software to set power limits for the package and measurement attributes associated with each limit,
- MSR_PKG_ENERGY_STATUS reports measured actual energy usage,
- MSR_PKG_POWER_INFO reports the package power range information for RAPL usage.

MSR_PKG_PERF_STATUS can report the performance impact of power limiting, but its availability may be model-specific.

**Figure 16-36. MSR_PKG_POWER_LIMIT Register**

MSR_PKG_POWER_LIMIT allows a software agent to define power limitation for the package domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_PKG_POWER_LIMIT. Two power limits can be specified, corresponding to time windows of different sizes. Each power limit provides independent clamping control that would permit the processor cores to go below OS-requested state to meet the power limits. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_PKG_POWER_LIMIT (Figure 16-36) are:

- **Package Power Limit #1** (bits 14:0): Sets the average power usage limit of the package domain corresponding to time window # 1. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.

- **Enable Power Limit #1** (bit 15): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #1** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit #1** (bits 23:17): Indicates the time window for power limit #1

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$$

Here "Y" is the unsigned integer value represented. by bits 21:17, "Z" is an unsigned integer represented by bits 23:22. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Package Power Limit #2** (bits 46:32): Sets the average power usage limit of the package domain corresponding to time window # 2. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #2** (bit 47): 0 = disabled; 1 = enabled.
- **Package Clamping Limitation #2** (bit 48): Allow going below OS-requested P/T state setting during time window specified by bits 55:49.
- **Time Window for Power Limit #2** (bits 55:49): Indicates the time window for power limit #2

$$\text{Time limit} = 2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$$

Here "Y" is the unsigned integer value represented. by bits 53:49, "Z" is an unsigned integer represented by bits 55:54. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT. This field may have a hard-coded value in hardware and ignores values written by software.
- **Lock** (bit 63): If set, all write attempts to this MSR are ignored until next RESET.

MSR_PKG_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the package domain. This MSR is updated every ~1msec. It has a wraparound time of around 60 secs when power consumption is high, and may be longer otherwise.

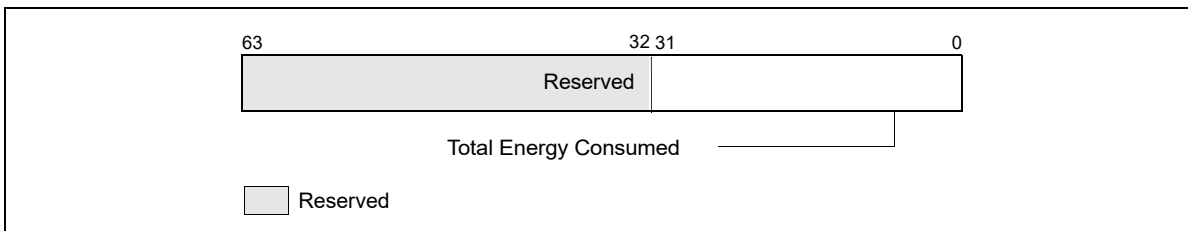


Figure 16-37. MSR_PKG_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PKG_POWER_INFO is a read-only MSR. It reports the package power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the package domain. It also provides the largest possible time window for software to program the RAPL interface.

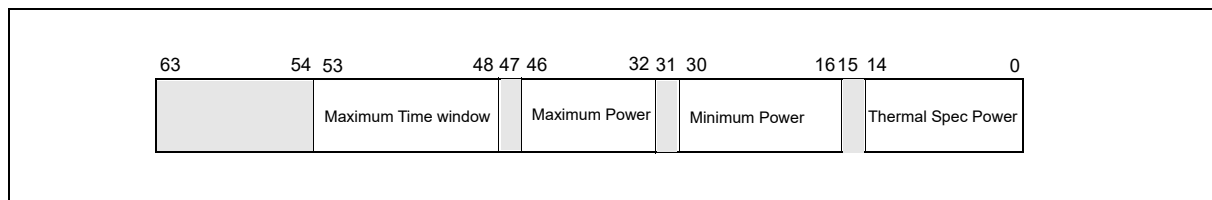


Figure 16-38. MSR_PKG_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.

- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the package domain. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the package domain. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_PKG_POWER_LIMIT. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.

MSR_PKG_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific; see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

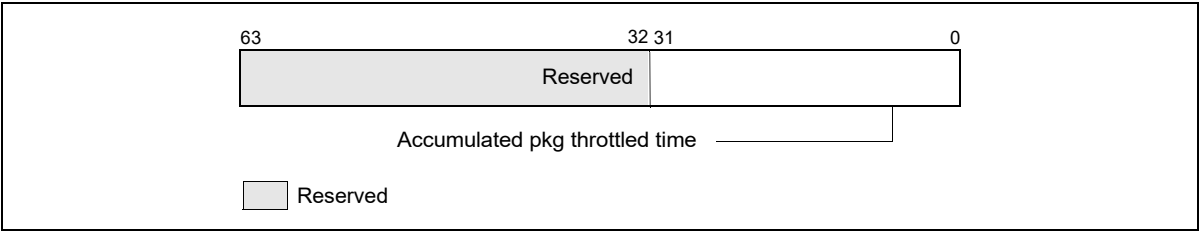


Figure 16-39. MSR_PKG_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the package has throttled. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.

16.10.4 PP0/PP1 RAPL Domains

The MSR interfaces defined for the PP0 and PP1 domains are identical in layout. Generally, PP0 refers to the processor cores. The availability of PP1 RAPL domain interface is platform-specific. For a client platform, the PP1 domain refers to the power plane of a specific device in the uncore. For server platforms, the PP1 domain is not supported, but its PP0 domain supports the MSR_PP0_PERF_STATUS interface.

- MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow software to set power limits for the respective power plane domain.
- MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS report actual energy usage on a power plane.
- MSR_PP0_POLICY/MSR_PP1_POLICY allow software to adjust balance for respective power plane.

MSR_PP0_PERF_STATUS can report the performance impact of power limiting, but it is not available in client platforms.

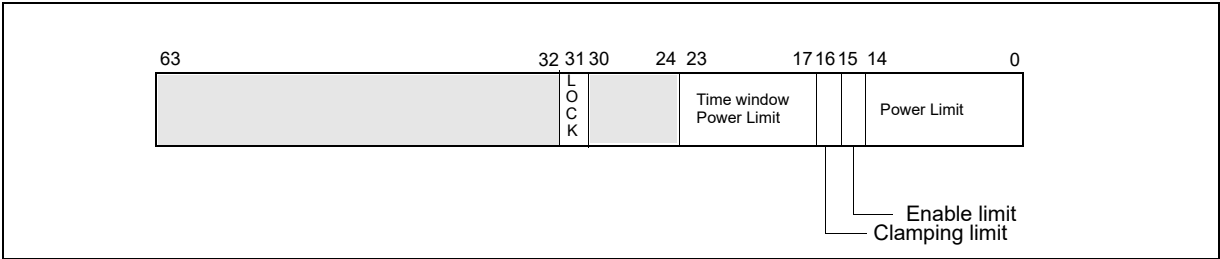


Figure 16-40. MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT Register

MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT allow a software agent to define power limitation for the respective power plane domain. A lock mechanism in each power plane domain allows the software agent to enforce power limit settings independently. Once a lock bit is set, the power limit settings in that power plane are static and un-modifiable until next RESET.

The bit fields of MSR_PP0_POWER_LIMIT/MSR_PP1_POWER_LIMIT (Figure 16-40) are:

- **Power Limit** (bits 14:0): Sets the average power usage limit of the respective power plane domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit** (bit 15): 0 = disabled; 1 = enabled.
- **Clamping Limitation** (bit 16): Allow going below OS-requested P/T state setting during time window specified by bits 23:17.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit #1 will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to the MSR and corresponding policy MSR_PP0_POLICY/MSR_PP1_POLICY are ignored until next RESET.

MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS are read-only MSRs. They report the actual energy use for the respective power plane domains. These MSRs are updated every ~1msec.

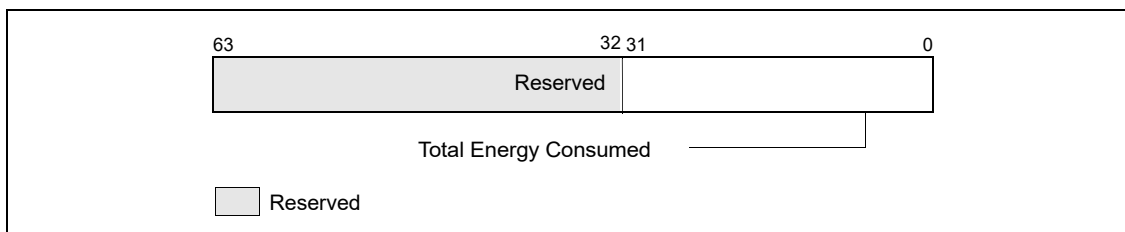


Figure 16-41. MSR_PP0_ENERGY_STATUS/MSR_PP1_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since the last time this register was cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_PP0_POLICY/MSR_PP1_POLICY provide balance power policy control for each power plane by providing inputs to the power budgeting management algorithm. On platforms that support PP0 (IA cores) and PP1 (uncore graphic device), the default values give priority to the non-IA power plane. These MSRs enable the PCU to balance power consumption between the IA cores and uncore graphic device.

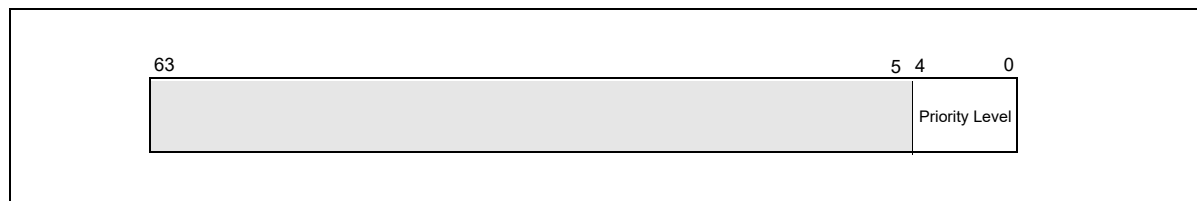


Figure 16-42. MSR_PP0_POLICY/MSR_PP1_POLICY Register

- **Priority Level** (bits 4:0): Priority level input to the PCU for respective power plane. PP0 covers the IA processor cores, PP1 covers the uncore graphic device. The value 31 is considered highest priority.

MSR_PP0_PERF_STATUS is a read-only MSR. It reports the total time for which the PP0 domain was throttled due to the power limits. This MSR is supported only in server platform. Throttling in this context is defined as going below the OS-requested P-state or T-state.

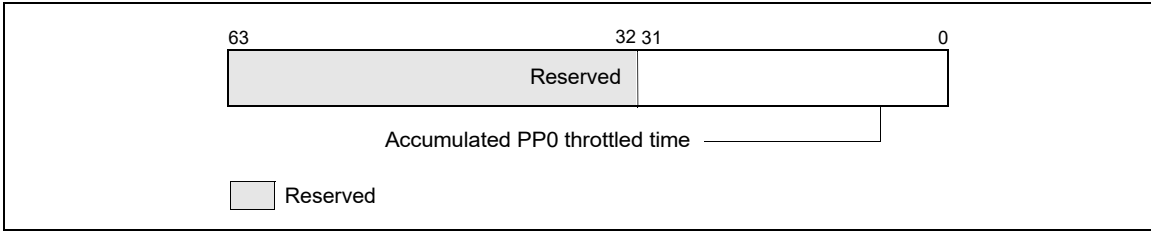


Figure 16-43. MSR_PPO_PERF_STATUS MSR

- **Accumulated PPO Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the PPO domain has throttled. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.

16.10.5 DRAM RAPL Domain

The MSR interfaces defined for the DRAM domains are supported only in the server platform. The MSR interfaces are:

- MSR_DRAM_POWER_LIMIT allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.
- MSR_DRAM_ENERGY_STATUS reports measured actual energy usage.
- MSR_DRAM_POWER_INFO reports the DRAM domain power range information for RAPL usage.
- MSR_DRAM_PERF_STATUS can report the performance impact of power limiting.

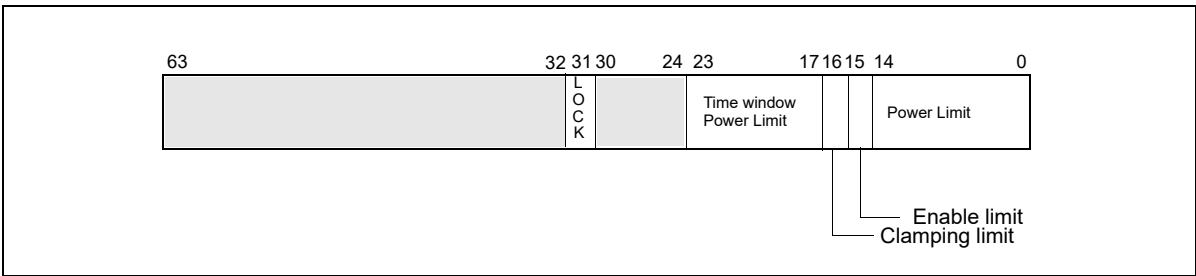


Figure 16-44. MSR_DRAM_POWER_LIMIT Register

MSR_DRAM_POWER_LIMIT allows a software agent to define power limitation for the DRAM domain. Power limitation is defined in terms of average power usage (Watts) over a time window specified in MSR_DRAM_POWER_LIMIT. A power limit can be specified along with a time window. A lock mechanism allow the software agent to enforce power limit settings. Once the lock bit is set, the power limit settings are static and un-modifiable until next RESET.

The bit fields of MSR_DRAM_POWER_LIMIT (Figure 16-44) are:

- **DRAM Power Limit #1** (bits 14:0): Sets the average power usage limit of the DRAM domain corresponding to time window # 1. The unit of this field is specified by the “Power Units” field of MSR_RAPL_POWER_UNIT.
- **Enable Power Limit #1** (bit 15): 0 = disabled; 1 = enabled.
- **Time Window for Power Limit** (bits 23:17): Indicates the length of time window over which the power limit will be used by the processor. The numeric value encoded by bits 23:17 is represented by the product of $2^Y * F$; where F is a single-digit decimal floating-point value between 1.0 and 1.3 with the fraction digit represented by bits 23:22, Y is an unsigned integer represented by bits 21:17. The unit of this field is specified by the “Time Units” field of MSR_RAPL_POWER_UNIT.
- **Lock** (bit 31): If set, all write attempts to this MSR are ignored until next RESET.

MSR_DRAM_ENERGY_STATUS is a read-only MSR. It reports the actual energy use for the DRAM domain. This MSR is updated every ~1msec.

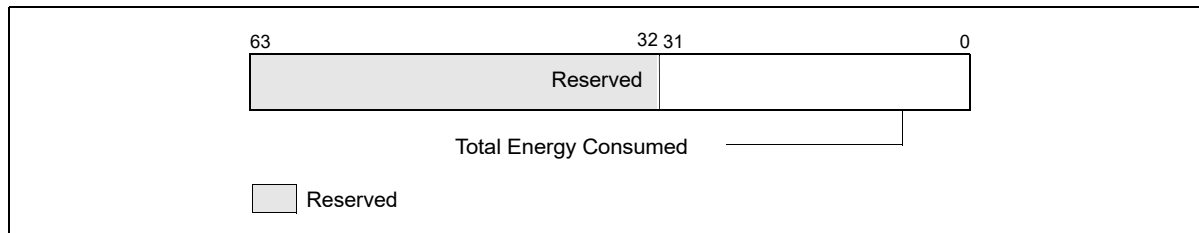


Figure 16-45. MSR_DRAM_ENERGY_STATUS MSR

- **Total Energy Consumed** (bits 31:0): The unsigned integer value represents the total amount of energy consumed since that last time this register is cleared. The unit of this field is specified by the "Energy Status Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_POWER_INFO is a read-only MSR. It reports the DRAM power range information for RAPL usage. This MSR provides maximum/minimum values (derived from electrical specification), thermal specification power of the DRAM domain. It also provides the largest possible time window for software to program the RAPL interface.

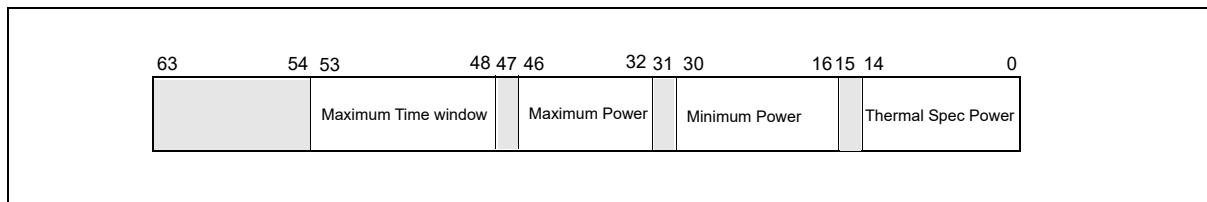


Figure 16-46. MSR_DRAM_POWER_INFO Register

- **Thermal Spec Power** (bits 14:0): The unsigned integer value is the equivalent of thermal specification power of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Minimum Power** (bits 30:16): The unsigned integer value is the equivalent of minimum power derived from electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Power** (bits 46:32): The unsigned integer value is the equivalent of maximum power derived from the electrical spec of the DRAM domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.
- **Maximum Time Window** (bits 53:48): The unsigned integer value is the equivalent of largest acceptable value to program the time window of MSR_DRAM_POWER_LIMIT. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

MSR_DRAM_PERF_STATUS is a read-only MSR. It reports the total time for which the package was throttled due to the RAPL power limits. Throttling in this context is defined as going below the OS-requested P-state or T-state. It has a wrap-around time of many hours. The availability of this MSR is platform specific; see Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

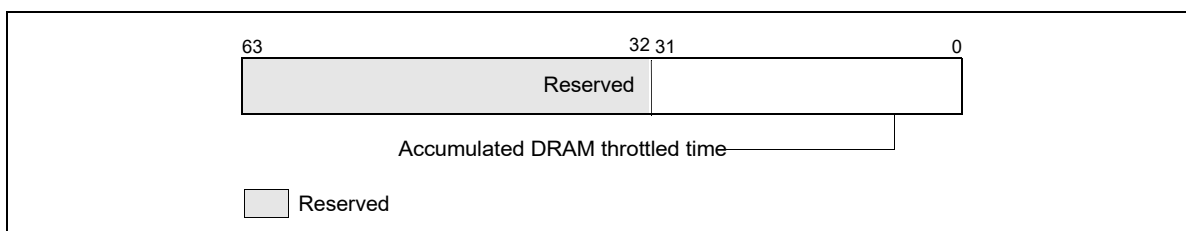


Figure 16-47. MSR_DRAM_PERF_STATUS MSR

- **Accumulated Package Throttled Time** (bits 31:0): The unsigned integer value represents the cumulative time (since the last time this register is cleared) that the DRAM domain has throttled. The unit of this field is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.

16. Updates to Chapter 17, Volume 3B

Change bars and violet text show changes to Chapter 17 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- In Section 17.3.1.1, "IA32_MCG_CAP MSR", updated Figure 17-2 and added MCG_SEAM_NR_P description.
- In Section 17.3.1.2, "IA32_MCG_STATUS MSR", updated Figure 17-3 and added SEAM_NR description.
- In Section 17.11.1, "Machine-Check Exception Handler", added a note to the RIPV flag in the IA32_MCG_STATUS register and the SEAM_NR flag.

This chapter describes the machine-check architecture and machine-check exception mechanism found in the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors. See Chapter 7, “Interrupt 18—Machine-Check Exception (#MC),” for more information on machine-check exceptions. A brief description of the Pentium processor’s machine check capability is also given.

Additionally, a signaling mechanism for software to respond to hardware corrected machine check error is covered.

17.1 MACHINE-CHECK ARCHITECTURE

The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors implement a machine-check architecture that provides a mechanism for detecting and reporting hardware (machine) errors, such as: system bus errors, ECC errors, parity errors, cache errors, and TLB errors. It consists of a set of model-specific registers (MSRs) that are used to set up machine checking and additional banks of MSRs used for recording errors that are detected.

The processor signals the detection of an uncorrected machine-check error by generating a machine-check exception (#MC), which is an abort class exception. The implementation of the machine-check architecture does not ordinarily permit the processor to be restarted reliably after generating a machine-check exception. However, the machine-check-exception handler can collect information about the machine-check error from the machine-check MSRs.

Starting with 45 nm Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH; see the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A. The processor can report information on corrected machine-check errors and deliver a programmable interrupt for software to respond to MC errors, referred to as corrected machine-check error interrupt (CMCI). See Section 17.5 for details.

Intel 64 processors supporting machine-check architecture and CMCI may also support an additional enhancement, namely, support for software recovery from certain uncorrected recoverable machine check errors. See Section 17.6 for details.

17.2 COMPATIBILITY WITH PENTIUM PROCESSOR

The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors support and extend the machine-check exception mechanism introduced in the Pentium processor. The Pentium processor reports the following machine-check errors:

- Data parity errors during read cycles.
- Unsuccessful completion of a bus cycle.

The above errors are reported using the P5_MC_TYPE and P5_MC_ADDR MSRs (implementation specific for the Pentium processor). Use the RDMSR instruction to read these MSRs. See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for the addresses.

The machine-check error reporting mechanism that Pentium processors use is similar to that used in Pentium 4, Intel Xeon, Intel Atom, and P6 family processors. When an error is detected, it is recorded in P5_MC_TYPE and P5_MC_ADDR; the processor then generates a machine-check exception (#MC).

See Section 17.3.3, “Mapping of the Pentium Processor Machine-Check Errors to the Machine-Check Architecture,” and Section 17.11.2, “Pentium Processor Machine-Check Exception Handling,” for information on compatibility between machine-check code written to run on the Pentium processors and code written to run on P6 family processors.

17.3 MACHINE-CHECK MSRS

Machine check MSRs in the Pentium 4, Intel Atom, Intel Xeon, and P6 family processors consist of a set of global control and status registers and several error-reporting register banks. See Figure 17-1.

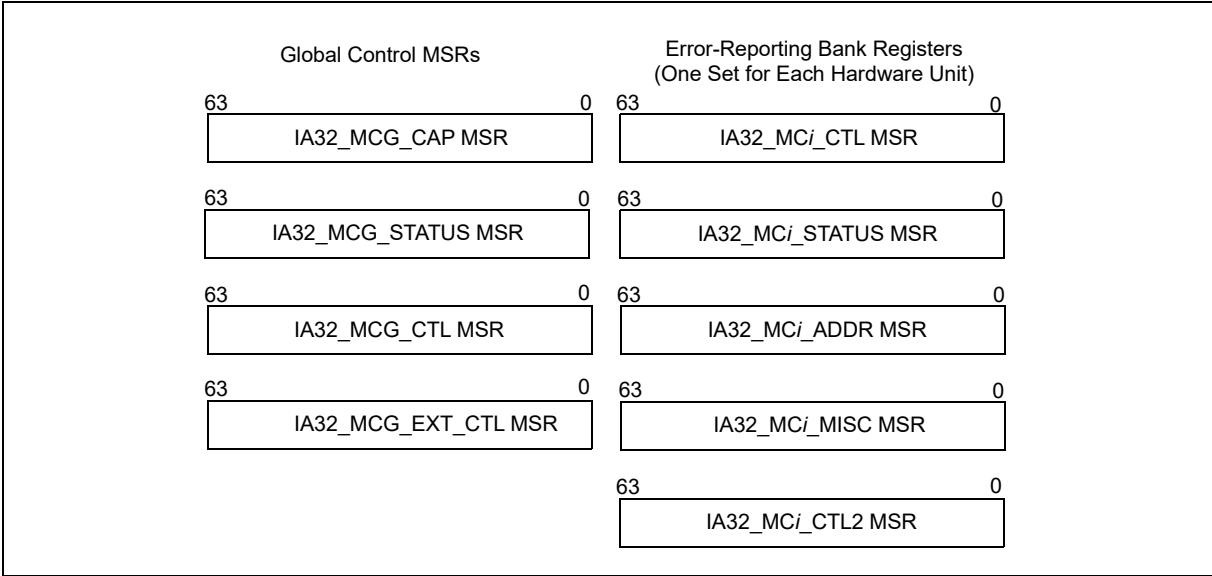


Figure 17-1. Machine-Check MSRs

Each error-reporting bank is associated with a specific hardware unit (or group of hardware units) in the processor. Use RDMSR and WRMSR to read and to write these registers.

17.3.1 Machine-Check Global Control MSRs

The machine-check global control MSRs include the IA32_MCG_CAP, IA32_MCG_STATUS, and optionally IA32_MCG_CTL and IA32_MCG_EXT_CTL. See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for the addresses of these registers.

17.3.1.1 IA32_MCG_CAP MSR

The IA32_MCG_CAP MSR is a read-only register that provides information about the machine-check architecture of the processor. Figure 17-2 shows the layout of the register.

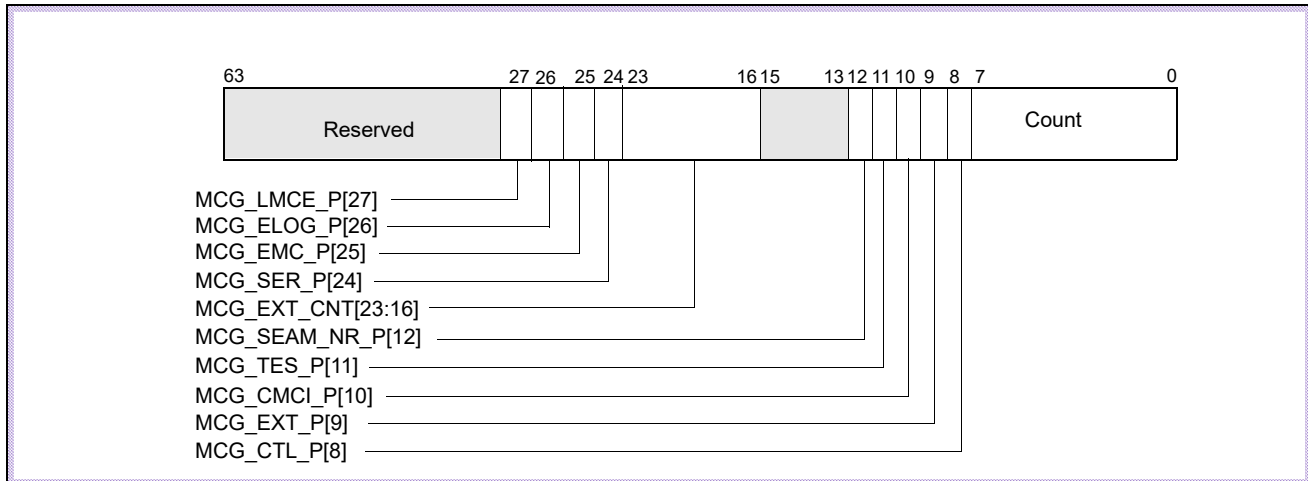


Figure 17-2. IA32_MCG_CAP Register

Where:

- **Count field, bits 7:0** — Indicates the number of hardware unit error-reporting banks available in a particular processor implementation.
- **MCG_CTL_P (control MSR present) flag, bit 8** — Indicates that the processor implements the IA32_MCG_CTL MSR when set; this register is absent when clear.
- **MCG_EXT_P (extended MSRs present) flag, bit 9** — Indicates that the processor implements the extended machine-check state registers found starting at MSR address 180H; these registers are absent when clear.
- **MCG_CMCI_P (Corrected MC error counting/signaling extension present) flag, bit 10** — Indicates (when set) that extended state and associated MSRs necessary to support the reporting of an interrupt on a corrected MC error event and/or count threshold of corrected MC errors, is present. When this bit is set, it does not imply this feature is supported across all banks. Software should check the availability of the necessary logic on a bank by bank basis when using this signaling capability (i.e., bit 30 settable in individual IA32_MCi_CTL2 register).
- **MCG_TESP_P (threshold-based error status present) flag, bit 11** — Indicates (when set) that bits 56:53 of the IA32_MCi_STATUS MSR are part of the architectural space. Bits 56:55 are reserved, and bits 54:53 are used to report threshold-based error status. Note that when MCG_TESP_P is not set, bits 56:53 of the IA32_MCi_STATUS MSR are model-specific.
- **MCG_SEAM_NRP (SEAM non-root operation indication present), bit 12** — Indicates (when set) that the processor supports the SEAM_NRP bit in position 12 of the IA32_MCG_STATUS MSR.
- **MCG_EXT_CNT, bits 23:16** — Indicates the number of extended machine-check state registers present. This field is meaningful only when the MCG_EXT_P flag is set.
- **MCG_SER_P (software error recovery support present) flag, bit 24** — Indicates (when set) that the processor supports software error recovery (see Section 17.6), and IA32_MCi_STATUS MSR bits 56:55 are used to report the signaling of uncorrected recoverable errors and whether software must take recovery actions for uncorrected errors. Note that when MCG_TESP_P is not set, bits 56:53 of the IA32_MCi_STATUS MSR are model-specific. If MCG_TESP_P is set but MCG_SER_P is not set, bits 56:55 are reserved.
- **MCG_EMCP_P (Enhanced Machine Check Capability) flag, bit 25** — Indicates (when set) that the processor supports enhanced machine check capabilities for firmware first signaling.
- **MCG_ELOG_P (extended error logging) flag, bit 26** — Indicates (when set) that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.

For additional information about extended error logging interface, see <https://cdrdv2.intel.com/v1/dl/getContent/671064>.

- **MCG_LMCE_P (local machine check exception) flag, bit 27** — Indicates (when set) that the following interfaces are present:
 - an extended state LMCE_S (located in bit 3 of IA32_MCG_STATUS), and
 - the IA32_MCG_EXT_CTL MSR, necessary to support Local Machine Check Exception (LMCE).

A non-zero MCG_LMCE_P indicates that, when LMCE is enabled as described in Section 17.3.1.5, some machine check errors may be delivered to only a single logical processor.

The effect of writing to the IA32_MCG_CAP MSR is undefined.

17.3.1.2 IA32_MCG_STATUS MSR

The IA32_MCG_STATUS MSR describes the current state of the processor after a machine-check exception has occurred (see Figure 17-3).

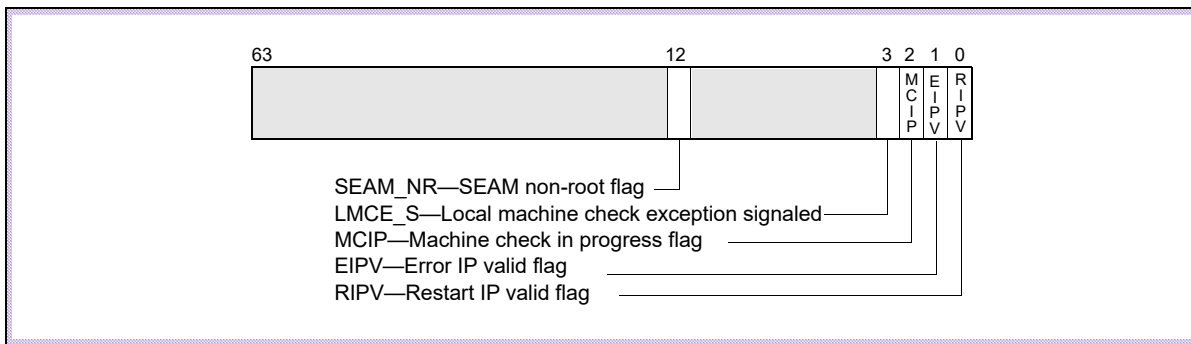


Figure 17-3. IA32_MCG_STATUS Register

Where:

- **RIPV (restart IP valid) flag, bit 0** — Indicates (when set) that program execution can be restarted reliably at the instruction pointed to by the instruction pointer pushed on the stack when the machine-check exception is generated. When clear, the program cannot be reliably restarted at the pushed instruction pointer.
- **EIPV (error IP valid) flag, bit 1** — Indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- **MCIP (machine check in progress) flag, bit 2** — Indicates (when set) that a machine-check exception was generated. Software can set or clear this flag. The occurrence of a second Machine-Check Event while MCIP is set will cause the processor to enter a shutdown state. For information on processor behavior in the shutdown state, please refer to the description in Chapter 7, “Interrupt and Exception Handling”: “Interrupt 8—Double Fault Exception (#DF)”.
- **LMCE_S (local machine check exception signaled), bit 3** — Indicates (when set) that a local machine-check exception was generated. This indicates that the current machine-check event was delivered to only this logical processor.
- **SEAM_NR (SEAM non-root operation), bit 12** — Indicates (when set) that the machine-check exception was observed while the logical processor was running in SEAM non-root operation.

Bits 63:13 and bits 11:4 in the IA32_MCG_STATUS MSR are reserved. Bit 12 is reserved if IA32_MCG_CAP.MCG_SEAM_NR_P = 0. An attempt to write to the IA32_MCG_STATUS MSR’s reserved bits with any value other than 0 results in #GP.

17.3.1.3 IA32_MCG_CTL MSR

The IA32_MCG_CTL MSR is present if the capability flag MCG_CTL_P is set in the IA32_MCG_CAP MSR.

IA32_MCG_CTL controls the reporting of machine-check exceptions. If present, writing 1s to this register enables machine-check features and writing all 0s disables machine-check features. All other values are undefined and/or implementation specific.

17.3.1.4 IA32_MCG_EXT_CTL MSR

The IA32_MCG_EXT_CTL MSR is present if the capability flag MCG_LMCE_P is set in the IA32_MCG_CAP MSR.

IA32_MCG_EXT_CTL.LMCE_EN (bit 0) allows the processor to signal some MCEs to only a single logical processor in the system.

If MCG_LMCE_P is not set in IA32_MCG_CAP, or platform software has not enabled LMCE by setting IA32_FEATURE_CONTROL.LMCE_ENABLED (bit 20), any attempt to write or read IA32_MCG_EXT_CTL will result in #GP.

The IA32_MCG_EXT_CTL MSR is cleared on RESET.

Figure 17-4 shows the layout of the IA32_MCG_EXT_CTL register

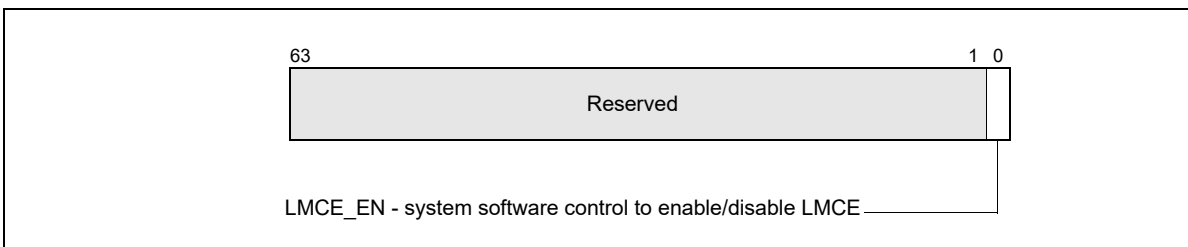


Figure 17-4. IA32_MCG_EXT_CTL Register

where

- **LMCE_EN (local machine check exception enable) flag, bit 0** - System software sets this to allow hardware to signal some MCEs to only a single logical processor. System software can set LMCE_EN only if the platform software has configured IA32_FEATURE_CONTROL as described in Section 17.3.1.5.

17.3.1.5 Enabling Local Machine Check

The intended usage of LMCE requires proper configuration by both platform software and system software. Platform software can turn LMCE on by setting bit 20 (LMCE_ENABLED) in IA32_FEATURE_CONTROL MSR (MSR address 3AH).

System software must ensure that both IA32_FEATURE_CONTROL.Lock (bit 0) and IA32_FEATURE_CONTROL.LMCE_ENABLED (bit 20) are set before attempting to set IA32_MCG_EXT_CTL.LMCE_EN (bit 0). When system software has enabled LMCE, then hardware will determine if a particular error can be delivered only to a single logical processor. Software should make no assumptions about the type of error that hardware can choose to deliver as LMCE. The severity and override rules stay the same as described in Table 17-8 to determine the recovery actions.

17.3.2 Error-Reporting Register Banks

Each error-reporting register bank can contain the IA32_MCi_CTL, IA32_MCi_STATUS, IA32_MCi_ADDR, and IA32_MCi_MISC MSRs. The number of reporting banks is indicated by bits [7:0] of IA32_MCG_CAP MSR (address 0179H). The first error-reporting register (IA32_MC0_CTL) always starts at address 400H.

See Chapter 2, "Model-Specific Registers (MSRs)," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4, for addresses of the error-reporting registers in the Pentium 4, Intel Atom, and Intel Xeon processors; and for addresses of the error-reporting registers P6 family processors.

17.3.2.1 IA32_MCi_CTL MSRs

The IA32_MCi_CTL MSR controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units). Each of the 64 flags (EEj) represents a potential error. Setting an EEj flag enables signaling #MC of the associated error and clearing it disables signaling of the error. Error logging happens regardless of the setting of these bits. The processor drops writes to bits that are not implemented. Figure 17-5 shows the bit fields of IA32_MCi_CTL.

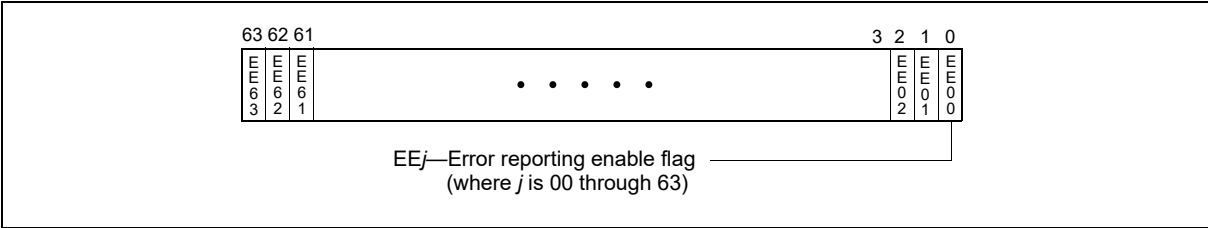


Figure 17-5. IA32_MCi_CTL Register

NOTE

For P6 family processors, processors based on Intel Core microarchitecture (excluding those on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH and onward): the operating system or executive software must not modify the contents of the IA32_MC0_CTL MSR. This MSR is internally aliased to the EBL_CR_POWERON MSR and controls platform-specific error handling features. System specific firmware (the BIOS) is responsible for the appropriate initialization of the IA32_MC0_CTL MSR. P6 family processors only allow the writing of all 1s or all 0s to the IA32_MCi_CTL MSR.

17.3.2.2 IA32_MCi_STATUS MSRS

Each IA32_MCi_STATUS MSR contains information related to a machine-check error if its VAL (valid) flag is set (see Figure 17-6). Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.

NOTE

Figure 17-6 depicts the IA32_MCi_STATUS MSR when IA32_MCG_CAP[24] = 1, IA32_MCG_CAP[11] = 1 and IA32_MCG_CAP[10] = 1. When IA32_MCG_CAP[24] = 0 and IA32_MCG_CAP[11] = 1, bits 56:55 is reserved and bits 54:53 for threshold-based error reporting. When IA32_MCG_CAP[11] = 0, bits 56:53 are part of the “Other Information” field. The use of bits 54:53 for threshold-based error reporting began with Intel Core Duo processors, and is currently used for cache memory. See Section 17.4, “Enhanced Cache Error reporting,” for more information. When IA32_MCG_CAP[10] = 0, bits 52:38 are part of the “Other Information” field. The use of bits 52:38 for corrected MC error count is introduced with Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_1AH.

Where:

- **MCA (machine-check architecture) error code field, bits 15:0** — Specifies the machine-check architecture-defined error code for the machine-check error condition detected. The machine-check architecture-defined error codes are guaranteed to be the same for all IA-32 processors that implement the machine-check architecture. See Section 17.10, “Interpreting the MCA Error Codes,” and Chapter 18, “Interpreting Machine Check Error Codes,” for information on machine-check error codes.
- **Model-specific error code field, bits 31:16** — Specifies the model-specific error code that uniquely identifies the machine-check error condition detected. The model-specific error codes may differ among IA-32 processors for the same machine-check error condition. See Chapter 18, “Interpreting Machine Check Error Codes,” for information on model-specific error codes.
- **Reserved, Error Status, and Other Information fields, bits 56:32** —

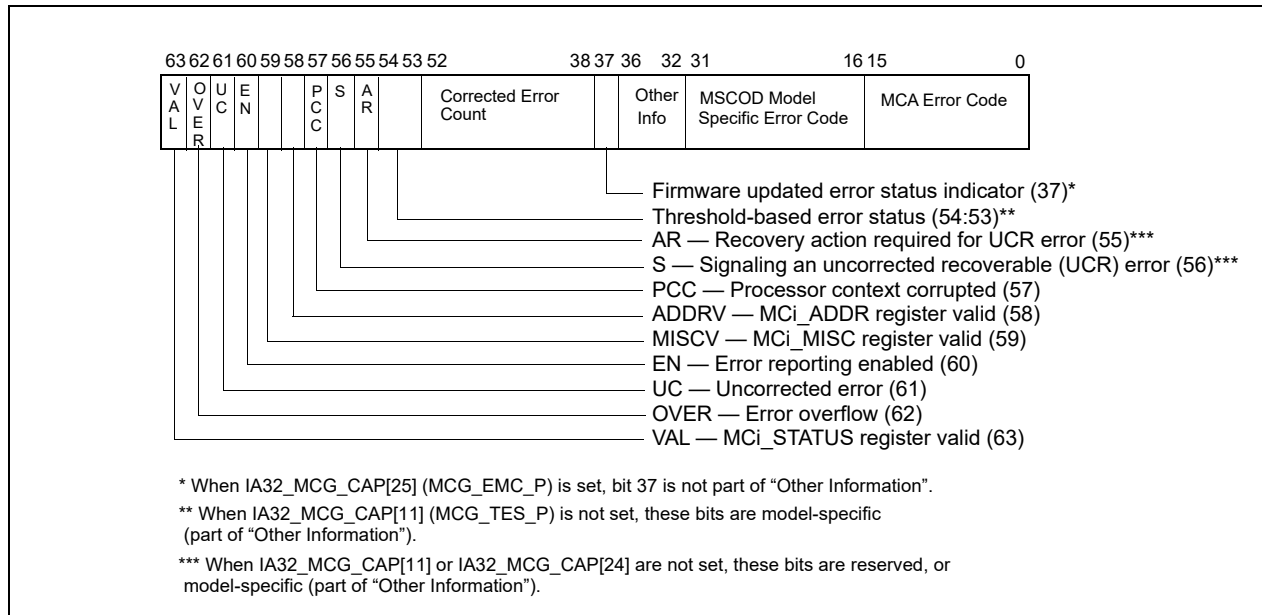


Figure 17-6. IA32_MCi_STATUS Register

- If IA32_MCG_CAP.MCG_EMC_P[bit 25] is 0, bits 37:32 contain "Other Information" that is implementation-specific and is not part of the machine-check architecture.
- If IA32_MCG_CAP.MCG_EMC_P is 1, "Other Information" is in bits 36:32. If bit 37 is 0, system firmware has not changed the contents of IA32_MCi_STATUS. If bit 37 is 1, system firmware may have edited the contents of IA32_MCi_STATUS.
- If IA32_MCG_CAP.MCG_CMCI_P[bit 10] is 0, bits 52:38 also contain "Other Information" (in the same sense as bits 37:32).
- If IA32_MCG_CAP[10] is 1, bits 52:38 are architectural (not model-specific). In this case, bits 52:38 reports the value of a 15 bit counter that increments each time a corrected error is observed by the MCA recording bank. This count value will continue to increment until cleared by software. The most significant bit, 52, is a sticky count overflow bit.
- If IA32_MCG_CAP[11] is 0, bits 56:53 also contain "Other Information" (in the same sense).
- If IA32_MCG_CAP[11] is 1, bits 56:53 are architectural (not model-specific). In this case, bits 56:53 have the following functionality:
 - If IA32_MCG_CAP[24] is 0, bits 56:55 are reserved.
 - If IA32_MCG_CAP[24] is 1, bits 56:55 are defined as follows:
 - S (Signaling) flag, bit 56 - Signals the reporting of UCR errors in this MC bank. See Section 17.6.2 for additional details.
 - AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. See Section 17.6.2 for additional details.
 - If the UC bit (Figure 17-6) is 1, bits 54:53 are undefined.
 - If the UC bit (Figure 17-6) is 0, bits 54:53 indicate the status of the hardware structure that reported the threshold-based error. See Table 17-1.

Table 17-1. Bits 54:53 in IA32_MCi_STATUS MSRs when IA32_MCG_CAP[11] = 1 and UC = 0

Bits 54:53	Meaning
00	No tracking - No hardware status tracking is provided for the structure reporting this event.
01	Green - Status tracking is provided for the structure posting the event; the current status is green (below threshold). For more information, see Section 17.4, “Enhanced Cache Error reporting.”
10	Yellow - Status tracking is provided for the structure posting the event; the current status is yellow (above threshold). For more information, see Section 17.4, “Enhanced Cache Error reporting.”
11	Reserved

- **PCC (processor context corrupt) flag, bit 57** — Indicates (when set) that the state of the processor might have been corrupted by the error condition detected and that reliable restarting of the processor may not be possible. When clear, this flag indicates that the error did not affect the processor’s state, and software may be able to restart. When system software supports recovery, consult Section 17.11.4, “Machine-Check Software Handler Guidelines for Error Recovery,” for additional rules that apply.
- **ADDRV (IA32_MCi_ADDR register valid) flag, bit 58** — Indicates (when set) that the IA32_MCi_ADDR register contains the address where the error occurred (see Section 17.3.2.3, “IA32_MCi_ADDR MSRs”). When clear, this flag indicates that the IA32_MCi_ADDR register is either not implemented or does not contain the address where the error occurred. Do not read these registers if they are not implemented in the processor.
- **MISCV (IA32_MCi_MISC register valid) flag, bit 59** — Indicates (when set) that the IA32_MCi_MISC register contains additional information regarding the error. When clear, this flag indicates that the IA32_MCi_MISC register is either not implemented or does not contain additional information regarding the error. Do not read these registers if they are not implemented in the processor.
- **EN (error enabled) flag, bit 60** — Indicates (when set) that the error was enabled by the associated EEj bit of the IA32_MCi_CTL register.
- **UC (error uncorrected) flag, bit 61** — Indicates (when set) that the processor did not or was not able to correct the error condition. When clear, this flag indicates that the processor was able to correct the error condition.
- **OVER (machine check overflow) flag, bit 62** — Indicates (when set) that a machine-check error occurred while the results of a previous error were still in the error-reporting register bank (that is, the VAL bit was already set in the IA32_MCi_STATUS register). The processor sets the OVER flag and software is responsible for clearing it. In general, enabled errors are written over disabled errors, and uncorrected errors are written over corrected errors. Uncorrected errors are not written over previous valid uncorrected errors. When MCG_CMCI_P is set, corrected errors may not set the OVER flag. Software can rely on corrected error count in IA32_MCi_Status[52:38] to determine if any additional corrected errors may have occurred. For more information, see Section 17.3.2.2.1, “Overwrite Rules for Machine Check Overflow.”
- **VAL (IA32_MCi_STATUS register valid) flag, bit 63** — Indicates (when set) that the information within the IA32_MCi_STATUS register is valid. When this flag is set, the processor follows the rules given for the OVER flag in the IA32_MCi_STATUS register when overwriting previously valid entries. The processor sets the VAL flag and software is responsible for clearing it.

17.3.2.2.1 Overwrite Rules for Machine Check Overflow

Table 17-2 shows the overwrite rules for how to treat a second event if the MC bank already contains a valid log from an earlier event – that is, what to do if the valid bit for an MC bank already is set to 1. When more than one structure posts events in a given bank, these rules specify whether a new event will overwrite a previous posting or not. These rules define a priority for uncorrected (highest priority), yellow, and green/unmonitored (lowest priority) status.

In Table 17-2, the values in the two left-most columns are IA32_MCi_STATUS[54:53].

Table 17-2. Overwrite Rules for Enabled Errors

First Event	Second Event	UC bit	Color	MCA Info
00/green	00/green	0	00/green	either
00/green	yellow	0	yellow	second error

Table 17-2. Overwrite Rules for Enabled Errors

First Event	Second Event	UC bit	Color	MCA Info
yellow	00/green	0	yellow	first error
yellow	yellow	0	yellow	either
00/green/yellow	UC	1	undefined	second
UC	00/green/yellow	1	undefined	first

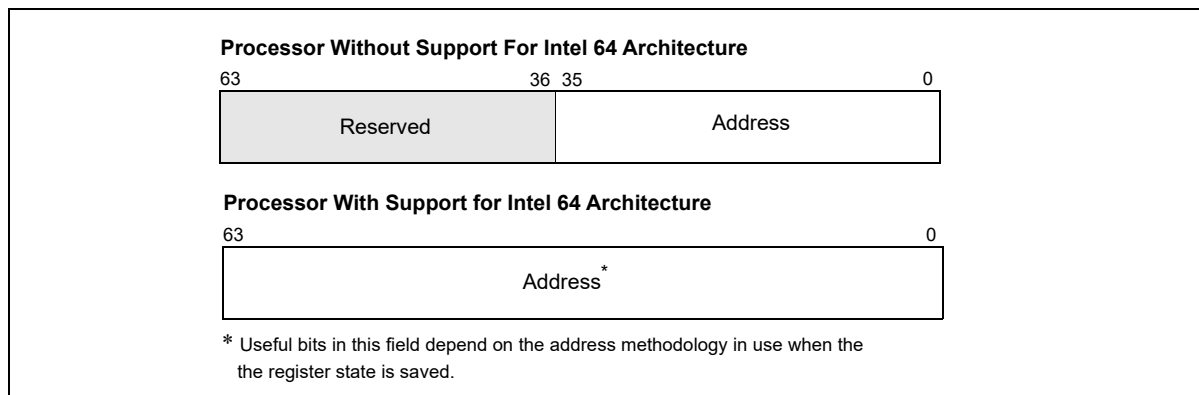
If a second event overwrites a previously posted event, the information (as guarded by individual valid bits) in the MCi bank is entirely from the second event. Similarly, if a first event is retained, all of the information previously posted for that event is retained. In general, when the logged error or the recent error is a corrected error, the OVER bit (MCi_Status[62]) may be set to indicate an overflow. When MCG_CMCI_P is set in IA32_MCG_CAP, system software should consult IA32_MCi_STATUS[52:38] to determine if additional corrected errors may have occurred. Software may re-read IA32_MCi_STATUS, IA32_MCi_ADDR, and IA32_MCi_MISC appropriately to ensure data collected represent the last error logged.

After software polls a posting and clears the register, the valid bit is no longer set and therefore the meaning of the rest of the bits, including the yellow/green/00 status field in bits 54:53, is undefined. The yellow/green indication will only be posted for events associated with monitored structures – otherwise the unmonitored (00) code will be posted in IA32_MCi_STATUS[54:53].

17.3.2.3 IA32_MCi_ADDR MSRs

The IA32_MCi_ADDR MSR contains the address of the code or data memory location that produced the machine-check error if the ADDRv flag in the IA32_MCi_STATUS register is set (see Section 17-7, “IA32_MCi_ADDR MSR”). The IA32_MCi_ADDR register is either not implemented or contains no address if the ADDRv flag in the IA32_MCi_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception.

The address returned is an offset into a segment, linear address, or physical address. This depends on the error encountered. When these registers are implemented, these registers can be cleared by explicitly writing 0s to these registers. Writing 1s to these registers will cause a general-protection exception. See Figure 17-7.

**Figure 17-7. IA32_MCi_ADDR MSR**

17.3.2.4 IA32_MCi_MISC MSRs

The IA32_MCi_MISC MSR contains additional information describing the machine-check error if the MISCV flag in the IA32_MCi_STATUS register is set. The IA32_MCi_MISC_MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MCi_STATUS register is clear.

When not implemented in the processor, all reads and writes to this MSR will cause a general protection exception. When implemented in a processor, these registers can be cleared by explicitly writing all bits to 0; writing a 1 to any

bit causes a general-protection exception to be generated. This register is not implemented in any of the error-reporting register banks for the Intel P6 family processors.

If both MISCV and IA32_MCG_CAP[24] are set, the IA32_MCi_MISC_MSR is defined according to Figure 17-8 to support software recovery of uncorrected errors (see Section 17.6).

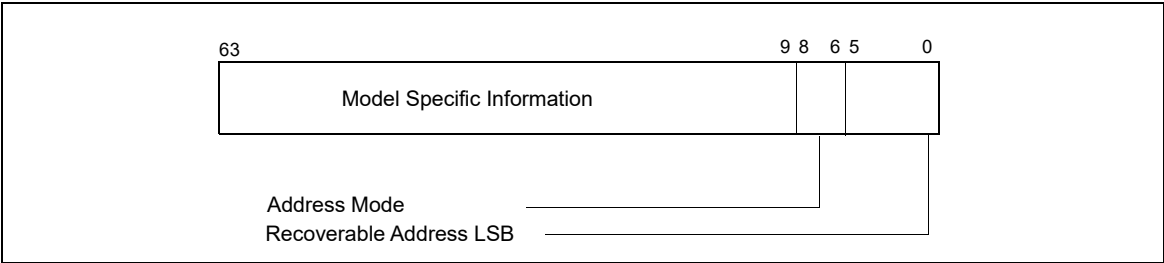


Figure 17-8. UCR Support in IA32_MCi_MISC Register

- Recoverable Address LSB (bits 5:0): The lowest valid recoverable address bit. Indicates the position of the least significant bit (LSB) of the recoverable error address. For example, if the processor logs bits [43:9] of the address, the LSB sub-field in IA32_MCi_MISC is 01001b (9 decimal). For this example, bits [8:0] of the recoverable error address in IA32_MCi_ADDR should be ignored.
- Address Mode (bits 8:6): Address mode for the address logged in IA32_MCi_ADDR. The supported address modes are given in Table 17-3.

Table 17-3. Address Mode in IA32_MCi_MISC[8:6]

IA32_MCi_MISC[8:6] Encoding	Definition
000	Segment Offset
001	Linear Address
010	Physical Address
011	Memory Address
100 to 110	Reserved
111	Generic

- Model Specific Information (bits 63:9): Not architecturally defined.

17.3.2.4.1 IOMCA

Logging and Signaling of errors from PCI Express domain is governed by PCI Express Advanced Error Reporting (AER) architecture. PCI Express architecture divides errors in two categories: Uncorrectable errors and Correctable errors. Uncorrectable errors can further be classified as Fatal or Non-Fatal. Uncorrected IO errors are signaled to the system software either as AER Message Signaled Interrupt (MSI) or via platform specific mechanisms such as NMI. Generally, the signaling mechanism is controlled by BIOS and/or platform firmware. Certain processors support an error handling mode, called IOMCA mode, where Uncorrected PCI Express errors are signaled in the form of machine check exception and logged in machine check banks.

When a processor is in this mode, Uncorrected PCI Express errors are logged in the MCACOD field of the IA32_MCi_STATUS register as Generic I/O error. The corresponding MCA error code is defined in Table 15-8. IA32_MCi_Status [15:0] Simple Error Code Encoding. Machine check logging complements and does not replace AER logging that occurs inside the PCI Express hierarchy. The PCI Express Root Complex and Endpoints continue to log the error in accordance with PCI Express AER mechanism. In IOMCA mode, MCI_MISC register in the bank that logged IOMCA can optionally contain information that link the Machine Check logs with the AER logs or proprietary logs. In such a scenario, the machine check handler can utilize the contents of MCI_MISC to locate the next level of error logs corresponding to the same error. Specifically, if MCI_Status.MISCV is 1 and MCACOD is 0x0E0B, MCI_MISC contains the PCI Express address of the Root Complex device containing the AER Logs. Software can consult

the header type and class code registers in the Root Complex device's PCIe Configuration space to determine what type of device it is. This Root Complex device can either be a PCI Express Root Port, PCI Express Root Complex Event Collector or a proprietary device.

Errors that originate from PCI Express or Legacy Endpoints are logged in the corresponding Root Port in addition to the generating device. If MISCV=1 and MCi_MISC contains the address of the Root Port or a Root Complex Event collector, software can parse the AER logs to learn more about the error.

If MISCV=1 and MCi_MISC points to a device that is neither a Root Complex Event Collector nor a Root Port, software must consult the Vendor ID/Device ID and use device specific knowledge to locate and interpret the error log registers. In some cases, the Root Complex device configuration space may not be accessible to the software and both the Vendor and Device ID read as 0xFFFF.

- The format of MCi_MISC for IOMCA errors is shown in Table 17-4.

Table 17-4. Address Mode in IA32_MCi_MISC[8:6]

63:40	39:32	31:16	15:9	8:6	5:0
RSVD	PCI Express Segment number	PCI Express Requestor ID	RSVD	ADDR MODE ¹	RECOV ADDR LSB ¹

NOTES:

1. Not Applicable if ADDR=0.

Refer to PCI Express Specification 3.0 for definition of PCI Express Requestor ID and AER architecture. Refer to PCI Firmware Specification 3.0 for an explanation of PCI Express Segment number and how software can access configuration space of a PCI Express device given the segment number and Requestor ID.

17.3.2.5 IA32_MCi_CTL2 MSRs

The IA32_MCi_CTL2 MSR provides the programming interface to use corrected MC error signaling capability that is indicated by IA32_MCG_CAP[10] = 1. Software must check for the presence of IA32_MCi_CTL2 on a per-bank basis.

When IA32_MCG_CAP[10] = 1, the IA32_MCi_CTL2 MSR for each bank exists, i.e., reads and writes to these MSR are supported. However, signaling interface for corrected MC errors may not be supported in all banks.

The layout of IA32_MCi_CTL2 is shown in Figure 17-9.

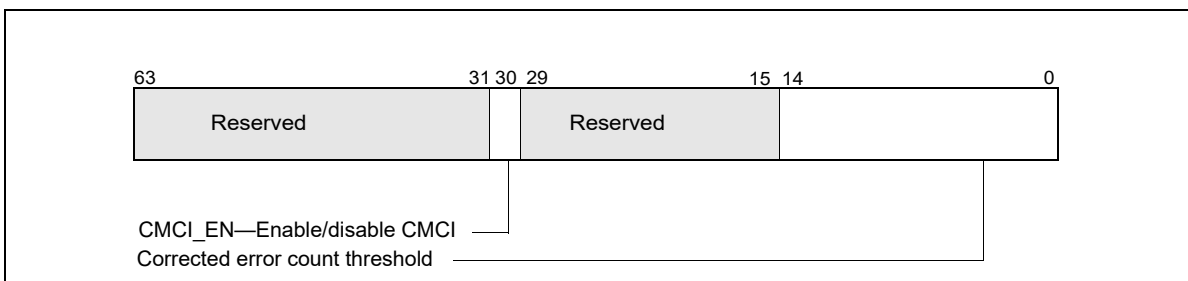


Figure 17-9. IA32_MCi_CTL2 Register

- **Corrected error count threshold, bits 14:0** — Software must initialize this field. The value is compared with the corrected error count field in IA32_MCi_STATUS, bits 38 through 52. An overflow event is signaled to the CMCI LVT entry (see Table 12-1) in the APIC when the count value equals the threshold value. The new LVT entry in the APIC is at 02F0H offset from the APIC_BASE. If CMCI interface is not supported for a particular bank (but IA32_MCG_CAP[10] = 1), this field will always read 0.
- **CMCI_EN (Corrected error interrupt enable/disable/indicator), bits 30** — Software sets this bit to enable the generation of corrected machine-check error interrupt (CMCI). If CMCI interface is not supported for a particular bank (but IA32_MCG_CAP[10] = 1), this bit is writeable but will always return 0 for that bank. This

bit also indicates CMCI is supported or not supported in the corresponding bank. See Section 17.5 for details of software detection of CMCI facility.

Some microarchitectural sub-systems that are the source of corrected MC errors may be shared by more than one logical processors. Consequently, the facilities for reporting MC errors and controlling mechanisms may be shared by more than one logical processors. For example, the IA32_MCi_CTL2 MSR is shared between logical processors sharing a processor core. Software is responsible to program IA32_MCi_CTL2 MSR in a consistent manner with CMCI delivery and usage.

After processor reset, IA32_MCi_CTL2 MSRs are zeroed.

17.3.2.6 IA32_MCG Extended Machine Check State MSRs

The Pentium 4 and Intel Xeon processors implement a variable number of extended machine-check state MSRs. The MCG_EXT_P flag in the IA32_MCG_CAP MSR indicates the presence of these extended registers, and the MCG_EXT_CNT field indicates the number of these registers actually implemented. See Section 17.3.1.1, “IA32_MCG_CAP MSR.” Also see Table 17-5.

Table 17-5. Extended Machine Check State MSRs in Processors Without Support for Intel® 64 Architecture

MSR	Address	Description
IA32_MCG_EAX	180H	Contains state of the EAX register at the time of the machine-check error.
IA32_MCG_EBX	181H	Contains state of the EBX register at the time of the machine-check error.
IA32_MCG_ECX	182H	Contains state of the ECX register at the time of the machine-check error.
IA32_MCG_EDX	183H	Contains state of the EDX register at the time of the machine-check error.
IA32_MCG_ESI	184H	Contains state of the ESI register at the time of the machine-check error.
IA32_MCG_EDI	185H	Contains state of the EDI register at the time of the machine-check error.
IA32_MCG_EBP	186H	Contains state of the EBP register at the time of the machine-check error.
IA32_MCG_ESP	187H	Contains state of the ESP register at the time of the machine-check error.
IA32_MCG_EFLAGS	188H	Contains state of the EFLAGS register at the time of the machine-check error.
IA32_MCG_EIP	189H	Contains state of the EIP register at the time of the machine-check error.
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.

In processors with support for Intel 64 architecture, 64-bit machine check state MSRs are aliased to the legacy MSRs. In addition, there may be registers beyond IA32_MCG_MISC. These may include up to five reserved MSRs (IA32_MCG_RESERVED[1:5]) and save-state MSRs for registers introduced in 64-bit mode. See Table 17-6.

Table 17-6. Extended Machine Check State MSRs In Processors With Support for Intel® 64 Architecture

MSR	Address	Description
IA32_MCG_RAX	180H	Contains state of the RAX register at the time of the machine-check error.
IA32_MCG_RBX	181H	Contains state of the RBX register at the time of the machine-check error.
IA32_MCG_RCX	182H	Contains state of the RCX register at the time of the machine-check error.
IA32_MCG_RDX	183H	Contains state of the RDX register at the time of the machine-check error.
IA32_MCG_RSI	184H	Contains state of the RSI register at the time of the machine-check error.
IA32_MCG_RDI	185H	Contains state of the RDI register at the time of the machine-check error.
IA32_MCG_RBP	186H	Contains state of the RBP register at the time of the machine-check error.
IA32_MCG_RSP	187H	Contains state of the RSP register at the time of the machine-check error.
IA32_MCG_RFLAGS	188H	Contains state of the RFLAGS register at the time of the machine-check error.
IA32_MCG_RIP	189H	Contains state of the RIP register at the time of the machine-check error.

Table 17-6. Extended Machine Check State MSRs In Processors With Support for Intel® 64 Architecture (Contd.)

MSR	Address	Description
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.
IA32_MCG_RESERVED[1:5]	18BH-18FH	These registers, if present, are reserved.
IA32_MCG_R8	190H	Contains state of the R8 register at the time of the machine-check error.
IA32_MCG_R9	191H	Contains state of the R9 register at the time of the machine-check error.
IA32_MCG_R10	192H	Contains state of the R10 register at the time of the machine-check error.
IA32_MCG_R11	193H	Contains state of the R11 register at the time of the machine-check error.
IA32_MCG_R12	194H	Contains state of the R12 register at the time of the machine-check error.
IA32_MCG_R13	195H	Contains state of the R13 register at the time of the machine-check error.
IA32_MCG_R14	196H	Contains state of the R14 register at the time of the machine-check error.
IA32_MCG_R15	197H	Contains state of the R15 register at the time of the machine-check error.

When a machine-check error is detected on a Pentium 4 or Intel Xeon processor, the processor saves the state of the general-purpose registers, the R/EFLAGS register, and the R/EIP in these extended machine-check state MSRs. This information can be used by a debugger to analyze the error.

These registers are read/write to zero registers. This means software can read them; but if software writes to them, only all zeros is allowed. If software attempts to write a non-zero value into one of these registers, a general-protection (#GP) exception is generated. These registers are cleared on a hardware reset (power-up or RESET), but maintain their contents following a soft reset (INIT reset).

17.3.3 Mapping of the Pentium Processor Machine-Check Errors to the Machine-Check Architecture

The Pentium processor reports machine-check errors using two registers: P5_MC_TYPE and P5_MC_ADDR. The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors map these registers to the IA32_MCi_STATUS and IA32_MCi_ADDR in the error-reporting register bank. This bank reports on the same type of external bus errors reported in P5_MC_TYPE and P5_MC_ADDR.

The information in these registers can then be accessed in two ways:

- By reading the IA32_MCi_STATUS and IA32_MCi_ADDR registers as part of a general machine-check exception handler written for Pentium 4, Intel Atom and P6 family processors.
- By reading the P5_MC_TYPE and P5_MC_ADDR registers using the RDMSR instruction.

The second capability permits a machine-check exception handler written to run on a Pentium processor to be run on a Pentium 4, Intel Xeon, Intel Atom, or P6 family processor. There is a limitation in that information returned by the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors is encoded differently than information returned by the Pentium processor. To run a Pentium processor machine-check exception handler on a Pentium 4, Intel Xeon, Intel Atom, or P6 family processor; the handler must be written to interpret P5_MC_TYPE encodings correctly.

17.4 ENHANCED CACHE ERROR REPORTING

Starting with Intel Core Duo processors, cache error reporting was enhanced. In earlier Intel processors, cache status was based on the number of correction events that occurred in a cache. In the new paradigm, called “threshold-based error status”, cache status is based on the number of lines (ECC blocks) in a cache that incur repeated corrections. The threshold is chosen by Intel, based on various factors. If a processor supports threshold-based error status, it sets IA32_MCG_CAP[11] (MCG_TES_P) to 1; if not, to 0.

A processor that supports enhanced cache error reporting contains hardware that tracks the operating status of certain caches and provides an indicator of their “health”. The hardware reports a “green” status when the number

of lines that incur repeated corrections is at or below a pre-defined threshold, and a “yellow” status when the number of affected lines exceeds the threshold. Yellow status means that the cache reporting the event is operating correctly, but you should schedule the system for servicing within a few weeks.

Intel recommends that you rely on this mechanism for structures supported by threshold-base error reporting.

The CPU/system/platform response to a yellow event should be less severe than its response to an uncorrected error. An uncorrected error means that a serious error has actually occurred, whereas the yellow condition is a warning that the number of affected lines has exceeded the threshold but is not, in itself, a serious event: the error was corrected and system state was not compromised.

The green/yellow status indicator is not a foolproof early warning for an uncorrected error resulting from the failure of two bits in the same ECC block. Such a failure can occur and cause an uncorrected error before the yellow threshold is reached. However, the chance of an uncorrected error increases as the number of affected lines increases.

17.5 CORRECTED MACHINE CHECK ERROR INTERRUPT

Corrected machine-check error interrupt (CMCI) is an architectural enhancement to the machine-check architecture. It provides capabilities beyond those of threshold-based error reporting (Section 17.4). With threshold-based error reporting, software is limited to use periodic polling to query the status of hardware corrected MC errors. CMCI provides a signaling mechanism to deliver a local interrupt based on threshold values that software can program using the IA32_MCi_CTL2 MSRs.

CMCI is disabled by default. System software is required to enable CMCI for each IA32_MCi bank that support the reporting of hardware corrected errors if IA32_MCG_CAP[10] = 1.

System software use IA32_MCi_CTL2 MSR to enable/disable the CMCI capability for each bank and program threshold values into IA32_MCi_CTL2 MSR. CMCI is not affected by the CR4.MCE bit, and it is not affected by the IA32_MCi_CTL MSRs.

To detect the existence of thresholding for a given bank, software writes only bits 14:0 with the threshold value. If the bits persist, then thresholding is available (and CMCI is available). If the bits are all 0's, then no thresholding exists. To detect that CMCI signaling exists, software writes a 1 to bit 30 of the MCI_CTL2 register. Upon subsequent read, if bit 30 = 0, no CMCI is available for this bank and no corrected or UCNA errors will be reported on this bank. If bit 30 = 1, then CMCI is available and enabled.

17.5.1 CMCI Local APIC Interface

The operation of CMCI is depicted in Figure 17-10.

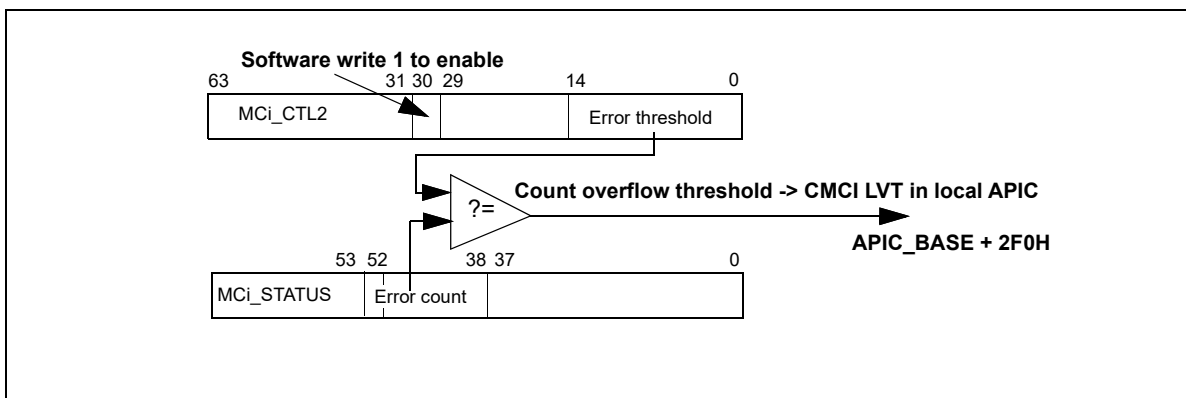


Figure 17-10. CMCI Behavior

CMCI interrupt delivery is configured by writing to the LVT CMCI register entry in the local APIC register space at default address of APIC_BASE + 2F0H. A CMCI interrupt can be delivered to more than one logical processors if multiple logical processors are affected by the associated MC errors. For example, if a corrected bit error in a cache

shared by two logical processors caused a CMCI, the interrupt will be delivered to both logical processors sharing that microarchitectural sub-system. Similarly, package level errors may cause CMCI to be delivered to all logical processors within the package. However, system level errors will not be handled by CMCI.

See Section 12.5.1, “Local Vector Table,” for details regarding the LVT CMCI register.

17.5.2 System Software Recommendation for Managing CMCI and Machine Check Resources

System software must enable and manage CMCI, set up interrupt handlers to service CMCI interrupts delivered to affected logical processors, program CMCI LVT entry, and query machine check banks that are shared by more than one logical processors.

This section describes techniques system software can implement to manage CMCI initialization, service CMCI interrupts in an efficient manner to minimize contentions to access shared MSR resources.

17.5.2.1 CMCI Initialization

Although a CMCI interrupt may be delivered to more than one logical processors depending on the nature of the corrected MC error, only one instance of the interrupt service routine needs to perform the necessary service and make queries to the machine-check banks. The following steps describes a technique that limits the amount of work the system has to do in response to a CMCI.

- To provide maximum flexibility, system software should define per-thread data structure for each logical processor to allow equal-opportunity and efficient response to interrupt delivery. Specifically, the per-thread data structure should include a set of per-bank fields to track which machine check bank it needs to access in response to a delivered CMCI interrupt. The number of banks that needs to be tracked is determined by `IA32_MCG_CAP[7:0]`.
- Initialization of per-thread data structure. The initialization of per-thread data structure must be done serially on each logical processor in the system. The sequencing order to start the per-thread initialization between different logical processor is arbitrary. But it must observe the following specific detail to satisfy the shared nature of specific MSR resources:
 - a. Each thread initializes its data structure to indicate that it does not own any MC bank registers.
 - b. Each thread examines `IA32_MCi_CTL2[30]` indicator for each bank to determine if another thread has already claimed ownership of that bank.
 - If `IA32_MCi_CTL2[30]` had been set by another thread. This thread can not own bank *i* and should proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
 - If `IA32_MCi_CTL2[30] = 0`, proceed to step c.
 - c. Check whether writing a 1 into `IA32_MCi_CTL2[30]` can return with 1 on a subsequent read to determine this bank can support CMCI.
 - If `IA32_MCi_CTL2[30] = 0`, this bank does not support CMCI. This thread can not own bank *i* and should proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
 - If `IA32_MCi_CTL2[30] = 1`, modify the per-thread data structure to indicate this thread claims ownership to the MC bank; proceed to initialize the error threshold count (bits 15:0) of that bank as described in Chapter 17, “CMCI Threshold Management”. Then proceed to step b. and examine the next machine check bank until all of the machine check banks are exhausted.
- After the thread has examined all of the machine check banks, it sees if it owns any MC banks to service CMCI. If any bank has been claimed by this thread:
 - Ensure that the CMCI interrupt handler has been set up as described in Chapter 17, “CMCI Interrupt Handler”.
 - Initialize the CMCI LVT entry, as described in Section 17.5.1, “CMCI Local APIC Interface.”
 - Log and clear all of `IA32_MCi_Status` registers for the banks that this thread owns. This will allow new errors to be logged.

17.5.2.2 CMCI Threshold Management

The Corrected MC error threshold field, IA32_MCi_CTL2[14:0], is architecturally defined. Specifically, all these bits are writable by software, but different processor implementations may choose to implement less than 15 bits as threshold for the overflow comparison with IA32_MCi_STATUS[52:38]. The following describes techniques that software can manage CMCI threshold to be compatible with changes in implementation characteristics:

- Software can set the initial threshold value to 1 by writing 1 to IA32_MCi_CTL2[14:0]. This will cause overflow condition on every corrected MC error and generates a CMCI interrupt.
- To increase the threshold and reduce the frequency of CMCI servicing:
 - a. Find the maximum threshold value a given processor implementation supports. The steps are:
 - Write 7FFFH to IA32_MCi_CTL2[14:0],
 - Read back IA32_MCi_CTL2[14:0]; these 15 bits (14:0) contain the maximum threshold supported by the processor.
 - b. Increase the threshold to a value below the maximum value discovered using step a.

17.5.2.3 CMCI Interrupt Handler

The following describes techniques system software may consider to implement a CMCI service routine:

- The service routine examines its private per-thread data structure to check which set of MC banks it has ownership. If the thread does not have ownership of a given MC bank, proceed to the next MC bank. Ownership is determined at initialization time which is described in Section 17.5.2.1.

If the thread had claimed ownership to an MC bank, this technique will allow each logical processors to handle corrected MC errors independently and requires no synchronization to access shared MSR resources. Consult Example 17-5 for guidelines on logging when processing CMCI.

17.6 RECOVERY OF UNCORRECTED RECOVERABLE (UCR) ERRORS

Recovery of uncorrected recoverable machine check errors is an enhancement in machine-check architecture. The first processor that supports this feature is 45 nm Intel 64 processor on which CPUID reports DisplayFamily_DisplayModel as 06H_2EH; see the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A. This allows system software to perform recovery action on a certain class of uncorrected errors and continue execution.

17.6.1 Detection of Software Error Recovery Support

Software must use bit 24 of IA32_MCG_CAP (MCG_SER_P) to detect the presence of software error recovery support (see Figure 17-2). When IA32_MCG_CAP[24] is set, this indicates that the processor supports software error recovery. When this bit is clear, this indicates that there is no support for error recovery from the processor and the primary responsibility of the machine check handler is logging the machine check error information and shutting down the system.

The new class of architectural MCA errors from which system software can attempt recovery is called Uncorrected Recoverable (UCR) Errors. UCR errors are uncorrected errors that have been detected and signaled but have not corrupted the processor context. For certain UCR errors, this means that once system software has performed a certain recovery action, it is possible to continue execution on this processor. UCR error reporting provides an error containment mechanism for data poisoning. The machine check handler will use the error log information from the error reporting registers to analyze and implement specific error recovery actions for UCR errors.

17.6.2 UCR Error Reporting and Logging

IA32_MCi_STATUS MSR is used for reporting UCR errors and existing corrected or uncorrected errors. The definitions of IA32_MCi_STATUS, including bit fields to identify UCR errors, is shown in Figure 17-6. UCR errors can be

signaled through either the corrected machine check interrupt (CMCI) or machine check exception (MCE) path depending on the type of the UCR error.

When IA32_MCG_CAP[24] is set, a UCR error is indicated by the following bit settings in the IA32_MCi_STATUS register:

- Valid (bit 63) = 1
- UC (bit 61) = 1
- PCC (bit 57) = 0

Additional information from the IA32_MCi_MISC and the IA32_MCi_ADDR registers for the UCR error are available when the ADDR_V and the MISC_V flags in the IA32_MCi_STATUS register are set (see Section 17.3.2.4). The MCA error code field of the IA32_MCi_STATUS register indicates the type of UCR error. System software can interpret the MCA error code field to analyze and identify the necessary recovery action for the given UCR error.

In addition, the IA32_MCi_STATUS register bit fields, bits 56:55, are defined (see Figure 17-6) to provide additional information to help system software to properly identify the necessary recovery action for the UCR error:

- S (Signaling) flag, bit 56 - Indicates (when set) that a machine check exception was generated for the UCR error reported in this MC bank and system software needs to check the AR flag and the MCA error code fields in the IA32_MCi_STATUS register to identify the necessary recovery action for this error. When the S flag in the IA32_MCi_STATUS register is clear, this UCR error was not signaled via a machine check exception and instead was reported as a corrected machine check (CMC). System software is not required to take any recovery action when the S flag in the IA32_MCi_STATUS register is clear.
- AR (Action Required) flag, bit 55 - Indicates (when set) that MCA error code specific recovery action must be performed by system software at the time this error was signaled. This recovery action must be completed successfully before any additional work is scheduled for this processor. When the RIP_V flag in the IA32_MCG_STATUS is clear, an alternative execution stream needs to be provided; when the MCA error code specific recovery specific recovery action cannot be successfully completed, system software must shut down the system. When the AR flag in the IA32_MCi_STATUS register is clear, system software may still take MCA error code specific recovery action but this is optional; system software can safely resume program execution at the instruction pointer saved on the stack from the machine check exception when the RIP_V flag in the IA32_MCG_STATUS register is set.

Both the S and the AR flags in the IA32_MCi_STATUS register are defined to be sticky bits, which mean that once set, the processor does not clear them. Only software and good power-on reset can clear the S and the AR-flags. Both the S and the AR flags are only set when the processor reports the UCR errors (MCG_CAP[24] is set).

17.6.3 UCR Error Classification

With the S and AR flag encoding in the IA32_MCi_STATUS register, UCR errors can be classified as:

- Uncorrected no action required (UCNA) - is a UCR error that is not signaled via a machine check exception and, instead, is reported to system software as a corrected machine check error. UCNA errors indicate that some data in the system is corrupted, but the data has not been consumed and the processor state is valid and you may continue execution on this processor. UCNA errors require no action from system software to continue execution. A UCNA error is indicated with UC=1, PCC=0, S=0 and AR=0 in the IA32_MCi_STATUS register.
- Software recoverable action optional (SRAO) - a UCR error is signaled either via a machine check exception or CMCI. System software recovery action is optional and not required to continue execution from this machine check exception. SRAO errors indicate that some data in the system is corrupt, but the data has not been consumed and the processor state is valid. SRAO errors provide the additional error information for system software to perform a recovery action. An SRAO error when signaled as a machine check is indicated with UC=1, PCC=0, S=1, EN=1 and AR=0 in the IA32_MCi_STATUS register. In cases when SRAO is signaled via CMCI the error signature is indicated via UC=1, PCC=0, S=0. Recovery actions for SRAO errors are MCA error code specific. The MISC_V and the ADDR_V flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAO error. If MISC_V and ADDR_V are not set, it is recommended that no system software error recovery be performed however, system software can resume execution.
- Software recoverable action required (SRAR) - a UCR error that requires system software to take a recovery action on this processor before scheduling another stream of execution on this processor. SRAR errors indicate

that the error was detected and raised at the point of the consumption in the execution flow. An SRAR error is indicated with UC=1, PCC=0, S=1, EN=1 and AR=1 in the IA32_MCi_STATUS register. Recovery actions are MCA error code specific. The MISCV and the ADDR_V flags in the IA32_MCi_STATUS register are set when the additional error information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. System software needs to inspect the MCA error code fields in the IA32_MCi_STATUS register to identify the specific recovery action for a given SRAR error. If MISCV and ADDR_V are not set, it is recommended that system software shutdown the system.

Table 17-7 summarizes UCR, corrected, and uncorrected errors.

Table 17-7. MC Error Classifications

Type of Error ¹	UC	EN	PCC	S	AR	Signaling	Software Action	Example
Uncorrected Error (UC)	1	1	1	x	x	MCE	If EN=1, reset the system, else log and OK to keep the system running.	
SRAR	1	1	0	1	1	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, must bugcheck. If OVER=1, reset system, else take specific recovery action.	Cache to processor load error.
SRAO	1	x ²	0	x ²	0	MCE/CMC	For known MCACOD, take specific recovery action; For unknown MCACOD, OK to keep the system running.	Patrol scrub and explicit writeback poison errors.
UCNA	1	x	0	0	0	CMC	Log the error and OK to keep the system running.	Poison detection error.
Corrected Error (CE)	0	x	x	x	x	CMC	Log the error and no corrective action required.	ECC in caches and memory.

NOTES:

1. SRAR, SRAO and UCNA errors are supported by the processor only when IA32_MCG_CAP[24] (MCG_SER_P) is set.
2. EN=1, S=1 when signaled via MCE. EN=x, S=0 when signaled via CMC.

17.6.4 UCR Error Overwrite Rules

In general, the overwrite rules are as follows:

- UCR errors will overwrite corrected errors.
- Uncorrected (PCC=1) errors overwrite UCR (PCC=0) errors.
- UCR errors are not written over previous UCR errors.
- Corrected errors do not write over previous UCR errors.

Regardless of whether the 1st error is retained or the 2nd error is overwritten over the 1st error, the OVER flag in the IA32_MCi_STATUS register will be set to indicate an overflow condition. As the S flag and AR flag in the IA32_MCi_STATUS register are defined to be sticky flags, a second event cannot clear these 2 flags once set, however the MC bank information may be filled in for the 2nd error. The table below shows the overwrite rules and how to treat a second error if the first event is already logged in a MC bank along with the resulting bit setting of the UC, PCC, and AR flags in the IA32_MCi_STATUS register. As UCNA and SRAO errors do not require recovery action from system software to continue program execution, a system reset by system software is not required unless the AR flag or PCC flag is set for the UCR overflow case (OVER=1, VAL=1, UC=1, PCC=0).

Table 17-8 lists overwrite rules for uncorrected errors, corrected errors, and uncorrected recoverable errors.

Table 17-8. Overwrite Rules for UC, CE, and UCR Errors

First Event	Second Event	UC	PCC	S	AR	MCA Bank	Reset System
CE	UCR	1	0	0 if UCNA, else 1	1 if SRAR, else 0	second	yes, if AR=1
UCR	CE	1	0	0 if UCNA, else 1	1 if SRAR, else 0	first	yes, if AR=1

Table 17-8. Overwrite Rules for UC, CE, and UCR Errors

First Event	Second Event	UC	PCC	S	AR	MCA Bank	Reset System
UCNA	UCNA	1	0	0	0	first	no
UCNA	SRAO	1	0	1	0	first	no
UCNA	SRAR	1	0	1	1	first	yes
SRAO	UCNA	1	0	1	0	first	no
SRAO	SRAO	1	0	1	0	first	no
SRAO	SRAR	1	0	1	1	first	yes
SRAR	UCNA	1	0	1	1	first	yes
SRAR	SRAO	1	0	1	1	first	yes
SRAR	SRAR	1	0	1	1	first	yes
UCR	UC	1	1	undefined	undefined	second	yes
UC	UCR	1	1	undefined	undefined	first	yes

17.7 STATIC LOCKSTEP MODE

Static Lockstep Mode (SLSM) places two cores into a state where they execute identical operations in parallel. Every input to both cores in the core-pair is duplicated cycle-by-cycle, while every output of both cores in the core-pair is compared cycle-by-cycle, in order to identify divergences and enable a high reliability execution mode. One core of each core-pair is the active core and it is available to software for execution; the other core is the shadow core and is unavailable for functional use. Only the outputs of the active core in each core-pair have normal access to external functional machine hardware. The shadow core exists only to perform comparison and its outputs are otherwise ignored.

A miscompare between one of the outputs within a core-pair results in an uncorrected error logged in the associated IA32_MCI_STATUS register and will cause a machine check exception. See FRC Error in Table 17-9, “Simple Error Code Encoding,” for more information.

BIOS is responsible for activating SLSM on the appropriate core-pairs within a system. This enables legacy software compatibility with SLSM execution because software is not required to have visibility into SLSM status. Once enabled, SLSM remains active until the next boot.

CPUID.07H.01H:EDX.STATIC_LOCKSTEP_MODE[24]=1 enumerates support for SLSM. Software can determine that a core is executing in SLSM by reading IA32_INTEGRITY_STATUS[STATIC_LOCKSTEP].

17.8 MACHINE-CHECK AVAILABILITY

The machine-check architecture and machine-check exception (#MC) are model-specific features. Software can execute the CPUID instruction to determine whether a processor implements these features. Following the execution of the CPUID instruction, the settings of the MCA flag (bit 14) and MCE flag (bit 7) in EDX indicate whether the processor implements the machine-check architecture and machine-check exception.

17.9 MACHINE-CHECK INITIALIZATION

To use the processors machine-check architecture, software must initialize the processor to activate the machine-check exception and the error-reporting mechanism.

Example 17-1 gives pseudocode for performing this initialization. This pseudocode checks for the existence of the machine-check architecture and exception; it then enables machine-check exception and the error-reporting register banks. The pseudocode shown is compatible with the Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors.

Following power up or power cycling, IA32_MCi_STATUS registers are not guaranteed to have valid data until after they are initially cleared to zero by software (as shown in the initialization pseudocode in Example 17-1).

Example 17-1. Machine-Check Initialization Pseudocode

Check CPUID Feature Flags for MCE and MCA support

IF CPU supports MCE

THEN

IF CPU supports MCA

THEN

IF (IA32_MCG_CAP.MCG_CTL_P = 1)

(* IA32_MCG_CTL register is present *)

THEN

IA32_MCG_CTL ← FFFFFFFFFFFFFFFFH;

(* enables all MCA features *)

FI

IF (IA32_MCG_CAP.MCG_LMCE_P = 1 and IA32_FEATURE_CONTROL.LOCK = 1 and IA32_FEATURE_CONTROL.LMCE_ENABLED = 1)

(* IA32_MCG_EXT_CTL register is present and platform has enabled LMCE to permit system software to use LMCE *)

THEN

IA32_MCG_EXT_CTL ← IA32_MCG_EXT_CTL | 01H;

(* System software enables LMCE capability for hardware to signal MCE to a single logical processor*)

FI

(* Determine number of error-reporting banks supported *)

COUNT ← IA32_MCG_CAP.Count;

MAX_BANK_NUMBER ← COUNT - 1;

IF (Processor Family is 6H and Processor EXTMODEL:MODEL is less than 1AH)

THEN

(* Enable logging of all errors except for MCO_CTL register *)

FOR error-reporting banks (1 through MAX_BANK_NUMBER)

DO

IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFH;

OD

ELSE

(* Enable logging of all errors including MCO_CTL register *)

FOR error-reporting banks (0 through MAX_BANK_NUMBER)

DO

IA32_MCi_CTL ← 0FFFFFFFFFFFFFFFH;

OD

FI

(* BIOS clears all errors only on power-on reset *)

IF (BIOS detects Power-on reset)

THEN

FOR error-reporting banks (0 through MAX_BANK_NUMBER)

DO

IA32_MCi_STATUS ← 0;

OD

ELSE

FOR error-reporting banks (0 through MAX_BANK_NUMBER)

DO

(Optional for BIOS and OS) Log valid errors

(OS only) IA32_MCi_STATUS ← 0;

OD

FI

FI

Setup the Machine Check Exception (#MC) handler for vector 18 in IDT

Set the MCE bit (bit 6) in CR4 register to enable Machine-Check Exceptions

FI

17.10 INTERPRETING THE MCA ERROR CODES

When the processor detects a machine-check error condition, it writes a 16-bit error code to the MCA error code field of one of the IA32_MCI_STATUS registers and sets the VAL (valid) flag in that register. The processor may also write a 16-bit model-specific error code in the IA32_MCI_STATUS register depending on the implementation of the machine-check architecture of the processor.

The MCA error codes are architecturally defined for Intel 64 and IA-32 processors. To determine the cause of a machine-check exception, the machine-check exception handler must read the VAL flag for each IA32_MCI_STATUS register. If the flag is set, the machine check-exception handler must then read the MCA error code field of the register. It is the encoding of the MCA error code field [15:0] that determines the type of error being reported and not the register bank reporting it.

There are two types of MCA error codes: simple error codes and compound error codes.

17.10.1 Simple Error Codes

Table 17-9 shows the simple error codes. These unique codes indicate global error information.

Table 17-9. IA32_MCI_Status [15:0] Simple Error Code Encoding

Error Code	Binary Encoding	Meaning
No Error	0000 0000 0000 0000	No error has been reported to this bank of error-reporting registers.
Unclassified	0000 0000 0000 0001	This error has not been classified into the MCA error classes.
Microcode ROM Parity Error	0000 0000 0000 0010	Parity error in internal microcode ROM
External Error	0000 0000 0000 0011	The BINIT# from another processor caused this processor to enter machine check. ¹
FRC Error	0000 0000 0000 0100	FRC (functional redundancy check) main/secondary error.
Internal Parity Error	0000 0000 0000 0101	Internal parity error.
SMM Handler Code Access Violation	0000 0000 0000 0110	An attempt was made by the SMM Handler to execute outside the ranges specified by SMRR.
Internal Timer Error	0000 0100 0000 0000	Internal timer error.
I/O Error	0000 1110 0000 1011	generic I/O error.
Internal Unclassified	0000 01xx xxxx xxxx	Internal unclassified errors. ²

NOTES:

1. BINIT# assertion will cause a machine check exception if the processor (or any processor on the same external bus) has BINIT# observation enabled during power-on configuration (hardware strapping) and if machine check exceptions are enabled (by setting CR4.MCE = 1).
2. At least one X must equal one. Internal unclassified errors have not been classified.

17.10.2 Compound Error Codes

Compound error codes describe errors related to the TLBs, memory, caches, bus and interconnect logic, and internal timer. A set of sub-fields is common to all of compound errors. These sub-fields describe the type of access, level in the cache hierarchy, and type of request. Table 17-10 shows the general form of the compound error codes.

Table 17-10. IA32_MCI_Status [15:0] Compound Error Code Encoding

Type	Form	Interpretation
Generic Cache Hierarchy	000F 0000 0000 11LL	Generic cache hierarchy error
TLB Errors	000F 0000 0001 TTLL	{TT}TLB{LL}_ERR
Memory Controller Errors	000F 0000 1MMM CCCC	{MMM}_CHANNEL{CCCC}_ERR

Table 17-10. IA32_MCi_Status [15:0] Compound Error Code Encoding (Contd.)

Cache Hierarchy Errors	000F 0001 RRRR TTLL	{TT}CACHE{LL}_{RRRR}_ERR
Extended Memory Errors	000F 0010 MMM CCCC	{MMM}_CHANNEL{CCCC}_ERR
Bus and Interconnect Errors	000F 1PPT RRRR IILL	BUS{LL}_{PP}_{RRRR}_{II}_{T}_ERR

The “Interpretation” column in the table indicates the name of a compound error. The name is constructed by substituting mnemonics for the sub-field names given within curly braces. For example, the error code ICACHEL1_RD_ERR is constructed from the form:

{TT}CACHE{LL}_{RRRR}_ERR,
where {TT} is replaced by I, {LL} is replaced by L1, and {RRRR} is replaced by RD.

For more information on the “Form” and “Interpretation” columns, see Section 17.10.2.1, “Correction Report Filtering (F) Bit,” through Section 17.10.2.5, “Bus and Interconnect Errors.”

17.10.2.1 Correction Report Filtering (F) Bit

Starting with Intel Core Duo processors, bit 12 in the “Form” column in Table 17-10 is used to indicate that a particular posting to a log may be the last posting for corrections in that line/entry, at least for some time:

- 0 in bit 12 indicates “normal” filtering (original P6/Pentium4/Atom/Xeon processor meaning).
- 1 in bit 12 indicates “corrected” filtering (filtering is activated for the line/entry in the posting). Filtering means that some or all of the subsequent corrections to this entry (in this structure) will not be posted. The enhanced error reporting introduced with the Intel Core Duo processors is based on tracking the lines affected by repeated corrections (see Section 17.4, “Enhanced Cache Error reporting”). This capability is indicated by IA32_MCG_CAP[11]. Only the first few correction events for a line are posted; subsequent redundant correction events to the same line are not posted. Uncorrected events are always posted.

The behavior of error filtering after crossing the yellow threshold is model-specific. Filtering has meaning only for corrected errors (UC=0 in IA32_MCi_STATUS MSR). System software must ignore filtering bit (12) for uncorrected errors.

17.10.2.2 Transaction Type (TT) Sub-Field

The 2-bit TT sub-field (Table 17-11) indicates the type of transaction (data, instruction, or generic). The sub-field applies to the TLB, cache, and interconnect error conditions. Note that interconnect error conditions are primarily associated with P6 family and Pentium processors, which utilize an external APIC bus separate from the system bus. The generic type is reported when the processor cannot determine the transaction type.

Table 17-11. Encoding for TT (Transaction Type) Sub-Field

Transaction Type	Mnemonic	Binary Encoding
Instruction	I	00
Data	D	01
Generic	G	10

17.10.2.3 Level (LL) Sub-Field

The 2-bit LL sub-field (see Table 17-12) indicates the level in the memory hierarchy where the error occurred (level 0, level 1, level 2, or generic). The LL sub-field also applies to the TLB, cache, and interconnect error conditions. The Pentium 4, Intel Xeon, Intel Atom, and P6 family processors support two levels in the cache hierarchy and one level in the TLBs. Again, the generic type is reported when the processor cannot determine the hierarchy level.

Table 17-12. Level Encoding for LL (Memory Hierarchy Level) Sub-Field

Hierarchy Level	Mnemonic	Binary Encoding
Level 0	L0	00

Table 17-12. Level Encoding for LL (Memory Hierarchy Level) Sub-Field (Contd.)

Level 1	L1	01
Level 2	L2	10
Generic	LG	11

17.10.2.4 Request (RRRR) Sub-Field

The 4-bit RRRR sub-field (see Table 17-13) indicates the type of action associated with the error. Actions include read and write operations, prefetches, cache evictions, and snoops. Generic error is returned when the type of error cannot be determined. Generic read and generic write are returned when the processor cannot determine the type of instruction or data request that caused the error. Eviction and snoop requests apply only to the caches. All of the other requests apply to TLBs, caches, and interconnects.

Table 17-13. Encoding of Request (RRRR) Sub-Field

Request Type	Mnemonic	Binary Encoding
Generic Error	ERR	0000
Generic Read	RD	0001
Generic Write	WR	0010
Data Read	DRD	0011
Data Write	DWR	0100
Instruction Fetch	IRD	0101
Prefetch	PREFETCH	0110
Eviction	EVICT	0111
Snoop	SNOOP	1000
Page Walk	PW	1001
EPT Page Walk	EPW	1010

17.10.2.5 Bus and Interconnect Errors

The bus and interconnect errors are defined with the 2-bit PP (participation), 1-bit T (time-out), and 2-bit II (memory or I/O) sub-fields, in addition to the LL and RRRR sub-fields (see Table 17-14). The bus error conditions are implementation dependent and related to the type of bus implemented by the processor. Likewise, the interconnect error conditions are predicated on a specific implementation-dependent interconnect model that describes the connections between the different levels of the storage hierarchy. The type of bus is implementation dependent, and as such is not specified in this document. A bus or interconnect transaction consists of a request involving an address and a response.

Table 17-14. Encodings of PP, T, and II Sub-Fields

Sub-Field	Transaction	Mnemonic	Binary Encoding
PP (Participation)	Local processor* originated request	SRC	00
	Local processor* responded to request	RES	01
	Local processor* observed error as third party	OBS	10
	Generic		11
T (Time-out)	Request timed out	TIMEOUT	1
	Request did not time out	NOTIMEOUT	0
II (Memory or I/O)	Memory Access	M	00
	Reserved		01
	I/O	IO	10

Table 17-14. Encodings of PP, T, and II Sub-Fields (Contd.)

	Other transaction		11

NOTE:

- * Local processor differentiates the processor reporting the error from other system components (including the APIC, other processors, etc.).

17.10.2.6 Memory Controller and Extended Memory Errors

The memory controller errors are defined with the 3-bit MMM (memory transaction type), and 4-bit CCCC (channel) sub-fields. The encodings for MMM and CCCC are defined in Table 17-15. Extended Memory errors use the same encodings and are used to report errors in memory used as a cache.

Table 17-15. Encodings of MMM and CCCC Sub-Fields

Sub-Field	Transaction	Mnemonic	Binary Encoding
MMM	Generic undefined request	GEN	000
	Memory read error	RD	001
	Memory write error	WR	010
	Address/Command Error	AC	011
	Memory Scrubbing Error	MS	100
	Reserved		101-111
CCCC	Channel number	CHN	0000-1110
	Channel not specified		1111

Note that the CCCC channel number may be enumerated from zero separately by each memory controller on a system. On a multi-socket system, or a system with multiple memory controllers per socket, it is necessary to also consider which machine check bank logged the error. See Chapter 18 for details on specific implementations.

17.10.3 Architecturally Defined UCR Errors

Software recoverable compound error code are defined in this section.

17.10.3.1 Architecturally Defined SRAO Errors

The following two SRAO errors are architecturally defined.

- UCR Errors detected by memory controller scrubbing; and
- UCR Errors detected during L3 cache (L3) explicit writebacks.

The MCA error code encodings for these two architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 17-10). Their values and compound encoding format are given in Table 17-16.

Table 17-16. MCA Compound Error Code Encoding for SRAO Errors

Type	MCACOD Value	MCA Error Code Encoding ¹
Memory Scrubbing	COH - CFH	0000_0000_1100_CCCC 000F 0000 1MMM CCCC (Memory Controller Error), where Memory subfield MMM = 100B (memory scrubbing) Channel subfield CCCC = channel # or generic
L3 Explicit Writeback	17AH	0000_0001_0111_1010 000F 0001 RRRR TTLL (Cache Hierarchy Error) where Request subfields RRRR = 0111B (Eviction) Transaction Type subfields TT = 10B (Generic) Level subfields LL = 10B

NOTES:

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 17-17 lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAO errors.

Table 17-17. IA32_MCi_STATUS Values for SRAO Errors

SRAO Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR	MCACOD
Memory Scrubbing	1	0	1	x ¹	1	1	0	x ¹	0	COH-CFH
L3 Explicit Writeback	1	0	1	x ¹	1	1	0	x ¹	0	17AH

NOTES:

1. When signaled as MCE, EN=1 and S=1. If error was signaled via CMC, then EN=x, and S=0.

For both the memory scrubbing and L3 explicit writeback errors, the ADDRv and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the memory scrubbing and L3 explicit writeback errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

MCE signal is broadcast to all logical processors as outlined in Section 17.11.4.1. If LMCE is supported and enabled, some errors (not limited to UCR errors) may be delivered to only a single logical processor. System software should consult IA32_MCG_STATUS.LMCE_S to determine if the MCE signaled is only to this logical processor.

IA32_MCi_STATUS banks can be shared by logical processors within a core or within the same package. So several logical processors may find an SRAO error in the shared IA32_MCi_STATUS bank but other processors do not find it in any of the IA32_MCi_STATUS banks. Table 17-18 shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the memory scrubbing and L3 explicit writeback errors on both the reporting and non-reporting logical processors.

Table 17-18. IA32_MCG_STATUS Flag Indication for SRAO Errors

SRAO Type	Reporting Logical Processors		Non-reporting Logical Processors	
	RIPV	EIPV	RIPV	EIPV
Memory Scrubbing	1	0	1	0
L3 Explicit Writeback	1	0	1	0

17.10.3.2 Architecturally Defined SRAR Errors

The following six SRAR errors are architecturally defined:

- UCR Errors detected on data load;
- UCR Errors detected on data page walk;
- UCR Errors detected on data page walk on EPT;

- UCR Errors detected on instruction fetch;
- UCR Errors detected on instruction fetch page walk; and
- UCR Errors detected on instruction fetch page walk on EPT.

The MCA error code encodings for these six architecturally-defined UCR errors corresponds to sub-classes of compound MCA error codes (see Table 17-10). Their values and compound encoding format are given in Table 17-19.

Table 17-19. MCA Compound Error Code Encoding for SRAR Errors

Type	MCACOD Value	MCA Error Code Encoding ¹
Data Load	134H	0000_0001_0011_0100: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0011B (Data Load), Transaction Type subfield TT= 01B (Data), and Level subfield LL = 00B (Level 0).
Data Page Walk	194H	0000_0001_1001_0100: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1001B (Page Walk), Transaction Type subfield TT= 01B (Data), and Level subfield LL = 00B (Level 0).
Data Page Walk on EPT	1A4H	0000_0001_1010_0100: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1010B (EPT Page Walk), Transaction Type subfield TT= 01B (Data), and Level subfield LL = 00B (Level 0).
Instruction Fetch	150H	0000_0001_0101_0000: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 0101B (Instruction Fetch), Transaction Type subfield TT= 00B (Instruction), and Level subfield LL = 00B (Level 0).
Instruction Fetch Page Walk	190H	0000_0001_1001_0000: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1001B (Page Walk), Transaction Type subfield TT= 00B (Instruction), and Level subfield LL = 00B (Level 0).
Instruction Fetch Page Walk on EPT	1A0H	0000_0001_1010_0000: 000F 0001 RRRR TTLL (Cache Hierarchy Error), where Request subfield RRRR = 1010B (EPT Page Walk), Transaction Type subfield TT= 00B (Instruction), and Level subfield LL = 00B (Level 0).

NOTES:

1. Note that for both of these errors the correction report filtering (F) bit (bit 12) of the MCA error must be ignored.

Table 17-20 lists values of relevant bit fields of IA32_MCi_STATUS for architecturally defined SRAR errors.

Table 17-20. IA32_MCi_STATUS Values for All Defined SRAR Errors

SRAR Error	Valid	OVER	UC	EN	MISCV	ADDRV	PCC	S	AR
All defined SRAR errors defined in Table 17-19	1	0	1	1	1	1	0	1	1

For all defined SRAR errors, the ADDRv and MISCV flags in the IA32_MCi_STATUS register are set to indicate that the offending physical address information is available from the IA32_MCi_MISC and the IA32_MCi_ADDR registers. For the data load and instruction fetch errors, the address mode in the IA32_MCi_MISC register should be set as physical address mode (010b) and the address LSB information in the IA32_MCi_MISC register should indicate the lowest valid address bit in the address information provided from the IA32_MCi_ADDR register.

MCE signal is broadcast to all logical processors on the system on which the UCR errors are supported, except when the processor supports LMCE and LMCE is enabled by system software (see Section 17.3.1.5). The IA32_MC-

G_STATUS MSR allows system software to distinguish the affected logical processor of an SRAR error amongst logical processors that observed SRAR via MCI_STATUS bank.

Table 17-21 shows the RIPV and EIPV flag indication in the IA32_MCG_STATUS register for the data load and instruction fetch errors on both the reporting and non-reporting logical processors. The recoverable SRAR error reported by a processor may be continuable, where the system software can interpret the context of continuable as follows: the error was isolated, contained. If software can rectify the error condition in the current instruction stream, the execution context on that logical processor can be continued without loss of information.

Table 17-21. IA32_MCG_STATUS Flag Indication for SRAR Errors

SRAR Type	Affected Logical Processor			Non-Affected Logical Processors		
	RIPV	EIPV	Continuable	RIPV	EIPV	Continuable
Recoverable-continuable	1	1	Yes ¹	1	0	Yes
Recoverable-not-continuable	0	x	No			

NOTES:

1. See the definition of the context of “continuable” above and additional details below.

SRAR Error And Affected Logical Processors

The affected logical processor is the one that has detected and raised an SRAR error at the point of the consumption in the execution flow. The affected logical processor should find the Data Load or the Instruction Fetch error information in the IA32_MCI_STATUS register that is reporting the SRAR error.

Table 17-21 list the actionable scenarios that system software can respond to an SRAR error on an affected logical processor according to RIPV and EIPV values:

- Recoverable-continuable SRAR Error (RIPV=1, EIPV=1):
For recoverable-continuable SRAR errors, the affected logical processor should find that both the IA32_MCG_STATUS.RIPV and the IA32_MCG_STATUS.EIPV flags are set, indicating that system software may be able to restart execution from the interrupted context if it is able to rectify the error condition. If system software cannot rectify the error condition then it must treat the error as a recoverable error where restarting execution with the interrupted context is not possible. Restarting without rectifying the error condition will result in most cases with another SRAR error on the same instruction.
- Recoverable-not-continuable SRAR Error (RIPV=0, EIPV=x):
For recoverable-not-continuable errors, the affected logical processor should find that either
 - IA32_MCG_STATUS.RIPV= 0, IA32_MCG_STATUS.EIPV=1, or
 - IA32_MCG_STATUS.RIPV= 0, IA32_MCG_STATUS.EIPV=0.
 In either case, this indicates that the error is detected at the instruction pointer saved on the stack for this machine check exception and restarting execution with the interrupted context is not possible. System software may take the following recovery actions for the affected logical processor:
 - The current executing thread cannot be continued. System software must terminate the interrupted stream of execution and provide a new stream of execution on return from the machine check handler for the affected logical processor.

SRAR Error And Non-Affected Logical Processors

The logical processors that observed but not affected by an SRAR error should find that the RIPV flag in the IA32_MCG_STATUS register is set and the EIPV flag in the IA32_MCG_STATUS register is cleared, indicating that it is safe to restart the execution at the instruction saved on the stack for the machine check exception on these processors after the recovery action is successfully taken by system software.

17.10.4 Multiple MCA Errors

When multiple MCA errors are detected within a certain detection window, the processor may aggregate the reporting of these errors together as a single event, i.e., a single machine exception condition. If this occurs,

system software may find multiple MCA errors logged in different MC banks on one logical processor or find multiple MCA errors logged across different processors for a single machine check broadcast event. In order to handle multiple UCR errors reported from a single machine check event and possibly recover from multiple errors, system software may consider the following:

- Whether it can recover from multiple errors is determined by the most severe error reported on the system. If the most severe error is found to be an unrecoverable error (VAL=1, UC=1, PCC=1 and EN=1) after system software examines the MC banks of all processors to which the MCA signal is broadcast, recovery from the multiple errors is not possible and system software needs to reset the system.
- When multiple recoverable errors are reported and no other fatal condition (e.g., overflowed condition for SRAR error) is found for the reported recoverable errors, it is possible for system software to recover from the multiple recoverable errors by taking necessary recovery action for each individual recoverable error. However, system software can no longer expect one to one relationship with the error information recorded in the IA32_MCi_STATUS register and the states of the RIPV and EIPV flags in the IA32_MCG_STATUS register as the states of the RIPV and the EIPV flags in the IA32_MCG_STATUS register may indicate the information for the most severe error recorded on the processor. System software is required to use the RIPV flag indication in the IA32_MCG_STATUS register to make a final decision of recoverability of the errors and find the restart-ability requirement after examining each IA32_MCi_STATUS register error information in the MC banks.

In certain cases where system software observes more than one SRAR error logged for a single logical processor, it can no longer rely on affected threads as specified in Table 15-20 above. System software is recommended to reset the system if this condition is observed.

17.10.5 Machine-Check Error Codes Interpretation

Chapter 18, “Interpreting Machine Check Error Codes,” provides information on interpreting the MCA error code, model-specific error code, and other information error code fields. For P6 family processors, information has been included on decoding external bus errors. For Pentium 4 and Intel Xeon processors; information is included on external bus, internal timer and cache hierarchy errors.

17.11 GUIDELINES FOR WRITING MACHINE-CHECK SOFTWARE

The machine-check architecture and error logging can be used in three different ways:

- To detect machine errors during normal instruction execution, using the machine-check exception (#MC).
- To periodically check and log machine errors.
- To examine recoverable UCR errors, determine software recoverability and perform recovery actions via a machine-check exception handler or a corrected machine-check interrupt handler.

To use the machine-check exception, the operating system or executive software must provide a machine-check exception handler. This handler may need to be designed specifically for each family of processors.

A special program or utility is required to log machine errors.

Guidelines for writing a machine-check exception handler or a machine-error logging utility are given in the following sections.

17.11.1 Machine-Check Exception Handler

The machine-check exception (#MC) corresponds to vector 18. To service machine-check exceptions, a trap gate must be added to the IDT. The pointer in the trap gate must point to a machine-check exception handler. Two approaches can be taken to designing the exception handler:

1. The handler can merely log all the machine status and error information, then call a debugger or shut down the system.
2. The handler can analyze the reported error information and, in some cases, attempt to correct the error and restart the processor.

For Pentium 4, Intel Xeon, Intel Atom, P6 family, and Pentium processors; virtually all machine-check conditions cannot be corrected (they result in abort-type exceptions). The logging of status and error information is therefore a baseline implementation requirement.

When IA32_MCG_CAP[24] is clear, consider the following when writing a machine-check exception handler:

- To determine the nature of the error, the handler must read each of the error-reporting register banks. The count field in the IA32_MCG_CAP register gives number of register banks. The first register of register bank 0 is at address 400H.
- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and do not need to be checked.
- To write a portable exception handler, only the MCA error code field in the IA32_MCi_STATUS register should be checked. See Section 17.10, “Interpreting the MCA Error Codes,” for information that can be used to write an algorithm to interpret this field.
- Correctable errors are corrected automatically by the processor. The UC flag in each IA32_MCi_STATUS register indicates whether the processor automatically corrected an error.
- The RIPV, PCC, and OVER flags in each IA32_MCi_STATUS register indicate whether recovery from the error is possible. If PCC or OVER are set, recovery is not possible. If RIPV is not set, program execution can not be restarted reliably. When recovery is not possible, the handler typically records the error information and signals an abort to the operating system.
- The RIPV flag in the IA32_MCG_STATUS register indicates whether the program can be restarted at the instruction indicated by the instruction pointer (the address of the instruction pushed on the stack when the exception was generated). If this flag is clear, the processor may still be able to be restarted (for debugging purposes) but not without loss of program continuity. *Note that, even when the RIPV flag is set, software may still decide to abort processor execution, e.g., because it does not expect certain kind of errors when running in supervisor mode.*
- The SEAM_NR flag in the IA32_MCG_STATUS register indicates that the reported error has been observed by the logical processor while running in SEAM non-root operation, and thus the address pushed on the stack is not that of the instruction on which the error was originally reported. *Note that, when the SEAM_NR flag is set, software may decide to restart processor execution, even though the RIPV flag is set, as the address pushed on the stack is that of an instruction for which the reported error is not expected to recur.*
- For unrecoverable errors, the EIPV flag in the IA32_MCG_STATUS register indicates whether the instruction indicated by the instruction pointer pushed on the stack (when the exception was generated) is related to the error. If the flag is clear, the pushed instruction may not be related to the error.
- The MCIP flag in the IA32_MCG_STATUS register indicates whether a machine-check exception was generated. Before returning from the machine-check exception handler, software should clear this flag so that it can be used reliably by an error logging utility. The MCIP flag also detects recursion. The machine-check architecture does not support recursion. When the processor detects machine-check recursion, it enters the shutdown state.

Example 17-2 gives typical steps carried out by a machine-check exception handler.

Example 17-2. Machine-Check Exception Handler Pseudocode

```

IF CPU supports MCE
  THEN
    IF CPU supports MCA
      THEN
        call errorlogging routine; (* returns restartability *)
      FI;
    ELSE (* Pentium(R) processor compatible *)
      READ P5_MC_ADDR
      READ P5_MC_TYPE;
      report RESTARTABILITY to console;
    FI;
  IF error is not restartable
    THEN
      report RESTARTABILITY to console;
  
```

```

        abort system;
    FI;
    CLEAR MCIP flag in IA32_MCG_STATUS;

```

17.11.2 Pentium Processor Machine-Check Exception Handling

Machine-check exception handler on P6 family, Intel Atom and later processor families, should follow the guidelines described in Section 17.11.1 and Example 17-2 that check the processor's support of MCA.

NOTE

On processors that support MCA (CPUID.01H:EDX.MCA = 1) reading the P5_MC_TYPE and P5_MC_ADDR registers may produce invalid data.

When machine-check exceptions are enabled for the Pentium processor (MCE flag is set in control register CR4), the machine-check exception handler uses the RDMSR instruction to read the error type from the P5_MC_TYPE register and the machine check address from the P5_MC_ADDR register. The handler then normally reports these register values to the system console before aborting execution (see Example 17-2).

17.11.3 Logging Correctable Machine-Check Errors

The error handling routine for servicing the machine-check exceptions is responsible for logging uncorrected errors.

If a machine-check error is correctable, the processor does not generate a machine-check exception for it. To detect correctable machine-check errors, a utility program must be written that reads each of the machine-check error-reporting register banks and logs the results in an accounting file or data structure. This utility can be implemented in either of the following ways.

- A system daemon that polls the register banks on an infrequent basis, such as hourly or daily.
- A user-initiated application that polls the register banks and records the exceptions. Here, the actual polling service is provided by an operating-system driver or through the system call interface.
- An interrupt service routine servicing CMCI can read the MC banks and log the error. Please refer to Section 17.11.4.2 for guidelines on logging correctable machine checks.

Example 17-3 gives pseudocode for an error logging utility.

Example 17-3. Machine-Check Error Logging Pseudocode

```

Assume that execution is restartable;
IF the processor supports MCA
    THEN
    FOR each bank of machine-check registers
        DO
            READ IA32_MCI_STATUS;
            IF VAL flag in IA32_MCI_STATUS = 1
                THEN
                    IF ADDR flag in IA32_MCI_STATUS = 1
                        THEN READ IA32_MCI_ADDR;
                    FI;
                    IF MISC flag in IA32_MCI_STATUS = 1
                        THEN READ IA32_MCI_MISC;
                    FI;
                    IF MCIP flag in IA32_MCG_STATUS = 1
                        (* Machine-check exception is in progress *)
                        AND PCC flag in IA32_MCI_STATUS = 1
                        OR RIPV flag in IA32_MCG_STATUS = 0
                        (* execution is not restartable *)
                        THEN
                            RESTARTABILITY = FALSE;
                            return RESTARTABILITY to calling procedure;

```

```

    FI;
    Save time-stamp counter and processor ID;
    Set IA32_MCi_STATUS to all 0s;
    Execute serializing instruction (i.e., CPUID);
    FI;
    OD;
FI;

```

If the processor supports the machine-check architecture, the utility reads through the banks of error-reporting registers looking for valid register entries. It then saves the values of the IA32_MCi_STATUS, IA32_MCi_ADDR, IA32_MCi_MISC, and IA32_MCG_STATUS registers for each bank that is valid. The routine minimizes processing time by recording the raw data into a system data structure or file, reducing the overhead associated with polling. User utilities analyze the collected data in an off-line environment.

When the MCIP flag is set in the IA32_MCG_STATUS register, a machine-check exception is in progress and the machine-check exception handler has called the exception logging routine.

Once the logging process has been completed the exception-handling routine must determine whether execution can be restarted, which is usually possible when damage has not occurred (The PCC flag is clear, in the IA32_MCi_STATUS register) and when the processor can guarantee that execution is restartable (the RIPV flag is set in the IA32_MCG_STATUS register). If execution cannot be restarted, the system is not recoverable and the exception-handling routine should signal the console appropriately before returning the error status to the Operating System kernel for subsequent shutdown.

The machine-check architecture allows buffering of exceptions from a given error-reporting bank although the Pentium 4, Intel Xeon, Intel Atom, and P6 family processors do not implement this feature. The error logging routine should provide compatibility with future processors by reading each hardware error-reporting bank's IA32_MCi_STATUS register and then writing 0s to clear the OVER and VAL flags in this register. The error logging utility should re-read the IA32_MCi_STATUS register for the bank ensuring that the valid bit is clear. The processor will write the next error into the register bank and set the VAL flags.

Additional information that should be stored by the exception-logging routine includes the processor's time-stamp counter value, which provides a mechanism to indicate the frequency of exceptions. A multiprocessing operating system stores the identity of the processor node incurring the exception using a unique identifier, such as the processor's APIC ID (see Section 12.8, "Handling Interrupts").

The basic algorithm given in Example 17-3 can be modified to provide more robust recovery techniques. For example, software has the flexibility to attempt recovery using information unavailable to the hardware. Specifically, the machine-check exception handler can, after logging carefully analyze the error-reporting registers when the error-logging routine reports an error that does not allow execution to be restarted. These recovery techniques can use external bus related model-specific information provided with the error report to localize the source of the error within the system and determine the appropriate recovery strategy.

17.11.4 Machine-Check Software Handler Guidelines for Error Recovery

17.11.4.1 Machine-Check Exception Handler for Error Recovery

When writing a machine-check exception (MCE) handler to support software recovery from Uncorrected Recoverable (UCR) errors, consider the following:

- When IA32_MCG_CAP [24] is zero, there are no recoverable errors supported and all machine-check are fatal exceptions. The logging of status and error information is therefore a baseline implementation requirement.
- When IA32_MCG_CAP [24] is 1, certain uncorrected errors called uncorrected recoverable (UCR) errors may be software recoverable. The handler can analyze the reported error information, and in some cases attempt to recover from the uncorrected error and continue execution.
- For processors on which CPUID reports DisplayFamily_DisplayModel as 06H_0EH and onward, an MCA signal is broadcast to all logical processors in the system; see the CPUID instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Due to the potentially shared machine check MSR resources among the logical processors on the same package/core, the MCE handler may be required to synchronize with the other processors that received a machine check error and

serialize access to the machine check registers when analyzing, logging, and clearing the information in the machine check registers.

- On processors that indicate ability for local machine-check exception (MCG_LMCE_P), hardware can choose to report the error to only a single logical processor if system software has enabled LMCE by setting `IA32_MCG_EXT_CTL[LMCE_EN] = 1` as outlined in Section 17.3.1.5.
- The VAL (valid) flag in each `IA32_MCi_STATUS` register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank do not contain valid error information and should not be checked.
- The MCE handler is primarily responsible for processing uncorrected errors. The UC flag in each `IA32_MCi_Status` register indicates whether the reported error was corrected (UC=0) or uncorrected (UC=1). The MCE handler can optionally log and clear the corrected errors in the MC banks if it can implement software algorithm to avoid the undesired race conditions with the CMCI or CMC polling handler.
- For uncorrectable errors, the EIPV flag in the `IA32_MCG_STATUS` register indicates (when set) that the instruction pointed to by the instruction pointer pushed onto the stack when the machine-check exception is generated is directly associated with the error. When this flag is cleared, the instruction pointed to may not be associated with the error.
- The MCIP flag in the `IA32_MCG_STATUS` register indicates whether a machine-check exception was generated. When a machine check exception is generated, it is expected that the MCIP flag in the `IA32_MCG_STATUS` register is set to 1. If it is not set, this machine check was generated by either an INT 18 instruction or some piece of hardware signaling an interrupt with vector 18.

When `IA32_MCG_CAP [24]` is 1, the following rules can apply when writing a machine check exception (MCE) handler to support software recovery:

- The PCC flag in each `IA32_MCi_STATUS` register indicates whether recovery from the error is possible for uncorrected errors (UC=1). If the PCC flag is set for enabled uncorrected errors (UC=1 and EN=1), recovery is not possible. When recovery is not possible, the MCE handler typically records the error information and signals the operating system to reset the system.
- The RIPV flag in the `IA32_MCG_STATUS` register indicates whether restarting the program execution from the instruction pointer saved on the stack for the machine check exception is possible. When the RIPV is set, program execution can be restarted reliably when recovery is possible. If the RIPV flag is not set, program execution cannot be restarted reliably. In this case the recovery algorithm may involve terminating the current program execution and resuming an alternate thread of execution upon return from the machine check handler when recovery is possible. When recovery is not possible, the MCE handler signals the operating system to reset the system.
- When the EN flag is zero but the VAL and UC flags are one in the `IA32_MCi_STATUS` register, the reported uncorrected error in this bank is not enabled. As uncorrected errors with the EN flag = 0 are not the source of machine check exceptions, the MCE handler should log and clear non-enabled errors when the S bit is set and should continue searching for enabled errors from the other `IA32_MCi_STATUS` registers. Note that when `IA32_MCG_CAP [24]` is 0, any uncorrected error condition (VAL =1 and UC=1) including the one with the EN flag cleared are fatal and the handler must signal the operating system to reset the system. For the errors that do not generate machine check exceptions, the EN flag has no meaning.
- When the VAL flag is one, the UC flag is one, the EN flag is one and the PCC flag is zero in the `IA32_MCi_STATUS` register, the error in this bank is an uncorrected recoverable (UCR) error. The MCE handler needs to examine the S flag and the AR flag to find the type of the UCR error for software recovery and determine if software error recovery is possible.
- When both the S and the AR flags are clear in the `IA32_MCi_STATUS` register for the UCR error (VAL=1, UC=1, EN=x and PCC=0), the error in this bank is an uncorrected no-action required error (UCNA). UCNA errors are uncorrected but do not require any OS recovery action to continue execution. These errors indicate that some data in the system is corrupt, but that data has not been consumed and may not be consumed. If that data is consumed a non-UCNA machine check exception will be generated. UCNA errors are signaled in the same way as corrected machine check errors and the CMCI and CMC polling handler is primarily responsible for handling UCNA errors. Like corrected errors, the MCA handler can optionally log and clear UCNA errors as long as it can avoid the undesired race condition with the CMCI or CMC polling handler. As UCNA errors are not the source of machine check exceptions, the MCA handler should continue searching for uncorrected or software recoverable errors in all other MC banks.

- When the S flag in the IA32_MCi_STATUS register is set for the UCR error ((VAL=1, UC=1, EN=1 and PCC=0), the error in this bank is software recoverable and it was signaled through a machine-check exception. The AR flag in the IA32_MCi_STATUS register further clarifies the type of the software recoverable errors.
- When the AR flag in the IA32_MCi_STATUS register is clear for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action optional (SRAO) error. The MCE handler and the operating system can analyze the IA32_MCi_STATUS [15:0] to implement MCA error code specific optional recovery action, but this recovery action is optional. System software can resume the program execution from the instruction pointer saved on the stack for the machine check exception when the RIPV flag in the IA32_MCG_STATUS register is set.
- Even if the OVER flag in the IA32_MCi_STATUS register is set for the SRAO error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=0), the MCE handler can take recovery action for the SRAO error logged in the IA32_MCi_STATUS register. Since the recovery action for SRAO errors is optional, restarting the program execution from the instruction pointer saved on the stack for the machine check exception is still possible for the overflowed SRAO error if the RIPV flag in the IA32_MCG_STATUS is set.
- When the AR flag in the IA32_MCi_STATUS register is set for the software recoverable error (VAL=1, UC=1, EN=1, PCC=0 and S=1), the error in this bank is a software recoverable action required (SRAR) error. The MCE handler and the operating system must take recovery action in order to continue execution after the machine-check exception. The MCA handler and the operating system need to analyze the IA32_MCi_STATUS [15:0] to determine the MCA error code specific recovery action. If no recovery action can be performed, the operating system must reset the system.
- When the OVER flag in the IA32_MCi_STATUS register is set for the SRAR error (VAL=1, UC=1, EN=1, PCC=0, S=1 and AR=1), the MCE handler cannot take recovery action as the information of the SRAR error in the IA32_MCi_STATUS register was potentially lost due to the overflow condition. Since the recovery action for SRAR errors must be taken, the MCE handler must signal the operating system to reset the system.
- When the MCE handler cannot find any uncorrected (VAL=1, UC=1 and EN=1) or any software recoverable errors (VAL=1, UC=1, EN=1, PCC=0 and S=1) in any of the IA32_MCi banks of the processors, this is an unexpected condition for the MCE handler and the handler should signal the operating system to reset the system.
- Before returning from the machine-check exception handler, software must clear the MCIP flag in the IA32_MCG_STATUS register. The MCIP flag is used to detect recursion. The machine-check architecture does not support recursion. When the processor receives a machine check when MCIP is set, it automatically enters the shutdown state.

Example 17-4 gives pseudocode for an MC exception handler that supports recovery of UCR.

Example 17-4. Machine-Check Error Handler Pseudocode Supporting UCR

```

MACHINE CHECK HANDLER: (* Called from INT 18 handler *)
NOERROR = TRUE;
ProcessorCount = 0;
IF CPU supports MCA
  THEN
    RESTARTABILITY = TRUE;
    IF (Processor Family = 6 AND DisplayModel ≥ 0EH) OR (Processor Family > 6)
      THEN
        IF (MCG_LMCE = 1)
          MCA_BROADCAST = FALSE;
        ELSE
          MCA_BROADCAST = TRUE;
        FI;
        Acquire SpinLock;
        ProcessorCount++; (* Allowing one logical processor at a time to examine machine check registers *)
        CALL MCA ERROR PROCESSING; (* returns RESTARTABILITY and NOERROR *)
      ELSE
        MCA_BROADCAST = FALSE;
        (* Implement a rendezvous mechanism with the other processors if necessary *)
        CALL MCA ERROR PROCESSING;
    FI;
  ELSE (* Pentium(R) processor compatible *)

```

MACHINE-CHECK ARCHITECTURE

```
    READ P5_MC_ADDR
    READ P5_MC_TYPE;
    RESTARTABILITY = FALSE;
FI;

IF NOERROR = TRUE
    THEN
        IF NOT (MCG_RIPV = 1 AND MCG_EIPV = 0)
            THEN
                RESTARTABILITY = FALSE;
            FI
        FI;
FI;

IF RESTARTABILITY = FALSE
    THEN
        Report RESTARTABILITY to console;
        Reset system;
FI;

IF MCA_BROADCAST = TRUE
    THEN
        IF ProcessorCount = MAX_PROCESSORS
            AND NOERROR = TRUE
            THEN
                Report RESTARTABILITY to console;
                Reset system;
            FI;
        Release SpinLock;
        Wait till ProcessorCount = MAX_PROCESSORS on system;
        (* implement a timeout and abort function if necessary *)
FI;
CLEAR IA32_MCG_STATUS;
RESUME Execution;
(* End of MACHINE CHECK HANDLER*)

MCA ERROR PROCESSING: (* MCA Error Processing Routine called from MCA Handler *)
IF MCIP flag in IA32_MCG_STATUS = 0
    THEN (* MCIP=0 upon MCA is unexpected *)
        RESTARTABILITY = FALSE;
FI;

FOR each bank of machine-check registers
    DO
        CLEAR_MC_BANK = FALSE;
        READ IA32_MCI_STATUS;
        IF VAL Flag in IA32_MCI_STATUS = 1
            THEN
                IF UC Flag in IA32_MCI_STATUS = 1
                    THEN
                        IF Bit 24 in IA32_MCG_CAP = 0
                            THEN (* the processor does not support software error recovery *)
                                RESTARTABILITY = FALSE;
                                NOERROR = FALSE;
                                GOTO LOG MCA REGISTER;
                            FI;
                        (* the processor supports software error recovery *)
                        IF EN Flag in IA32_MCI_STATUS = 0 AND OVER Flag in IA32_MCI_STATUS=0
                            THEN (* It is a spurious MCA Log. Log and clear the register *)
                                CLEAR_MC_BANK = TRUE;
                                GOTO LOG MCA REGISTER;
                            FI;
                        IF PCC = 1 and EN = 1 in IA32_MCI_STATUS
                            THEN (* processor context might have been corrupted *)
                                RESTARTABILITY = FALSE;
                            ELSE (* It is an uncorrected recoverable (UCR) error *)
                                IF S Flag in IA32_MCI_STATUS = 0
```

```

        THEN
            IF AR Flag in IA32_MCi_STATUS = 0
                THEN (* It is a uncorrected no action required (UCNA) error *)
                    GOTO CONTINUE; (* let CMCi and CMC polling handler to process *)
                ELSE
                    RESTARTABILITY = FALSE; (* S=0, AR=1 is illegal *)
            FI
        FI;
    IF RESTARTABILITY = FALSE
        THEN (* no need to take recovery action if RESTARTABILITY is already false *)
            NOERROR = FALSE;
            GOTO LOG MCA REGISTER;
        FI;
    (* S in IA32_MCi_STATUS = 1 *)
    IF AR Flag in IA32_MCi_STATUS = 1
        THEN (* It is a software recoverable and action required (SRAR) error *)
            IF OVER Flag in IA32_MCi_STATUS = 1
                THEN
                    RESTARTABILITY = FALSE;
                    NOERROR = FALSE;
                    GOTO LOG MCA REGISTER;
                FI
            IF MCACOD Value in IA32_MCi_STATUS is recognized
                AND Current Processor is an Affected Processor
                THEN
                    Implement MCACOD specific recovery action;
                    CLEAR_MC_BANK = TRUE;
                ELSE
                    RESTARTABILITY = FALSE;
                FI;
            ELSE (* It is a software recoverable and action optional (SRAO) error *)
                IF OVER Flag in IA32_MCi_STATUS = 0 AND
                    MCACOD in IA32_MCi_STATUS is recognized
                THEN
                    Implement MCACOD specific recovery action;
                FI;
                CLEAR_MC_BANK = TRUE;
            FI; AR
        FI; PCC
        NOERROR = FALSE;
        GOTO LOG MCA REGISTER;
    ELSE (* It is a corrected error; continue to the next IA32_MCi_STATUS *)
        GOTO CONTINUE;
    FI; UC
FI; VAL
LOG MCA REGISTER:
    SAVE IA32_MCi_STATUS;
    IF MISCV in IA32_MCi_STATUS
        THEN
            SAVE IA32_MCi_MISC;
        FI;
    IF ADDRv in IA32_MCi_STATUS
        THEN
            SAVE IA32_MCi_ADDR;
        FI;
    IF CLEAR_MC_BANK = TRUE
        THEN
            SET all 0 to IA32_MCi_STATUS;
            IF MISCV in IA32_MCi_STATUS
                THEN
                    SET all 0 to IA32_MCi_MISC;
            FI;
            IF ADDRv in IA32_MCi_STATUS
                THEN
                    SET all 0 to IA32_MCi_ADDR;

```



```

        FI;
    FI;
    CONTINUE;
OD;
(*END FOR *)
RETURN;
(* End of MCA ERROR PROCESSING*)

```

17.11.4.2 Corrected Machine-Check Handler for Error Recovery

When writing a corrected machine check handler, which is invoked as a result of CMCI or called from an OS CMC Polling dispatcher, consider the following:

- The VAL (valid) flag in each IA32_MCi_STATUS register indicates whether the error information in the register is valid. If this flag is clear, the registers in that bank does not contain valid error information and does not need to be checked.
- The CMCI or CMC polling handler is responsible for logging and clearing corrected errors. The UC flag in each IA32_MCi_Status register indicates whether the reported error was corrected (UC=0) or not (UC=1).
- When IA32_MCG_CAP [24] is one, the CMC handler is also responsible for logging and clearing uncorrected no-action required (UCNA) errors. When the UC flag is one but the PCC, S, and AR flags are zero in the IA32_MCi_STATUS register, the reported error in this bank is an uncorrected no-action required (UCNA) error. In cases when SRAO error are signaled as UCNA error via CMCI, software can perform recovery for those errors identified in Table 17-16.
- In addition to corrected errors and UCNA errors, the CMC handler optionally logs uncorrected (UC=1 and PCC=1), software recoverable machine check errors (UC=1, PCC=0 and S=1), but should avoid clearing those errors from the MC banks. Clearing these errors may result in accidentally removing these errors before these errors are actually handled and processed by the MCE handler for attempted software error recovery.

Example 17-5 gives pseudocode for a CMCI handler with UCR support.

Example 17-5. Corrected Error Handler Pseudocode with UCR Support

Corrected Error HANDLER: (* Called from CMCI handler or OS CMC Polling Dispatcher*)

```

IF CPU supports MCA
    THEN
        FOR each bank of machine-check registers
            DO
                READ IA32_MCi_STATUS;
                IF VAL flag in IA32_MCi_STATUS = 1
                    THEN
                        IF UC Flag in IA32_MCi_STATUS = 0 (* It is a corrected error *)
                            THEN
                                GOTO LOG CMC ERROR;
                            ELSE
                                IF Bit 24 in IA32_MCG_CAP = 0
                                    THEN
                                        GOTO CONTINUE;
                                FI;
                                IF S Flag in IA32_MCi_STATUS = 0 AND AR Flag in IA32_MCi_STATUS = 0
                                    THEN (* It is a uncorrected no action required error *)
                                        GOTO LOG CMC ERROR
                                FI
                                IF EN Flag in IA32_MCi_STATUS = 0
                                    THEN (* It is a spurious MCA error *)
                                        GOTO LOG CMC ERROR
                                FI;
                            FI;
                        FI;
                    GOTO CONTINUE;
                LOG CMC ERROR:
                SAVE IA32_MCi_STATUS;
                If MISCV Flag in IA32_MCi_STATUS

```



```
        THEN
            SAVE IA32_MCI_MISC;
            SET all 0 to IA32_MCI_MISC;
        FI;
    IF ADDR_V Flag in IA32_MCI_STATUS
        THEN
            SAVE IA32_MCI_ADDR;
            SET all 0 to IA32_MCI_ADDR;
        FI;
    SET all 0 to IA32_MCI_STATUS;
    CONTINUE;
OD;
( *END FOR *)
FI;
```


17. Updates to Chapter 19, Volume 3B

Change bars and violet text show changes to Chapter 19 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- In Section 19.19.7.1, "Memory Bandwidth Allocation Enumeration", added CPUID.10H.ResID=3:ECX[0] and updated Figure 19-38.
- Added Section 19.21, "Core Telemetry for Intel® Platform Monitoring Technology (PMT)".

CHAPTER 19

DEBUG, BRANCH PROFILE, TSC, AND INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FEATURES

NOTE

This chapter makes numerous references to last-branch recording (LBR) facilities. Unless noted otherwise, all such references in this chapter are to an earlier non-architectural form of the feature. Chapter 20 defines an architectural form of last-branch recording that is supported on newer processors.

Intel 64 and IA-32 architectures provide debug facilities for use in debugging code and monitoring performance. These facilities are valuable for debugging application software, system software, and multitasking operating systems. Debug support is accessed using debug registers (DR0 through DR7) and model-specific registers (MSRs):

- Debug registers hold the addresses of memory and I/O locations called breakpoints. Breakpoints are user-selected locations in a program, a data-storage area in memory, or specific I/O ports. They are set where a programmer or system designer wishes to halt execution of a program and examine the state of the processor by invoking debugger software. A debug exception (#DB) is generated when a memory or I/O access is made to a breakpoint address.
- MSRs monitor branches, interrupts, and exceptions; they record addresses of the last branch, interrupt or exception taken and the last branch taken before an interrupt or exception.
- Time stamp counter is described in Section 19.17, "Time-Stamp Counter."
- Features that allow monitoring of shared platform resources such as the L3 cache are described in Section 19.18, "Intel® Resource Director Technology (Intel® RDT) Monitoring Features."
- Features that enable control over shared platform resources are described in Section 19.19, "Intel® Resource Director Technology (Intel® RDT) Allocation Features."
- Features that enable control over shared platform resources for non-CPU agents are described in Section 19.20, "Intel® Resource Director Technology (Intel® RDT) for Non-CPU Agents."¹

19.1 OVERVIEW OF DEBUG SUPPORT FACILITIES

The following processor facilities support debugging and performance monitoring:

- **Debug exception (#DB)** — Transfers program control to a debug procedure or task when a debug event occurs.
- **Breakpoint exception (#BP)** — See breakpoint instruction (INT3) below.
- **Breakpoint-address registers (DR0 through DR3)** — Specifies the addresses of up to 4 breakpoints.
- **Debug status register (DR6)** — Reports the conditions that were in effect when a debug or breakpoint exception was generated.
- **Debug control register (DR7)** — Specifies the forms of memory or I/O access that cause breakpoints to be generated.
- **T (trap) flag, TSS** — Generates a debug exception (#DB) when an attempt is made to switch to a task with the T flag set in its TSS.
- **RF (resume) flag, EFLAGS register** — Suppresses multiple exceptions to the same instruction.
- **TF (trap) flag, EFLAGS register** — Generates a debug exception (#DB) after every execution of an instruction.

1. Additional information about Intel® RDT can be found in the document titled "Intel® Resource Director Technology Architecture Specification," available here: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.

- **Breakpoint instruction (INT3)** — Generates a breakpoint exception (#BP) that transfers program control to the debugger procedure or task. This instruction is an alternative way to set instruction breakpoints. It is especially useful when more than four breakpoints are desired, or when breakpoints are being placed in the source code.
- **Last branch recording facilities** — Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consist of a branch-from and a branch-to instruction address. Send branch records out on the system bus as branch trace messages (BTMs).

These facilities allow a debugger to be called as a separate task or as a procedure in the context of the current program or task. The following conditions can be used to invoke the debugger:

- Task switch to a specific task.
- Execution of the breakpoint instruction.
- Execution of any instruction.
- Execution of an instruction at a specified address.
- Read or write to a specified memory address/range.
- Write to a specified memory address/range.
- Input from a specified I/O address/range.
- Output to a specified I/O address/range.
- Attempt to change the contents of a debug register.

19.2 DEBUG REGISTERS

Eight debug registers (see Figure 19-1 for 32-bit operation and Figure 19-2 for 64-bit operation) control the debug operation of the processor. These registers can be written to and read using the move to/from debug register form of the MOV instruction. A debug register may be the source or destination operand for one of these instructions.

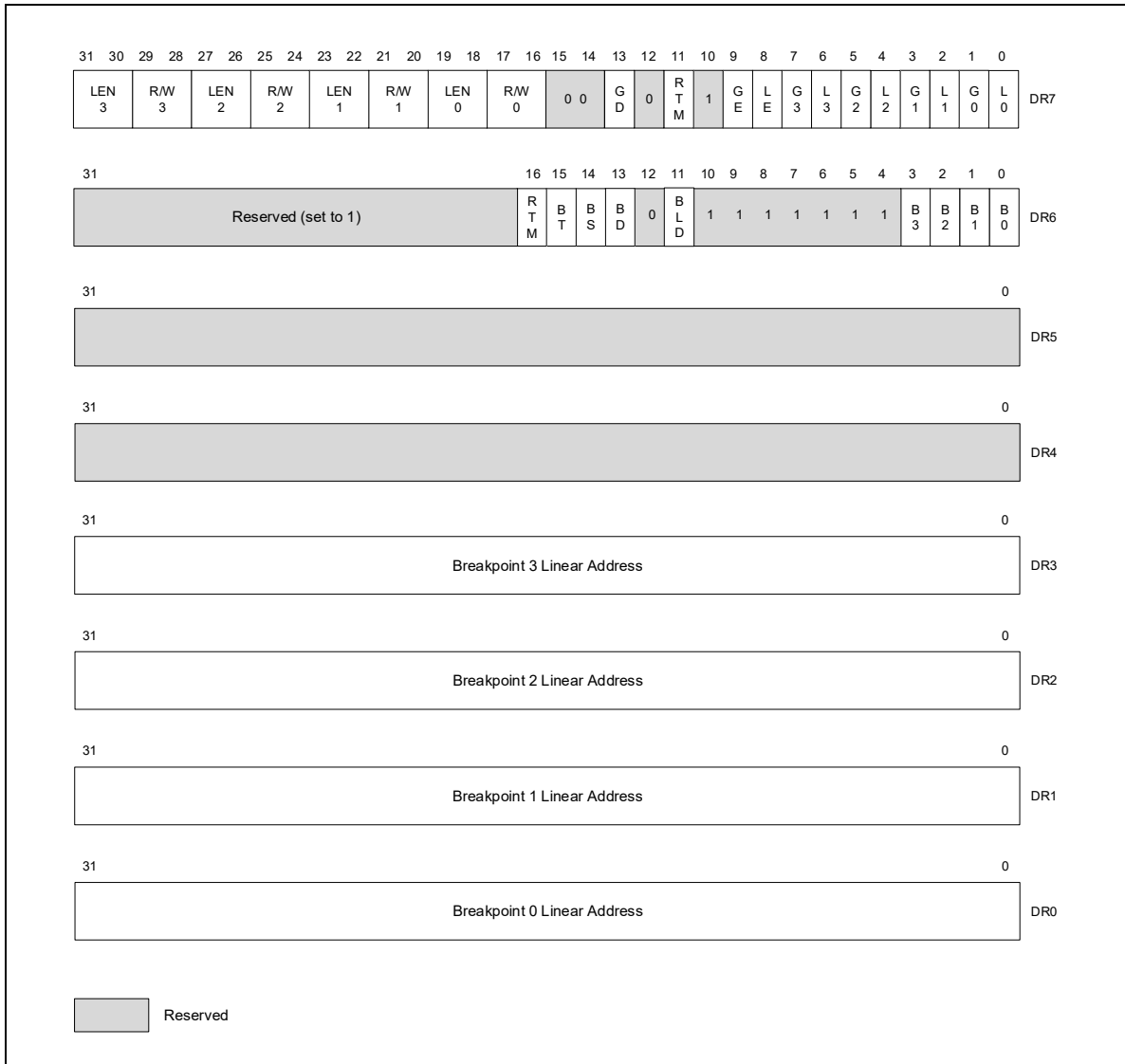


Figure 19-1. Debug Registers

Debug registers are privileged resources; a MOV instruction that accesses these registers can only be executed in real-address mode, in SMM or in protected mode at a CPL of 0. An attempt to read or write the debug registers from any other privilege level generates a general-protection exception (#GP).

The primary function of the debug registers is to set up and monitor from 1 to 4 breakpoints, numbered 0 through 3. For each breakpoint, the following information can be specified:

- The linear address where the breakpoint is to occur.
- The length of the breakpoint location: 1, 2, 4, or 8 bytes (refer to the notes in Section 19.2.4).
- The operation that must be performed at the address for a debug exception to be generated.
- Whether the breakpoint is enabled.
- Whether the breakpoint condition was present when the debug exception was generated.

The following paragraphs describe the functions of flags and fields in the debug registers.

19.2.1 Debug Address Registers (DR0-DR3)

Each of the debug-address registers (DR0 through DR3) holds the 32-bit linear address of a breakpoint (see Figure 19-1). Breakpoint comparisons are made before physical address translation occurs. The contents of debug register DR7 further specifies breakpoint conditions.

19.2.2 Debug Registers DR4 and DR5

Debug registers DR4 and DR5 are reserved when debug extensions are enabled (when the DE flag in control register CR4 is set) and attempts to reference the DR4 and DR5 registers cause invalid-opcode exceptions (#UD). When debug extensions are not enabled (when the DE flag is clear), these registers are aliased to debug registers DR6 and DR7.

19.2.3 Debug Status Register (DR6)

The debug status register (DR6) reports debug conditions that were sampled at the time the last debug exception was generated (see Figure 19-1). Updates to this register only occur when an exception is generated. The flags in this register show the following information:

- **B0 through B3 (breakpoint condition detected) flags (bits 0 through 3)** — Indicates (when set) that its associated breakpoint condition was met when a debug exception was generated. These flags are set if the condition described for each breakpoint by the LEN_n and R/W_n flags in debug control register DR7 is true. They may or may not be set if the breakpoint is not enabled by the Ln or the Gn flags in register DR7. Therefore on a #DB, a debug handler should check only those B0-B3 bits which correspond to an enabled breakpoint.
- **BLD (bus-lock detected) flag (bit 11)** — Indicates (when **clear**) that the debug exception was triggered by the assertion of a bus lock when $CPL > 0$ and OS bus-lock detection was enabled (see Section 19.3.1.6). Other debug exceptions do not modify this bit. To avoid confusion in identifying debug exceptions, software debug-exception handlers should set bit 11 to 1 before returning. (Software that never enables OS bus-lock detection need not do this as $DR6[11] = 1$ following reset.) This bit is always 1 if the processor does not support OS bus-lock detection.
- **BD (debug register access detected) flag (bit 13)** — Indicates that the next instruction in the instruction stream accesses one of the debug registers (DR0 through DR7). This flag is enabled when the GD (general detect) flag in debug control register DR7 is set. See Section 19.2.4, “Debug Control Register (DR7),” for further explanation of the purpose of this flag.
- **BS (single step) flag (bit 14)** — Indicates (when set) that the debug exception was triggered by the single-step execution mode (enabled with the TF flag in the EFLAGS register). The single-step mode is the highest-priority debug exception. When the BS flag is set, any of the other debug status bits also may be set.
- **BT (task switch) flag (bit 15)** — Indicates (when set) that the debug exception resulted from a task switch where the T flag (debug trap flag) in the TSS of the target task was set. See Section 9.2.1, “Task-State Segment (TSS),” for the format of a TSS. There is no flag in debug control register DR7 to enable or disable this exception; the T flag of the TSS is the only enabling flag.
- **RTM (restricted transactional memory) flag (bit 16)** — Indicates (when **clear**) that a debug exception (#DB) or breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 19.3.3). This bit is set for any other debug exception (including all those that occur when advanced debugging of RTM transactional regions is not enabled). This bit is always 1 if the processor does not support RTM.

Certain debug exceptions may clear bits 0-3. The remaining contents of the DR6 register are never cleared by the processor. To avoid confusion in identifying debug exceptions, debug handlers should clear the register (except bit 16, which they should set) before returning to the interrupted task.

19.2.4 Debug Control Register (DR7)

The debug control register (DR7) enables or disables breakpoints and sets breakpoint conditions (see Figure 19-1). The flags and fields in this register control the following things:

- **L0 through L3 (local breakpoint enable) flags (bits 0, 2, 4, and 6)** — Enables (when set) the breakpoint condition for the associated breakpoint for the current task. When a breakpoint condition is detected and its associated *L_n* flag is set, a debug exception is generated. The processor automatically clears these flags on every task switch to avoid unwanted breakpoint conditions in the new task.
- **G0 through G3 (global breakpoint enable) flags (bits 1, 3, 5, and 7)** — Enables (when set) the breakpoint condition for the associated breakpoint for all tasks. When a breakpoint condition is detected and its associated *G_n* flag is set, a debug exception is generated. The processor does not clear these flags on a task switch, allowing a breakpoint to be enabled for all tasks.
- **LE and GE (local and global exact breakpoint enable) flags (bits 8, 9)** — This feature is not supported in the P6 family processors, later IA-32 processors, and Intel 64 processors. When set, these flags cause the processor to detect the exact instruction that caused a data breakpoint condition. For backward and forward compatibility with other Intel processors, we recommend that the LE and GE flags be set to 1 if exact breakpoints are required.
- **RTM (restricted transactional memory) flag (bit 11)** — Enables (when set) advanced debugging of RTM transactional regions (see Section 19.3.3). This advanced debugging is enabled only if IA32_DEBUGCTL.RTM is also set.
- **GD (general detect enable) flag (bit 13)** — Enables (when set) debug-register protection, which causes a debug exception to be generated prior to any MOV instruction that accesses a debug register. When such a condition is detected, the BD flag in debug status register DR6 is set prior to generating the exception. This condition is provided to support in-circuit emulators.

When the emulator needs to access the debug registers, emulator software can set the GD flag to prevent interference from the program currently executing on the processor.

The processor clears the GD flag upon entering to the debug exception handler, to allow the handler access to the debug registers.

- **R/W0 through R/W3 (read/write) fields (bits 16, 17, 20, 21, 24, 25, 28, and 29)** — Specifies the breakpoint condition for the corresponding breakpoint. The DE (debug extensions) flag in control register CR4 determines how the bits in the R/W_{*n*} fields are interpreted. When the DE flag is set, the processor interprets bits as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Break on I/O reads or writes.
- 11 — Break on data reads or writes but not instruction fetches.

When the DE flag is clear, the processor interprets the R/W_{*n*} bits the same as for the Intel386™ and Intel486™ processors, which is as follows:

- 00 — Break on instruction execution only.
- 01 — Break on data writes only.
- 10 — Undefined.
- 11 — Break on data reads or writes but not instruction fetches.

- **LEN0 through LEN3 (Length) fields (bits 18, 19, 22, 23, 26, 27, 30, and 31)** — Specify the size of the memory location at the address specified in the corresponding breakpoint address register (DR0 through DR3). These fields are interpreted as follows:

- 00 — 1-byte length.
- 01 — 2-byte length.
- 10 — Undefined (or 8 byte length, see note below).
- 11 — 4-byte length.

If the corresponding R/W_{*n*} field in register DR7 is 00 (instruction execution), then the LEN_{*n*} field should also be 00. The effect of using other lengths is undefined. See Section 19.2.5, “Breakpoint Field Recognition,” below.

NOTES

For Pentium® 4 and Intel® Xeon® processors with a CPUID signature corresponding to family 15 (model 3, 4, and 6), break point conditions permit specifying 8-byte length on data read/write with an of encoding 10B in the LEN_n field.

Encoding 10B is also supported in processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture, the respective CPUID signatures corresponding to family 6, model 15, and family 6, DisplayModel value 23 (see the CPUID instruction in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A). The Encoding 10B is supported in processors based on Intel Atom® microarchitecture, with CPUID signature of family 6, DisplayModel value 1CH. The encoding 10B is undefined for other processors.

19.2.5 Breakpoint Field Recognition

Breakpoint address registers (debug registers DR0 through DR3) and the LEN_n fields for each breakpoint define a range of sequential byte addresses for a data or I/O breakpoint. The LEN_n fields permit specification of a 1-, 2-, 4- or 8-byte range, beginning at the linear address specified in the corresponding debug register (DR_n). Two-byte ranges must be aligned on word boundaries; 4-byte ranges must be aligned on doubleword boundaries, 8-byte ranges must be aligned on quadword boundaries. I/O addresses are zero-extended (from 16 to 32 bits, for comparison with the breakpoint address in the selected debug register). These requirements are enforced by the processor; it uses LEN_n field bits to mask the lower address bits in the debug registers. Unaligned data or I/O breakpoint addresses do not yield valid results.

A data breakpoint for reading or writing data is triggered if any of the bytes participating in an access is within the range defined by a breakpoint address register and its LEN_n field. Table 19-1 provides an example setup of debug registers and data accesses that would subsequently trap or not trap on the breakpoints.

A data breakpoint for an unaligned operand can be constructed using two breakpoints, where each breakpoint is byte-aligned and the two breakpoints together cover the operand. The breakpoints generate exceptions only for the operand, not for neighboring bytes.

Instruction breakpoint addresses must have a length specification of 1 byte (the LEN_n field is set to 00). Instruction breakpoints for other operand sizes are undefined. The processor recognizes an instruction breakpoint address only when it points to the first byte of an instruction. If the instruction has prefixes, the breakpoint address must point to the first prefix.

Table 19-1. Breakpoint Examples

Debug Register Setup			
Debug Register	R/W _n	Breakpoint Address	LEN _n
DR0	R/W0 = 11 (Read/Write)	A0001H	LEN0 = 00 (1 byte)
DR1	R/W1 = 01 (Write)	A0002H	LEN1 = 00 (1 byte)
DR2	R/W2 = 11 (Read/Write)	B0002H	LEN2 = 01) (2 bytes)
DR3	R/W3 = 01 (Write)	C0000H	LEN3 = 11 (4 bytes)
Data Accesses			
Operation		Address	Access Length (In Bytes)
Data operations that trap			
- Read or write		A0001H	1
- Read or write		A0001H	2
- Write		A0002H	1
- Write		A0002H	2
- Read or write		B0001H	4
- Read or write		B0002H	1
- Read or write		B0002H	2
- Write		C0000H	4
- Write		C0001H	2
- Write		C0003H	1

Table 19-1. Breakpoint Examples (Contd.)

Debug Register Setup			
Debug Register	R/Wn	Breakpoint Address	LENn
Data operations that do not trap			
- Read or write		A0000H	1
- Read		A0002H	1
- Read or write		A0003H	4
- Read or write		B0000H	2
- Read		C0000H	2
- Read or write		C0004H	4

19.2.6 Debug Registers and Intel® 64 Processors

For Intel 64 architecture processors, debug registers DR0–DR7 are 64 bits. In 16-bit or 32-bit modes (protected mode and compatibility mode), writes to a debug register fill the upper 32 bits with zeros. Reads from a debug register return the lower 32 bits. In 64-bit mode, MOV DRn instructions read or write all 64 bits. Operand-size prefixes are ignored.

In 64-bit mode, the upper 32 bits of DR6 and DR7 are reserved and must be written with zeros. Writing 1 to any of the upper 32 bits results in a #GP(0) exception (see Figure 19-2). All 64 bits of DR0–DR3 are writable by software. However, MOV DRn instructions do not check that addresses written to DR0–DR3 are in the linear-address limits of the processor implementation (address matching is supported only on valid addresses generated by the processor implementation). Breakpoint conditions for 8-byte memory read/writes are supported in all modes.

19.3 DEBUG EXCEPTIONS

The Intel 64 and IA-32 architectures dedicate two interrupt vectors to handling debug exceptions: vector 1 (debug exception, #DB) and vector 3 (breakpoint exception, #BP). The following sections describe how these exceptions are generated and typical exception handler operations.

19.3.1 Debug Exception (#DB)—Interrupt Vector 1

The debug-exception handler is usually a debugger program or part of a larger software system. The processor generates a debug exception for any of several conditions. The debugger checks flags in the DR6 and DR7 registers to determine which condition caused the exception and which other conditions might apply. Table 19-2 shows the states of these flags following the generation of each kind of breakpoint condition.

Instruction-breakpoint and general-detect condition (see Section 19.3.1.3, “General-Detect Exception Condition”) result in faults; other debug-exception conditions result in traps. The debug exception may report one or both at one time. The following sections describe each class of debug exception.

The INT1 instruction generates a debug exception as a trap. Hardware vendors may use the INT1 instruction for hardware debug. For that reason, Intel recommends software vendors instead use the INT3 instruction for software breakpoints.

See also: Chapter 7, “Interrupt 1—Debug Exception (#DB),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

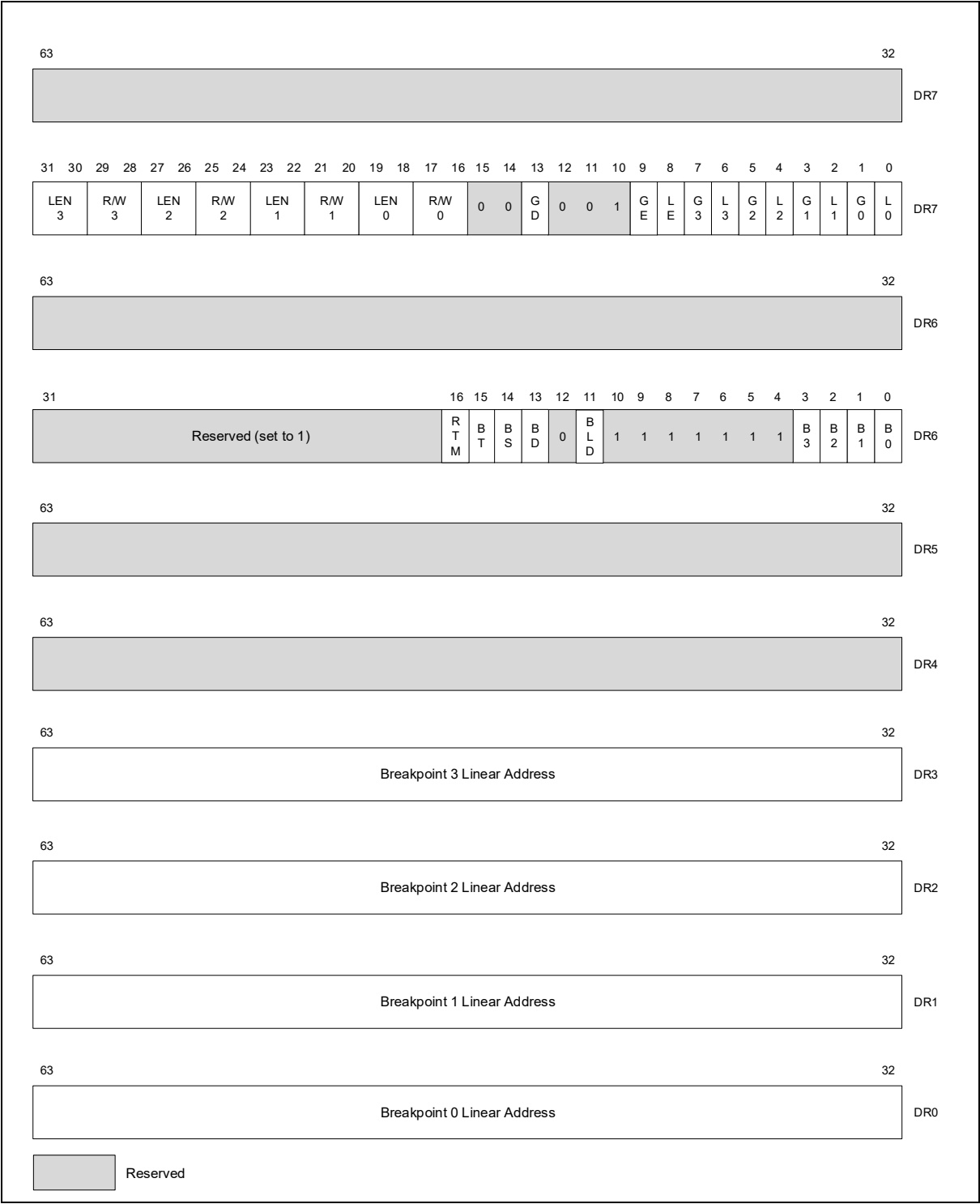


Figure 19-2. DR6/DR7 Layout on Processors Supporting Intel® 64 Architecture

Table 19-2. Debug Exception Conditions

Debug or Breakpoint Condition	DR6 Flags Tested	DR7 Flags Tested	Exception Class
Single-step trap	BS = 1		Trap
Instruction breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 0	Fault
Data write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 1	Trap
I/O read or write breakpoint, at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 2	Trap
Data read or write (but not instruction fetches), at addresses defined by DR n and LEN n	B n = 1 and (G n or L n = 1)	R/W n = 3	Trap
General detect fault, resulting from an attempt to modify debug registers (usually in conjunction with in-circuit emulation)	BD = 1	None	Fault
Task switch	BT = 1	None	Trap
INT1 instruction	None	None	Trap

19.3.1.1 Instruction-Breakpoint Exception Condition

The processor reports an instruction breakpoint when it attempts to execute an instruction at an address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect instruction execution (R/W flag is set to 0). Upon reporting the instruction breakpoint, the processor generates a fault-class, debug exception (#DB) before it executes the target instruction for the breakpoint.

Instruction breakpoints are the highest priority debug exceptions. They are serviced before any other exceptions detected during the decoding or execution of an instruction. However, if an instruction breakpoint is placed on an instruction located immediately after a POP SS/MOV SS instruction, the breakpoint will be suppressed as if EFLAGS.RF were 1 (see the next paragraph and Section 7.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).

Because the debug exception for an instruction breakpoint is generated before the instruction is executed, if the instruction breakpoint is not removed by the exception handler; the processor will detect the instruction breakpoint again when the instruction is restarted and generate another debug exception. To prevent looping on an instruction breakpoint, the Intel 64 and IA-32 architectures provide the RF flag (resume flag) in the EFLAGS register (see Section 2.3, “System Flags and Fields in the EFLAGS Register,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). When the RF flag is set, the processor ignores instruction breakpoints.

All Intel 64 and IA-32 processors manage the RF flag as follows. The RF Flag is cleared at the start of the instruction after the check for instruction breakpoints, CS limit violations, and FP exceptions. Task Switches and IRETD/IRETQ instructions transfer the RF image from the TSS/stack to the EFLAGS register.

When calling an event handler, Intel 64 and IA-32 processors establish the value of the RF flag in the EFLAGS image pushed on the stack:

- For any fault-class exception except a debug exception generated in response to an instruction breakpoint, the value pushed for RF is 1.
- For any interrupt arriving after any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For any trap-class exception generated by any iteration of a repeated string instruction but the last iteration, the value pushed for RF is 1.
- For other cases, the value pushed for RF is the value that was in EFLAG.RF at the time the event handler was called. This includes:
 - Debug exceptions generated in response to instruction breakpoints
 - Hardware-generated interrupts arriving between instructions (including those arriving after the last iteration of a repeated string instruction)

- Trap-class exceptions generated after an instruction completes (including those generated after the last iteration of a repeated string instruction)
- Software-generated interrupts (RF is pushed as 0, since it was cleared at the start of the software interrupt)

As noted above, the processor does not set the RF flag prior to calling the debug exception handler for debug exceptions resulting from instruction breakpoints. The debug exception handler can prevent recurrence of the instruction breakpoint by setting the RF flag in the EFLAGS image on the stack. If the RF flag in the EFLAGS image is set when the processor returns from the exception handler, it is copied into the RF flag in the EFLAGS register by IRETD/IRETQ or a task switch that causes the return. The processor then ignores instruction breakpoints for the duration of the next instruction. (Note that the POPF, POPFD, and IRET instructions do not transfer the RF image into the EFLAGS register.) Setting the RF flag does not prevent other types of debug-exception conditions (such as, I/O or data breakpoints) from being detected, nor does it prevent non-debug exceptions from being generated.

For the Pentium processor, when an instruction breakpoint coincides with another fault-type exception (such as a page fault), the processor may generate one spurious debug exception after the second exception has been handled, even though the debug exception handler set the RF flag in the EFLAGS image. To prevent a spurious exception with Pentium processors, all fault-class exception handlers should set the RF flag in the EFLAGS image.

19.3.1.2 Data Memory and I/O Breakpoint Exception Conditions

Data memory and I/O breakpoints are reported when the processor attempts to access a memory or I/O address specified in a breakpoint-address register (DR0 through DR3) that has been set up to detect data or I/O accesses (R/W flag is set to 1, 2, or 3). The processor generates the exception after it executes the instruction that made the access, so these breakpoint condition causes a trap-class exception to be generated.

Because data breakpoints are traps, an instruction that writes memory overwrites the original data before the debug exception generated by a data breakpoint is generated. If a debugger needs to save the contents of a write breakpoint location, it should save the original contents before setting the breakpoint. The handler can report the saved value after the breakpoint is triggered. The address in the debug registers can be used to locate the new value stored by the instruction that triggered the breakpoint.

If a data breakpoint is detected during an iteration of a string instruction executed with fast-string operation (see Section 7.3.9.3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1), delivery of the resulting debug exception may be delayed until completion of the corresponding group of iterations.

Intel486 and later processors ignore the GE and LE flags in DR7. In Intel386 processors, exact data breakpoint matching does not occur unless it is enabled by setting the LE and/or the GE flags.

For repeated INS and OUTS instructions that generate an I/O-breakpoint debug exception, the processor generates the exception after the completion of the first iteration. Repeated INS and OUTS instructions generate a data-breakpoint debug exception after the iteration in which the memory address breakpoint location is accessed.

If an execution of the MOV or POP instruction loads the SS register and encounters a data breakpoint, the resulting debug exception is delivered after completion of the next instruction (the one after the MOV or POP).

Any pending data or I/O breakpoints are lost upon delivery of an exception. For example, if a machine-check exception (#MC) occurs following an instruction that encounters a data breakpoint (but before the resulting debug exception is delivered), the data breakpoint is lost. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, the data breakpoint is lost if the next instruction causes a fault.

Delivery of events due to INT n , INT3, or INTO does not cause a loss of data breakpoints. If a MOV or POP instruction that loads the SS register encounters a data breakpoint, and the next instruction is software interrupt (INT n , INT3, or INTO), a debug exception (#DB) resulting from a data breakpoint will be delivered after the transition to the software-interrupt handler. The #DB handler should account for the fact that the #DB may have been delivered after a invocation of a software-interrupt handler, and in particular that the CPL may have changed between recognition of the data breakpoint and delivery of the #DB.

19.3.1.3 General-Detect Exception Condition

When the GD flag in DR7 is set, the general-detect debug exception occurs when a program attempts to access any of the debug registers (DR0 through DR7) at the same time they are being used by another application, such as an emulator or debugger. This protection feature guarantees full control over the debug registers when required. The

debug exception handler can detect this condition by checking the state of the BD flag in the DR6 register. The processor generates the exception before it executes the MOV instruction that accesses a debug register, which causes a fault-class exception to be generated.

19.3.1.4 Single-Step Exception Condition

The processor generates a single-step debug exception if (while an instruction is being executed) it detects that the TF flag in the EFLAGS register is set. The exception is a trap-class exception, because the exception is generated after the instruction is executed. The processor will not generate this exception after the instruction that sets the TF flag. For example, if the POPF instruction is used to set the TF flag, a single-step trap does not occur until after the instruction that follows the POPF instruction.

The processor clears the TF flag before calling the exception handler. If the TF flag was set in a TSS at the time of a task switch, the exception occurs after the first instruction is executed in the new task.

The TF flag normally is not cleared by privilege changes inside a task. The INT *n*, INT3, and INTO instructions, however, do clear this flag. Therefore, software debuggers that single-step code must recognize and emulate INT *n* or INTO instructions rather than executing them directly. To maintain protection, the operating system should check the CPL after any single-step trap to see if single stepping should continue at the current privilege level.

The interrupt priorities guarantee that, if an external interrupt occurs, single stepping stops. When both an external interrupt and a single-step interrupt occur together, the single-step interrupt is processed first. This operation clears the TF flag. After saving the return address or switching tasks, the external interrupt input is examined before the first instruction of the single-step handler executes. If the external interrupt is still pending, then it is serviced. The external interrupt handler does not run in single-step mode. To single step an interrupt handler, single step an INT *n* instruction that calls the interrupt handler.

If an occurrence of the MOV or POP instruction loads the SS register executes with EFLAGS.TF = 1, no single-step debug exception occurs following the MOV or POP instruction.

19.3.1.5 Task-Switch Exception Condition

The processor generates a debug exception after a task switch if the T flag of the new task's TSS is set. This exception is generated after program control has passed to the new task, and prior to the execution of the first instruction of that task. The exception handler can detect this condition by examining the BT flag of the DR6 register.

If entry 1 (#DB) in the IDT is a task gate, the T bit of the corresponding TSS should not be set. Failure to observe this rule will put the processor in a loop.

19.3.1.6 OS Bus-Lock Detection

OS bus-lock detection is a feature that causes the processor to generate a debug exception (called a **bus-lock detection debug exception**) if it detects that a bus lock has been asserted (see Section 10.1.2). Such an exception is a trap-class exception, because it is generated after execution of an instruction that asserts a bus lock. The exception thus does not prevent assertion of the bus lock. Delivery of a bus-lock detection debug exception clears DR6.BLD.

Software can enable OS bus-lock detection by setting IA32_DEBUGCTL.BLD[bit 2]. Bus-lock detection debug exceptions occur only if CPL > 0.

19.3.2 Breakpoint Exception (#BP)—Interrupt Vector 3

The breakpoint exception (interrupt 3) is caused by execution of an INT3 instruction. See Chapter 7, "Interrupt 3—Breakpoint Exception (#BP)." Debuggers use breakpoint exceptions in the same way that they use the breakpoint registers; that is, as a mechanism for suspending program execution to examine registers and memory locations. With earlier IA-32 processors, breakpoint exceptions are used extensively for setting instruction breakpoints.

With the Intel386 and later IA-32 processors, it is more convenient to set breakpoints with the breakpoint-address registers (DR0 through DR3). However, the breakpoint exception still is useful for breakpointing debuggers,

because a breakpoint exception can call a separate exception handler. The breakpoint exception is also useful when it is necessary to set more breakpoints than there are debug registers or when breakpoints are being placed in the source code of a program under development.

19.3.3 Debug Exceptions, Breakpoint Exceptions, and Restricted Transactional Memory (RTM)

Chapter 16, “Programming with Intel® AVX10,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, describes Restricted Transactional Memory (RTM). This is an instruction-set interface that allows software to identify **transactional regions** (or critical sections) using the XBEGIN and XEND instructions.

Execution of an RTM transactional region begins with an XBEGIN instruction. If execution of the region successfully reaches an XEND instruction, the processor ensures that all memory operations performed within the region appear to have occurred instantaneously when viewed from other logical processors. Execution of an RTM transaction region does not succeed if the processor cannot commit the updates atomically. When this happens, the processor rolls back the execution, a process referred to as a **transactional abort**. In this case, the processor discards all updates performed in the region, restores architectural state to appear as if the execution had not occurred, and resumes execution at a fallback instruction address that was specified with the XBEGIN instruction.

If debug exception (#DB) or breakpoint exception (#BP) occurs within an RTM transaction region, a transactional abort occurs, the processor sets EAX[4], and no exception is delivered.

Software can enable **advanced debugging of RTM transactional regions** by setting DR7.RTM[bit 11] and IA32_DEBUGCTL.RTM[bit 15]. If these bits are both set, the transactional abort caused by a #DB or #BP within an RTM transaction region does **not** resume execution at the fallback instruction address specified with the XBEGIN instruction that begin the region. Instead, execution is resumed at that XBEGIN instruction, and a #DB is delivered. (A #DB is delivered even if the transactional abort was caused by a #BP.) Such a #DB will clear DR6.RTM[bit 16] (all other debug exceptions set DR6[16]).

19.4 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING OVERVIEW

P6 family processors introduced the ability to set breakpoints on taken branches, interrupts, and exceptions, and to single-step from one branch to the next. This capability has been modified and extended in the Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors to allow logging of branch trace messages in a branch trace store (BTS) buffer in memory.

See the following sections for processor specific implementation of last branch, interrupt, and exception recording:

- Section 19.5, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ 2 Duo and Intel Atom® Processors).”
- Section 19.6, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Microarchitecture.”
- Section 19.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture.”
- Section 19.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture.”
- Section 19.11, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Haswell Microarchitecture.”
- Section 19.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture.”
- Section 19.14, “Last Branch, Interrupt, and Exception Recording (Intel® Core™ Solo and Intel® Core™ Duo Processors).”
- Section 19.15, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors).”
- Section 19.16, “Last Branch, Interrupt, and Exception Recording (P6 Family Processors).”

The following subsections of Section 19.4 describe common features of profiling branches. These features are generally enabled using the IA32_DEBUGCTL MSR (older processor may have implemented a subset or model-specific features, see definitions of MSR_DEBUGCTLA, MSR_DEBUGCTLB, MSR_DEBUGCTL).

19.4.1 IA32_DEBUGCTL MSR

The **IA32_DEBUGCTL** MSR provides bit field controls to enable debug trace interrupts, debug trace stores, trace messages enable, single stepping on branches, last branch record recording, and to control freezing of LBR stack or performance counters on a PMI request. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 19-3 for the MSR layout and the bullets below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the Section 19.5.1, “LBR Stack” (Intel® Core™2 Duo and Intel Atom® processor family) and Section 19.9.1, “LBR Stack” (processors based on Nehalem microarchitecture).
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 19.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **BLD (bus-lock detection) flag (bit 2)** — If this bit is set, OS bus-lock detection is enabled when CPL > 0. See Section 19.3.1.6.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 19.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 19.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bit 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 19.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

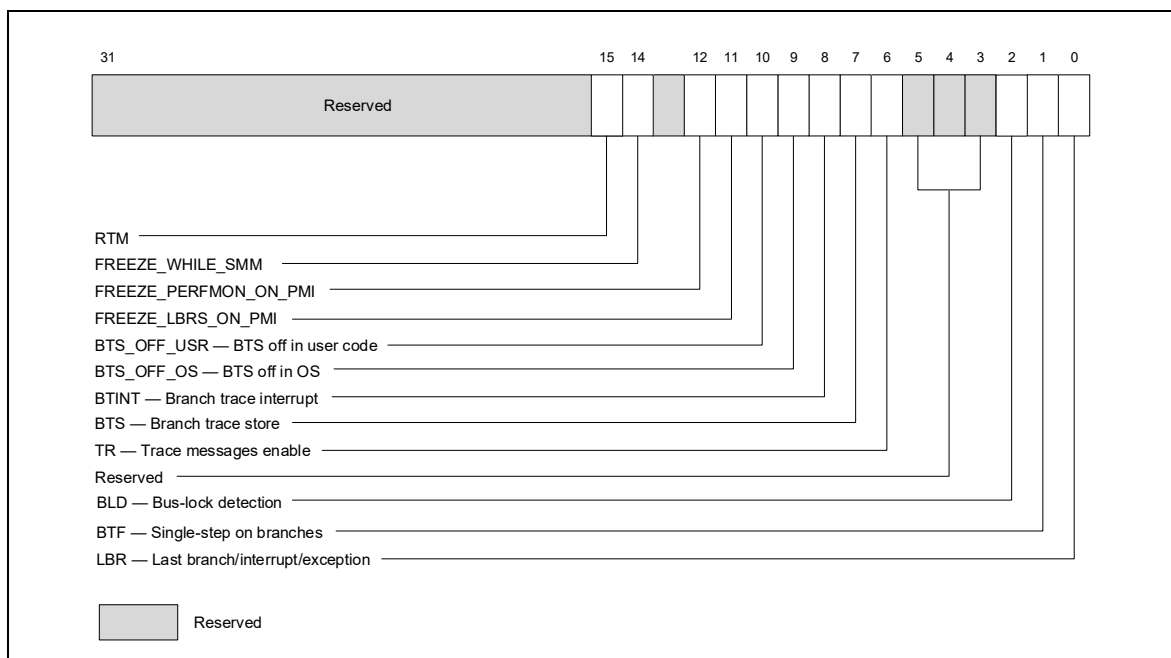


Figure 19-3. IA32_DEBUGCTL MSR for Processors Based on Intel® Core™ Microarchitecture

- **BTS_OFF_OS (branch trace off in privileged code) flag (bit 9)** — When set, BTS or BTM is skipped if CPL is 0. See Section 19.13.2.
- **BTS_OFF_USR (branch trace off in user code) flag (bit 10)** — When set, BTS or BTM is skipped if CPL is greater than 0. See Section 19.13.2.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — When set, the LBR stack is frozen on a hardware PMI request (e.g., when a counter overflows and is configured to trigger PMI). See Section 19.4.7 for details.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — When set, the performance counters (IA32_PMCx and IA32_FIXED_CTRx) are frozen on a PMI request. See Section 19.4.7 for details.
- **FREEZE_WHILE_SMM (bit 14)** — If this bit is set, upon the delivery of an SMI, the processor will clear all the enable bits of IA32_PERF_GLOBAL_CTRL, save a copy of the content of IA32_DEBUGCTL and disable LBR, BTF, TR, and BTS fields of IA32_DEBUGCTL before transferring control to the SMI handler. If Intel Thread Director support was enabled before transferring control to the SMI handler, then the processor will also reset the Intel Thread Director history (see Section 16.6.11 for more details about Intel Thread Director enable, reset, and history reset operations).
Subsequently, the enable bits of IA32_PERF_GLOBAL_CTRL will be set to 1, the saved copy of IA32_DEBUGCTL prior to SMI delivery will be restored, after the SMI handler issues RSM to complete its service. If Intel Thread Director support is enabled when RSM is executed, then the processor resets the Intel Thread Director history.
Note that system software must check if the processor supports the IA32_DEBUGCTL.FREEZE_WHILE_SMM control bit. IA32_DEBUGCTL.FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 21.8 for details of detecting the presence of IA32_PERF_CAPABILITIES MSR.
- **RTM (bit 15)** — If this bit is set, advanced debugging of RTM transactional regions is enabled if DR7.RTM is also set. See Section 19.3.3.

19.4.2 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag (bit 0) in the IA32_DEBUGCTL MSR is set, the processor automatically begins recording branch records for taken branches, interrupts, and exceptions (except for debug exceptions) in the LBR stack MSRs.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler. This action does not clear previously stored LBR stack MSRs.

A debugger can use the linear addresses in the LBR stack to re-set breakpoints in the breakpoint address registers (DR0 through DR3). This allows a backward trace from the manifestation of a particular bug toward its source.

On some processors, if the LBR flag is cleared and TR flag in the IA32_DEBUGCTL MSR remains set, the processor will continue to update LBR stack MSRs. This is because those processors use the entries in the LBR stack in the process of generating BTM/BTS records. A #DB does not automatically clear the TR flag.

19.4.3 Single-Stepping on Branches

When software sets both the BTF flag (bit 1) in the IA32_DEBUGCTL MSR and the TF flag in the EFLAGS register, the processor generates a single-step debug exception only after instructions that cause a branch.¹ This mechanism allows a debugger to single-step on control transfers caused by branches. This “branch single stepping” helps isolate a bug to a particular block of code before instruction single-stepping further narrows the search. The processor clears the BTF flag when it generates a debug exception. The debugger must set the BTF flag before resuming program execution to continue single-stepping on branches.

1. Executions of CALL, IRET, and JMP that cause task switches never cause single-step debug exceptions (regardless of the value of the BTF flag). A debugger desiring debug exceptions on switches to a task should set the T flag (debug trap flag) in the TSS of that task. See Section 9.2.1, “Task-State Segment (TSS).”

19.4.4 Branch Trace Messages

Setting the TR flag (bit 6) in the IA32_DEBUGCTL MSR enables branch trace messages (BTMs). Thereafter, when the processor detects a branch, exception, or interrupt, it sends a branch record out on the system bus as a BTM. A debugging device that is monitoring the system bus can read these messages and synchronize operations with taken branch, interrupt, and exception events.

When interrupts or exceptions occur in conjunction with a taken branch, additional BTMs are sent out on the bus, as described in Section 19.4.2, “Monitoring Branches, Exceptions, and Interrupts.”

For the P6 processor family, Pentium M processor family, and processors based on Intel Core microarchitecture, TR and LBR bits can not be set at the same time due to hardware limitation. The content of LBR stack is undefined when TR is set.

For processors with Intel NetBurst microarchitecture, Intel Atom processors, and Intel Core and related Intel Xeon processors both starting with the Nehalem microarchitecture, the processor can collect branch records in the LBR stack and at the same time send/store BTMs when both the TR and LBR flags are set in the IA32_DEBUGCTL MSR (or the equivalent MSR_DEBUGCTLA, MSR_DEBUGCTLB).

The following exception applies:

- BTM may not be observable on Intel Atom processor families that do not provide an externally visible system bus (i.e., processors based on the Silvermont microarchitecture or later).

19.4.4.1 Branch Trace Message Visibility

Branch trace message (BTM) visibility is implementation specific and limited to systems with a front side bus (FSB). BTMs may not be visible to newer system link interfaces or a system bus that deviates from a traditional FSB.

19.4.5 Branch Trace Store (BTS)

A trace of taken branches, interrupts, and exceptions is useful for debugging code by providing a method of determining the decision path taken to reach a particular code location. The LBR flag (bit 0) of IA32_DEBUGCTL provides a mechanism for capturing records of taken branches, interrupts, and exceptions and saving them in the last branch record (LBR) stack MSRs, setting the TR flag for sending them out onto the system bus as BTMs. The branch trace store (BTS) mechanism provides the additional capability of saving the branch records in a memory-resident BTS buffer, which is part of the DS save area. The BTS buffer can be configured to be circular so that the most recent branch records are always available or it can be configured to generate an interrupt when the buffer is nearly full so that all the branch records can be saved. The BTINT flag (bit 8) can be used to enable the generation of interrupt when the BTS buffer is full. See Section 19.4.9.2, “Setting Up the DS Save Area,” for additional details.

Setting this flag (BTS) alone can greatly reduce the performance of the processor. CPL-qualified branch trace storing mechanism can help mitigate the performance impact of sending/logging branch trace messages.

19.4.6 CPL-Qualified Branch Trace Mechanism

CPL-qualified branch trace mechanism is available to a subset of Intel 64 and IA-32 processors that support the branch trace storing mechanism. The processor supports the CPL-qualified branch trace mechanism if CPUID.01H:ECX[4] = 1.

The CPL-qualified branch trace mechanism is described in Section 19.4.9.4. System software can selectively specify CPL qualification to not send/store Branch Trace Messages associated with a specified privilege level. Two bit fields, BTS_OFF_USR (bit 10) and BTS_OFF_OS (bit 9), are provided in the debug control register to specify the CPL of BTMs that will not be logged in the BTS buffer or sent on the bus.

19.4.7 Freezing LBR and Performance Counters on PMI

Many issues may generate a performance monitoring interrupt (PMI); a PMI service handler will need to determine cause to handle the situation. Two capabilities that allow a PMI service routine to improve branch tracing and performance monitoring are available for processors supporting architectural performance monitoring version 2 or

greater (i.e., CPUID.0AH:EAX[7:0] > 1). These capabilities provides the following interface in IA32_DEBUGCTL to reduce runtime overhead of PMI servicing, profiler-contributed skew effects on analysis or counter metrics:

- **Freezing LBRs on PMI (bit 11)** — Allows the PMI service routine to ensure the content in the LBR stack are associated with the target workload and not polluted by the branch flows of handling the PMI. Depending on the version ID enumerated by CPUID.0AH:EAX.VERSION[7:0], two flavors are supported:
 - Legacy Freeze_LBR_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID > 1. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the LBR is frozen on the overflowed condition of the buffer area, the processor clears the LBR bit (bit 0) in IA32_DEBUGCTL. Software must then re-enable IA32_DEBUGCTL.LBR to resume recording branches. When using this feature, software should be careful about writes to IA32_DEBUGCTL to avoid re-enabling LBRs by accident if they were just disabled.
 - Streamlined Freeze_LBR_on_PMI is supported for ArchPerfMonVerID >= 4. If IA32_DEBUGCTL.Freeze_LBR_On_PMI = 1, the processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.LBR_Frz = 1 to disable recording, but does not change the LBR bit (bit 0) in IA32_DEBUGCTL. The LBRs are frozen on the overflowed condition of the buffer area.
- **Freezing PMCs on PMI (bit 12)** — Allows the PMI service routine to ensure the content in the performance counters are associated with the target workload and not polluted by the PMI and activities within the PMI service routine. Depending on the version ID enumerated by CPUID.0AH:EAX.VERSION[7:0], two flavors are supported:
 - Legacy Freeze_PerfMon_on_PMI is supported for ArchPerfMonVerID <= 3 and ArchPerfMonVerID > 1. If IA32_DEBUGCTL.Freeze_PerfMon_On_PMI = 1, the performance counters are frozen on the counter overflowed condition when the processor clears the IA32_PERF_GLOBAL_CTRL MSR (see Figure 21-3). The PMCs affected include both general-purpose counters and fixed-function counters (see Section 21.6.2.1, “Fixed-function Performance Counters”). Software must re-enable counts by writing 1s to the corresponding enable bits in IA32_PERF_GLOBAL_CTRL before leaving a PMI service routine to continue counter operation.
 - Streamlined Freeze_PerfMon_on_PMI is supported for ArchPerfMonVerID >= 4. The processor behaves as follows:
 - sets IA32_PERF_GLOBAL_STATUS.CTR_Frz = 1 to disable counting on a counter overflow condition, but does not change the IA32_PERF_GLOBAL_CTRL MSR.

Freezing LBRs and PMCs on PMIs (both legacy and streamlined operation) occur when one of the following applies:

- A performance counter had an overflow and was programmed to signal a PMI in case of an overflow.
 - For the general-purpose counters; enabling PMI is done by setting bit 20 of the IA32_PERFEVTSELx register.
 - For the fixed-function counters; enabling PMI is done by setting the 3rd bit in the corresponding 4-bit control field of the MSR_PERF_FIXED_CTR_CTRL register (see Figure 21-1) or IA32_FIXED_CTR_CTRL MSR (see Figure 21-2).
- The PEBS buffer is almost full and reaches the interrupt threshold.
- The BTS buffer is almost full and reaches the interrupt threshold.

Table 19-3 compares the interaction of the processor with the PMI handler using the legacy versus streamlined Freeza_PerfMon_On_PMI interface.

Table 19-3. Legacy and Streamlined Operation with Freeze_PerfMon_On_PMI = 1, Counter Overflowed

Legacy Freeze_PerfMon_On_PMI	Streamlined Freeze_PerfMon_On_PMI	Comment
Processor freezes the counters on overflow	Processor freezes the counters on overflow	Unchanged
Processor clears IA32_PERF_GLOBAL_CTRL	Processor set IA32_PERF_GLOBAL_STATUS.CTR_FTZ	
Handler reads IA32_PERF_GLOBAL_STATUS(0x38E) to examine which counter(s) overflowed	mask = RDMSR(0x38E)	Similar
Handler services the PMI	Handler services the PMI	Unchanged
Handler writes 1s to IA32_PERF_GLOBAL_OVF_CTL (0x390)	Handler writes mask into IA32_PERF_GLOBAL_OVF_RESET (0x390)	
Processor clears IA32_PERF_GLOBAL_STATUS	Processor clears IA32_PERF_GLOBAL_STATUS	Unchanged
Handler re-enables IA32_PERF_GLOBAL_CTRL	None	Reduced software overhead

19.4.8 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel 64 and IA-32 processor families. However, the number of MSRs in the LBR stack and the valid range of TOS pointer value can vary between different processor families. Table 19-4 lists the LBR stack size and TOS pointer range for several processor families according to the CPUID signatures of DisplayFamily_DisplayModel encoding (see the CPUID instruction in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A).

Table 19-4. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Component of an LBR Entry	Range of TOS Pointer
06_5CH, 06_5FH	32	FROM_IP, TO_IP	0 to 31
06_4EH, 06_5EH, 06_8EH, 06_9EH, 06_55H, 06_66H, 06_7AH, 06_67H, 06_6AH, 06_6CH, 06_7DH, 06_7EH, 06_8CH, 06_8DH, 06_6AH, 06_A5H, 06_A6H, 06_A7H, 06_A8H, 06_86H, 06_8AH, 06_96H, 06_9CH	32	FROM_IP, TO_IP, LBR_INFO ¹	0 to 31
06_3DH, 06_47H, 06_4FH, 06_56H, 06_3CH, 06_45H, 06_46H, 06_3FH, 06_2AH, 06_2DH, 06_3AH, 06_3EH, 06_1AH, 06_1EH, 06_1FH, 06_2EH, 06_25H, 06_2CH, 06_2FH	16	FROM_IP, TO_IP	0 to 15
06_17H, 06_1DH, 06_0FH	4	FROM_IP, TO_IP	0 to 3
06_37H, 06_4AH, 06_4CH, 06_4DH, 06_5AH, 06_5DH, 06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	8	FROM_IP, TO_IP	0 to 7

NOTES:

1. See Section 19.12.

The last branch recording mechanism tracks not only branch instructions (e.g., JMP, Jcc, LOOP, and CALL instructions), but also other operations that cause a change in the instruction pointer (e.g., external interrupts, traps, and faults). The branch recording mechanisms generally employs a set of MSRs, referred to as last branch record (LBR) stack. The size and exact locations of the LBR stack are generally model-specific (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for model-specific MSR addresses).

- **Last Branch Record (LBR) Stack** — The LBR consists of N pairs of MSRs (N is listed in the LBR stack size column of Table 19-4) that store source and destination address of recent branches (see Figure 19-3):
 - MSR_LASTBRANCH_0_FROM_IP (address is model specific) through the next consecutive (N-1) MSR address store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address is model specific) through the next consecutive (N-1) MSR address store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant M bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address is model specific) contains an M-bit pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. The valid range of the M-bit POS pointer is given in Table 19-4.

19.4.8.1 LBR Stack and Intel® 64 Processors

LBR MSRs are 64-bits. In 64-bit mode, last branch records store the full address. Outside of 64-bit mode, the upper 32-bits of branch addresses will be stored as 0.

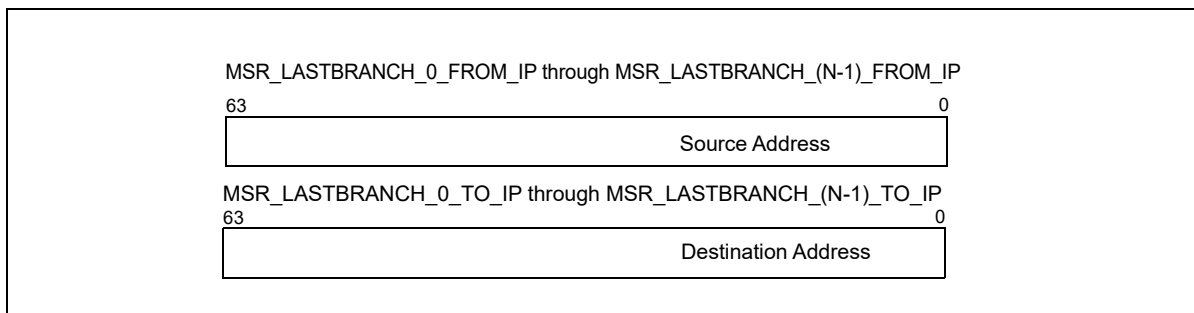


Figure 19-4. 64-bit Address Layout of LBR MSR

Software should query an architectural MSR IA32_PERF_CAPABILITIES[5:0] about the format of the address that is stored in the LBR stack. Four formats are defined by the following encoding:

- **000000B (32-bit record format)** — Stores 32-bit offset in current CS of respective source/destination,
- **000001B (64-bit LIP record format)** — Stores 64-bit linear address of respective source/destination,
- **000010B (64-bit EIP record format)** — Stores 64-bit offset (effective address) of respective source/destination.
- **000011B (64-bit EIP record format) and Flags** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction info is reported in the upper bit of 'FROM' registers in the LBR stack. See LBR stack details below for flag support and definition.
- **000100B (64-bit EIP record format), Flags, and TSX** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction and TSX info are reported in the upper bits of 'FROM' registers in the LBR stack.
- **000101B (64-bit EIP record format), Flags, TSX, and LBR_INFO** — Stores 64-bit offset (effective address) of respective source/destination. Misprediction, TSX, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.
- **000110B (64-bit LIP record format), Flags, and Cycles** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction info is reported in the upper bits of

'FROM' registers in the LBR stack. Elapsed cycles since the last LBR update are reported in the upper 16 bits of the 'TO' registers in the LBR stack (see Section 19.6).

- **000111B (64-bit LIP record format), Flags, and LBR_INFO** — Stores 64-bit linear address (CS.Base + effective address) of respective source/destination. Misprediction, and elapsed cycles since the last LBR update are reported in the LBR_INFO MSR stack.

Processor's support for the architectural MSR IA32_PERF_CAPABILITIES is provided by CPUID.01H:ECX.PERF_CAPABILITIES (bit 15).

19.4.8.2 LBR Stack and IA-32 Processors

The LBR MSRs in IA-32 processors introduced prior to Intel 64 architecture store the 32-bit "To Linear Address" and "From Linear Address" using the high and low half of each 64-bit MSR.

19.4.8.3 Last Exception Records and Intel 64 Architecture

Intel 64 and IA-32 processors also provide MSRs that store the branch record for the last branch taken prior to an exception or an interrupt. The location of the last exception record (LER) MSRs are model specific. The MSRs that store last exception records are 64-bits. If IA-32e mode is disabled, only the lower 32-bits of the address is recorded. If IA-32e mode is enabled, the processor writes 64-bit values into the MSR. In 64-bit mode, last exception records store 64-bit addresses; in compatibility mode, the upper 32-bits of last exception records are cleared.

19.4.9 BTS and DS Save Area

The **Debug store (DS)** feature flag (bit 21), returned by CPUID.01H:EDX[21] indicates that the processor provides the debug store (DS) mechanism. The DS mechanism allows:

- BTMs to be stored in a memory-resident BTS buffer. See Section 19.4.5, "Branch Trace Store (BTS)."
- Processor event-based sampling (PEBS) also uses the DS save area provided by debug store mechanism. The capability of PEBS varies across different microarchitectures. See Section 21.6.2.4, "Processor Event Based Sampling (PEBS)," and the relevant PEBS sub-sections across the core PMU sections in Chapter 21, "Performance Monitoring."

When CPUID.01H:EDX[21] is set:

- The BTS_UNAVAILABLE and PEBS_UNAVAILABLE flags in the IA32_MISC_ENABLE MSR indicate (when clear) the availability of the BTS and PEBS facilities, including the ability to set the BTS and BTINT bits in the appropriate DEBUGCTL MSR.
- The IA32_DS_AREA MSR exists and points to the DS save area.

The debug store (DS) save area is a software-designated area of memory that is used to collect the following two types of information:

- **Branch records** — When the BTS flag in the IA32_DEBUGCTL MSR is set, a branch record is stored in the BTS buffer in the DS save area whenever a taken branch, interrupt, or exception is detected.
- **PEBS records** — When a performance counter is configured for PEBS, a PEBS record is stored in the PEBS buffer in the DS save area after the counter overflow occurs. This record contains the architectural state of the processor (state of the 8 general purpose registers, EIP register, and EFLAGS register) at the next occurrence of the PEBS event that caused the counter to overflow. When the state information has been logged, the counter is automatically reset to a specified value, and event counting begins again. The content layout of a PEBS record varies across different implementations that support PEBS. See Section 21.6.2.4.2 for details of enumerating PEBS record format.

NOTES

Prior to processors based on the Goldmont microarchitecture, PEBS facility only supports a subset of implementation-specific precise events. See Section 21.5.3.1 for a PEBS enhancement that can generate records for both precise and non-precise events.

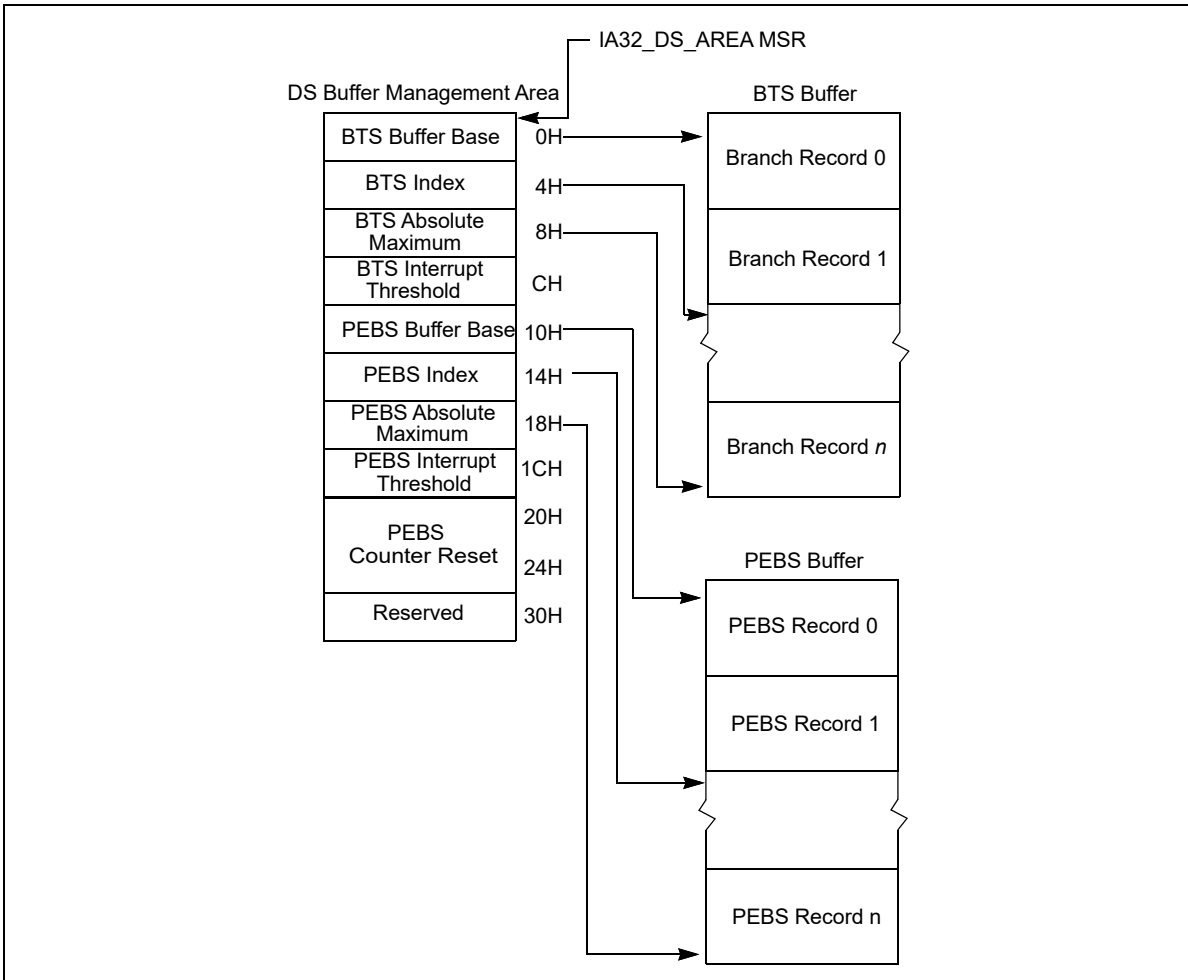
The DS save area and recording mechanism are disabled on INIT, processor Reset or transition to system-management mode (SMM) or IA-32e mode. It is similarly disabled on the generation of a machine-check exception on 45nm and 32nm Intel Atom processors and on processors with Netburst or Intel Core microarchitecture.

The BTS and PEBS facilities may not be available on all processors. The availability of these facilities is indicated by the `BTS_UNAVAILABLE` and `PEBS_UNAVAILABLE` flags, respectively, in the `IA32_MISC_ENABLE` MSR (see Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4).

The DS save area is divided into three parts: buffer management area, branch trace store (BTS) buffer, and PEBS buffer (see Figure 19-5). The buffer management area is used to define the location and size of the BTS and PEBS buffers. The processor then uses the buffer management area to keep track of the branch and/or PEBS records in their respective buffers and to record the performance counter reset value. The linear address of the first byte of the DS buffer management area is specified with the `IA32_DS_AREA` MSR.

The fields in the buffer management area are as follows:

- **BTS buffer base** — Linear address of the first byte of the BTS buffer. This address should point to a natural doubleword boundary.
- **BTS index** — Linear address of the first byte of the next BTS record to be written to. Initially, this address should be the same as the address in the BTS buffer base field.
- **BTS absolute maximum** — Linear address of the next byte past the end of the BTS buffer. This address should be a multiple of the BTS record size (12 bytes) plus 1.
- **BTS interrupt threshold** — Linear address of the BTS record on which an interrupt is to be generated. This address must point to an offset from the BTS buffer base that is a multiple of the BTS record size. Also, it must be several records short of the BTS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the BTS absolute maximum record.
- **PEBS buffer base** — Linear address of the first byte of the PEBS buffer. This address should point to a natural doubleword boundary.
- **PEBS index** — Linear address of the first byte of the next PEBS record to be written to. Initially, this address should be the same as the address in the PEBS buffer base field.
- **PEBS absolute maximum** — Linear address of the next byte past the end of the PEBS buffer. This address should be a multiple of the PEBS record size (40 bytes) plus 1.
- **PEBS interrupt threshold** — Linear address of the PEBS record on which an interrupt is to be generated. This address must point to an offset from the PEBS buffer base that is a multiple of the PEBS record size. Also, it must be several records short of the PEBS absolute maximum address to allow a pending interrupt to be handled prior to processor writing the PEBS absolute maximum record.
- **PEBS counter reset value** — A 64-bit value that the counter is to be set to when a PEBS record is written. Bits beyond the size of the counter are ignored. This value allows state information to be collected regularly every time the specified number of events occur.

Figure 19-5. DS Save Area Example¹**NOTES:**

1. This example represents the format for a system that supports PEBS on only one counter.

Figure 19-6 shows the structure of a 12-byte branch record in the BTS buffer. The fields in each record are as follows:

- **Last branch from** — Linear address of the instruction from which the branch, interrupt, or exception was taken.
- **Last branch to** — Linear address of the branch target or the first instruction in the interrupt or exception service routine.
- **Branch predicted** — Bit 4 of field indicates whether the branch that was taken was predicted (set) or not predicted (clear).

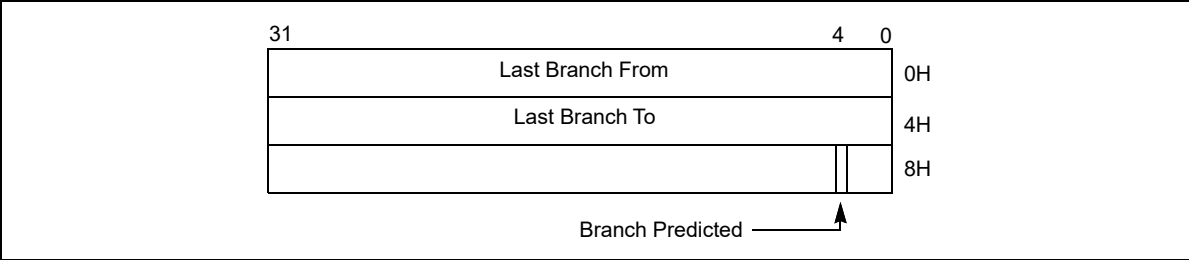


Figure 19-6. 32-bit Branch Trace Record Format

Figure 19-7 shows the structure of the 40-byte PEBS records. Nominally the register values are those at the beginning of the instruction that caused the event. However, there are cases where the registers may be logged in a partially modified state. The linear IP field shows the value in the EIP register translated from an offset into the current code segment to a linear address.

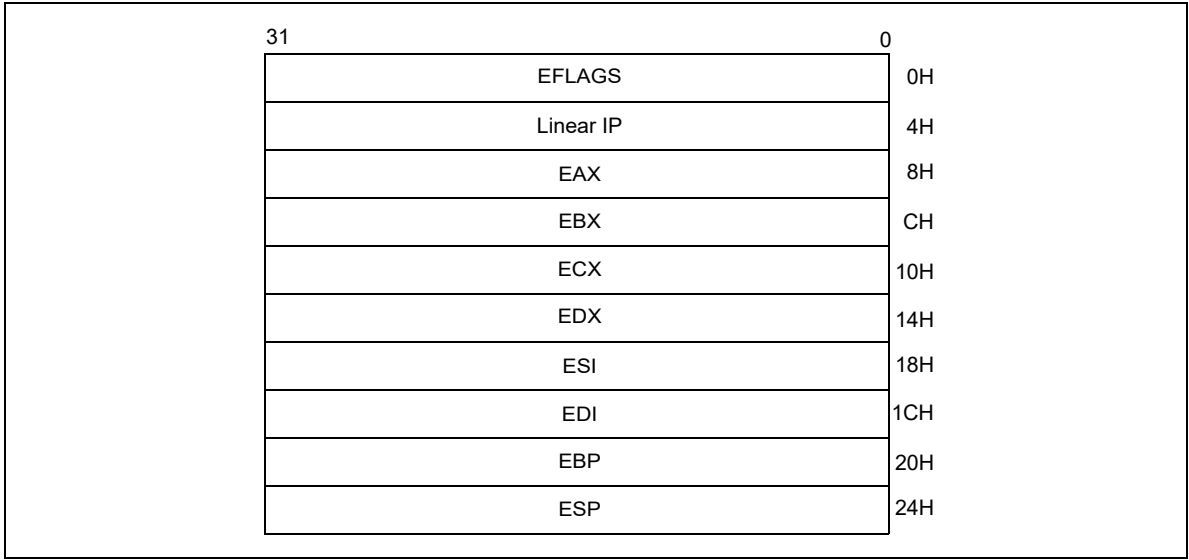


Figure 19-7. PEBS Record Format

19.4.9.1 64 Bit Format of the DS Save Area

When DTES64 = 1 (CPUID.01H:ECX[2] = 1), the structure of the DS save area is shown in Figure 19-8. When DTES64 = 0 (CPUID.01H:ECX[2] = 0) and IA-32e mode is active, the structure of the DS save area is shown in Figure 19-8. If IA-32e mode is not active the structure of the DS save area is as shown in Figure 19-5.

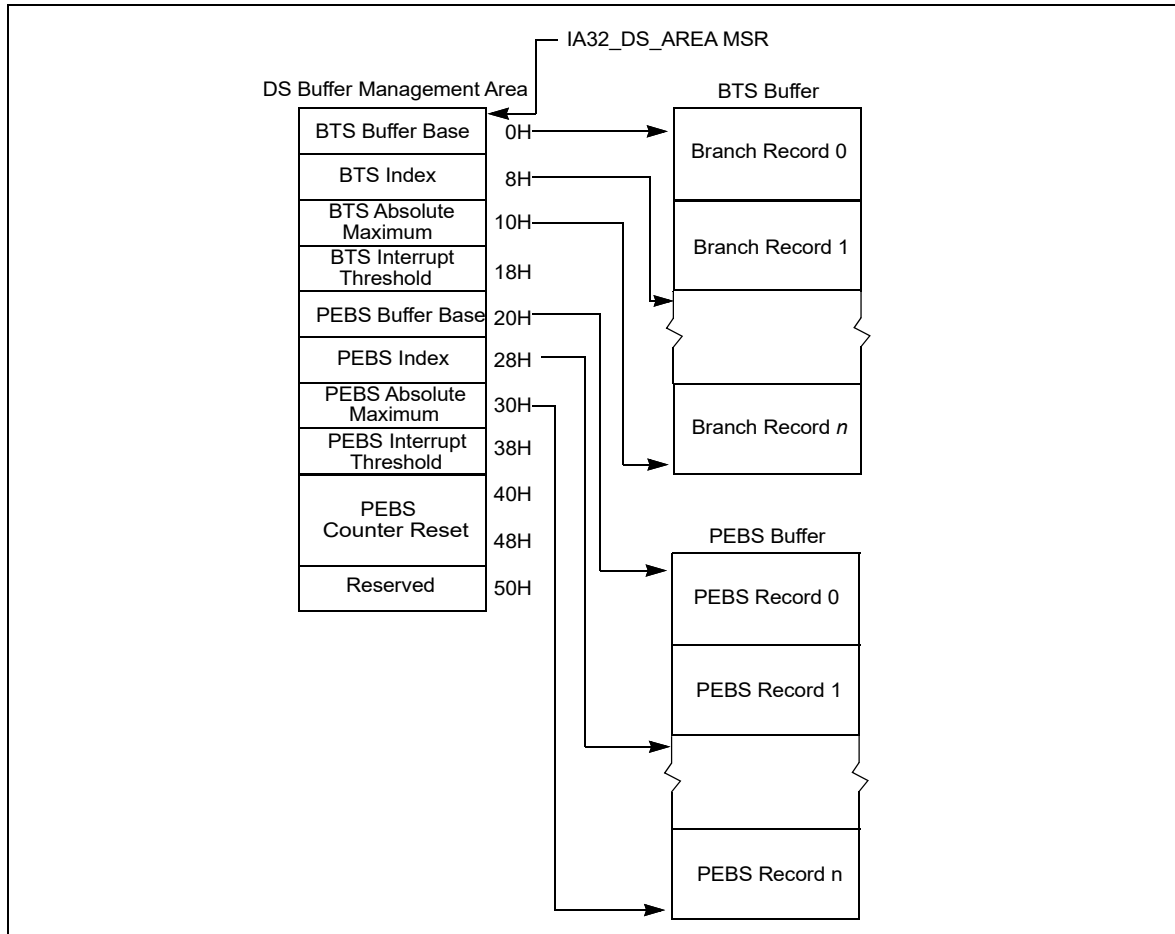


Figure 19-8. IA-32e Mode DS Save Area Example¹

NOTES:

1. This example represents the format for a system that supports PEBS on only one counter.

The IA32_DS_AREA MSR holds the 64-bit linear address of the first byte of the DS buffer management area. The structure of a branch trace record is similar to that shown in Figure 19-6, but each field is 8 bytes in length. This makes each BTS record 24 bytes (see Figure 19-9). The structure of a PEBS record is similar to that shown in Figure 19-7, but each field is 8 bytes in length and architectural states include register R8 through R15. This makes the size of a PEBS record in 64-bit mode 144 bytes (see Figure 19-10).

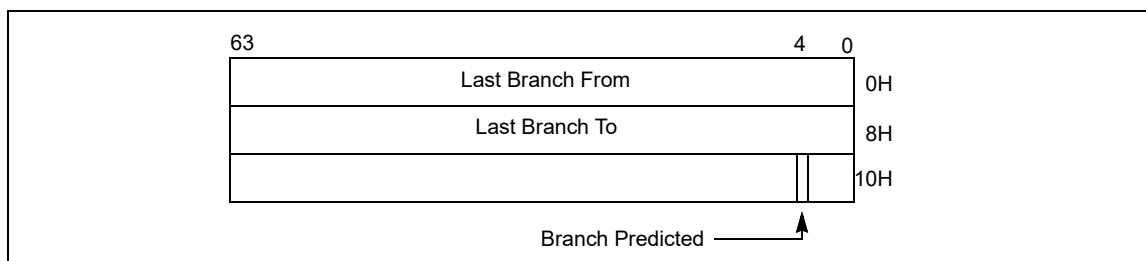


Figure 19-9. 64-bit Branch Trace Record Format

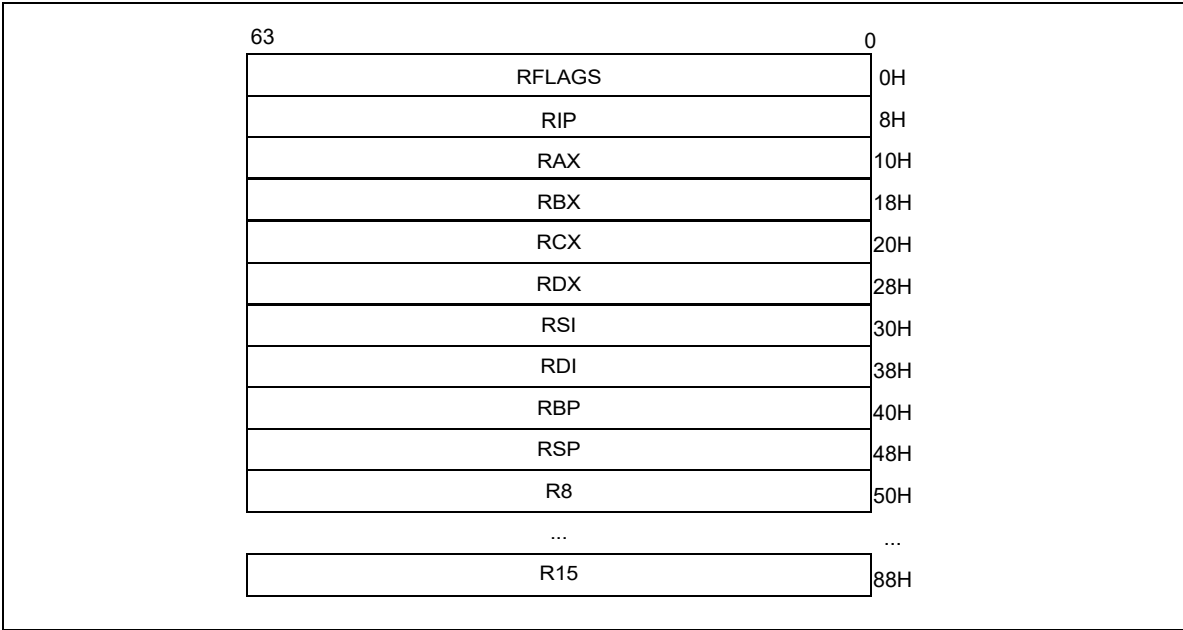


Figure 19-10. 64-bit PEBS Record Format

Fields in the buffer management area of a DS save area are described in Section 19.4.9.

The format of a branch trace record and a PEBS record are the same as the 64-bit record formats shown in Figures 19-9 and Figures 19-10, with the exception that the branch predicted bit is not supported by Intel Core microarchitecture or Intel Atom microarchitecture. The 64-bit record formats for BTS and PEBS apply to DS save area for all operating modes.

The procedures used to program IA32_DEBUGCTL MSR to set up a BTS buffer or a CPL-qualified BTS are described in Section 19.4.9.3 and Section 19.4.9.4.

Required elements for writing a DS interrupt service routine are largely the same on processors that support using DS Save area for BTS or PEBS records. However, on processors based on Intel NetBurst® microarchitecture, re-enabling counting requires writing to CCCRs. But a DS interrupt service routine on processors supporting architectural performance monitoring should:

- Re-enable the enable bits in IA32_PERF_GLOBAL_CTRL MSR if it is servicing an overflow PMI due to PEBS.
- Clear overflow indications by writing to IA32_PERF_GLOBAL_OVF_CTRL when a counting configuration is changed. This includes bit 62 (ClrOvfBuffer) and the overflow indication of counters used in either PEBS or general-purpose counting (specifically: bits 0 or 1; see Figures 21-3).

19.4.9.2 Setting Up the DS Save Area

To save branch records with the BTS buffer, the DS save area must first be set up in memory as described in the following procedure (See Section 21.6.2.4.1, “Setting up the PEBS Buffer,” for instructions for setting up a PEBS buffer, respectively, in the DS save area):

1. Create the DS buffer management information area in memory (see Section 19.4.9, “BTS and DS Save Area,” and Section 19.4.9.1, “64 Bit Format of the DS Save Area”). Also see the additional notes in this section.
2. Write the base linear address of the DS buffer management area into the IA32_DS_AREA MSR.
3. Set up the performance counter entry in the xAPIC LVT for fixed delivery and edge sensitive. See Section 12.5.1, “Local Vector Table.”
4. Establish an interrupt handler in the IDT for the vector associated with the performance counter entry in the xAPIC LVT.

5. Write an interrupt service routine to handle the interrupt. See Section 19.4.9.5, “Writing the DS Interrupt Service Routine.”

The following restrictions should be applied to the DS save area.

- The recording of branch records in the BTS buffer (or PEBS records in the PEBS buffer) may not operate properly if accesses to the linear addresses in any of the three DS save area sections cause page faults, VM exits, or the setting of accessed or dirty flags in the paging structures (ordinary or EPT). For that reason, system software should establish paging structures (both ordinary and EPT) to prevent such occurrences. Implications of this may be that an operating system should allocate this memory from a non-paged pool and that system software cannot do “lazy” page-table entry propagation for these pages. Some newer processor generations support “lazy” EPT page-table entry propagation for PEBS; see Section 21.3.9.1 and Section 21.9.5 for more information. A virtual-machine monitor may choose to allow use of PEBS by guest software only if EPT maps all guest-physical memory as present and read/write.
- The DS save area can be larger than a page, but the pages must be mapped to contiguous linear addresses. The buffer may share a page, so it need not be aligned on a 4-KByte boundary. For performance reasons, the base of the buffer must be aligned on a doubleword boundary and should be aligned on a cache line boundary.
- It is recommended that the buffer size for the BTS buffer and the PEBS buffer be an integer multiple of the corresponding record sizes.
- The precise event records buffer should be large enough to hold the number of precise event records that can occur while waiting for the interrupt to be serviced.
- The DS save area should be in kernel space. It must not be on the same page as code, to avoid triggering self-modifying code actions.
- There are no memory type restrictions on the buffers, although it is recommended that the buffers be designated as WB memory type for performance considerations.
- Either the system must be prevented from entering A20M mode while DS save area is active, or bit 20 of all addresses within buffer bounds must be 0.
- Pages that contain buffers must be mapped to the same physical addresses for all processes, such that any change to control register CR3 will not change the DS addresses.
- The DS save area is expected to be used only on systems with an enabled APIC. The LVT Performance Counter entry in the APIC must be initialized to use an interrupt gate instead of the trap gate.

19.4.9.3 Setting Up the BTS Buffer

Three flags in the MSR_DEBUGCTLA MSR (see Table 19-5), IA32_DEBUGCTL (see Figure 19-3), or MSR_DEBUGCTLB (see Figure 19-16) control the generation of branch records and storing of them in the BTS buffer; these are TR, BTS, and BTINT. The TR flag enables the generation of BTMs. The BTS flag determines whether the BTMs are sent out on the system bus (clear) or stored in the BTS buffer (set). BTMs cannot be simultaneously sent to the system bus and logged in the BTS buffer. The BTINT flag enables the generation of an interrupt when the BTS buffer is full. When this flag is clear, the BTS buffer is a circular buffer.

Table 19-5. IA32_DEBUGCTL Flag Encodings

TR	BTS	BTINT	Description
0	X	X	Branch trace messages (BTMs) off
1	0	X	Generate BTMs
1	1	0	Store BTMs in the BTS buffer, used here as a circular buffer
1	1	1	Store BTMs in the BTS buffer, and generate an interrupt when the buffer is nearly full

The following procedure describes how to set up a DS Save area to collect branch records in the BTS buffer:

1. Place values in the BTS buffer base, BTS index, BTS absolute maximum, and BTS interrupt threshold fields of the DS buffer management area to set up the BTS buffer in memory.
2. Set the TR and BTS flags in the IA32_DEBUGCTL for Intel Core Solo and Intel Core Duo processors or later processors (or MSR_DEBUGCTLA MSR for processors based on Intel NetBurst Microarchitecture; or MSR_DEBUGCTLB for Pentium M processors).

3. Clear the BTINT flag in the corresponding IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR; or MSR_DEBUGCTLB) if a circular BTS buffer is desired.

NOTES

If the buffer size is set to less than the minimum allowable value (i.e., BTS absolute maximum < 1 + size of BTS record), the results of BTS is undefined.

In order to prevent generating an interrupt, when working with circular BTS buffer, SW need to set BTS interrupt threshold to a value greater than BTS absolute maximum (fields of the DS buffer management area). It's not enough to clear the BTINT flag itself only.

19.4.9.4 Setting Up CPL-Qualified BTS

If the processor supports CPL-qualified last branch recording mechanism, the generation of branch records and storing of them in the BTS buffer are determined by: TR, BTS, BTS_OFF_OS, BTS_OFF_USR, and BTINT. The encoding of these five bits are shown in Table 19-6.

Table 19-6. CPL-Qualified Branch Trace Store Encodings

TR	BTS	BTS_OFF_OS	BTS_OFF_USR	BTINT	Description
0	X	X	X	X	Branch trace messages (BTMs) off
1	0	X	X	X	Generates BTMs but do not store BTMs
1	1	0	0	0	Store all BTMs in the BTS buffer, used here as a circular buffer
1	1	1	0	0	Store BTMs with CPL > 0 in the BTS buffer
1	1	0	1	0	Store BTMs with CPL = 0 in the BTS buffer
1	1	1	1	X	Generate BTMs but do not store BTMs
1	1	0	0	1	Store all BTMs in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	1	0	1	Store BTMs with CPL > 0 in the BTS buffer; generate an interrupt when the buffer is nearly full
1	1	0	1	1	Store BTMs with CPL = 0 in the BTS buffer; generate an interrupt when the buffer is nearly full

19.4.9.5 Writing the DS Interrupt Service Routine

The BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector and interrupt service routine (called the debug store interrupt service routine or DS ISR). To handle BTS, non-precise event-based sampling, and PEBS interrupts: separate handler routines must be included in the DS ISR. Use the following guidelines when writing a DS ISR to handle BTS, non-precise event-based sampling, and/or PEBS interrupts.

- The DS interrupt service routine (ISR) must be part of a kernel driver and operate at a current privilege level of 0 to secure the buffer storage area.
- Because the BTS, non-precise event-based sampling, and PEBS facilities share the same interrupt vector, the DS ISR must check for all the possible causes of interrupts from these facilities and pass control on to the appropriate handler.

BTS and PEBS buffer overflow would be the sources of the interrupt if the buffer index matches/exceeds the interrupt threshold specified. Detection of non-precise event-based sampling as the source of the interrupt is accomplished by checking for counter overflow.

- There must be separate save areas, buffers, and state for each processor in an MP system.
- Upon entering the ISR, branch trace messages and PEBS should be disabled to prevent race conditions during access to the DS save area. This is done by clearing TR flag in the IA32_DEBUGCTL (or MSR_DEBUGCTLA MSR) and by clearing the precise event enable flag in the MSR_PEBS_ENABLE MSR. These settings should be restored to their original values when exiting the ISR.

- The processor will not disable the DS save area when the buffer is full and the circular mode has not been selected. The current DS setting must be retained and restored by the ISR on exit.
- After reading the data in the appropriate buffer, up to but not including the current index into the buffer, the ISR must reset the buffer index to the beginning of the buffer. Otherwise, everything up to the index will look like new entries upon the next invocation of the ISR.
- The ISR must clear the mask bit in the performance counter LVT entry.
- The ISR must re-enable the counters to count via IA32_PERF_GLOBAL_CTRL/IA32_PERF_GLOBAL_OVF_CTRL if it is servicing an overflow PMI due to PEBS (or via CCCR's ENABLE bit on processor based on Intel NetBurst microarchitecture).
- The Pentium 4 Processor and Intel Xeon Processor mask PMIs upon receiving an interrupt. Clear this condition before leaving the interrupt handler.

19.5 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ 2 DUO AND INTEL ATOM® PROCESSORS)

The Intel Core 2 Duo processor family and Intel Xeon processors based on Intel Core microarchitecture or enhanced Intel Core microarchitecture provide last branch interrupt and exception recording. The facilities described in this section also apply to 45 nm and 32 nm Intel Atom processors. These capabilities are similar to those found in Pentium 4 processors, including support for the following facilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provide bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 19.4.1 for a description of the flags. See Figure 19-3 for the MSR layout.
- **Last branch record (LBR) stack** — There are a collection of MSR pairs that store the source and destination addresses related to recently executed branches. See Section 19.5.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts**
 - See Section 19.4.2 and Section 19.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.
 - 45 nm and 32 nm Intel Atom processors clear the TR flag when the FREEZE_LBRS_ON_PMI flag is set.
- **Branch trace messages** — See Section 19.4.4.
- **Last exception records** — See Section 19.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 19.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 19.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 19.4.7 for legacy Freeze_PerfMon_On_PMI operation.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 19.4.1.

19.5.1 LBR Stack

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported across Intel Core 2, Intel Atom processor families, and Intel processors based on Intel NetBurst microarchitecture.

Four pairs of MSRs are supported in the LBR stack for Intel Core 2 processors families and Intel processors based on Intel NetBurst microarchitecture:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_3_FROM_IP (address 43H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_3_TO_IP (address 63H) store destination addresses

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 2 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

Eight pairs of MSRs are supported in the LBR stack for 45 nm and 32 nm Intel Atom processors:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 40H) through MSR_LASTBRANCH_7_FROM_IP (address 47H) store source addresses
 - MSR_LASTBRANCH_0_TO_IP (address 60H) through MSR_LASTBRANCH_7_TO_IP (address 67H) store destination addresses
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

The address format written in the FROM_IP/TO_IP MSRS may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 19.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

19.5.2 LBR Stack in Intel Atom® Processors based on the Silvermont Microarchitecture

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in Intel Atom processors based on the Silvermont and Airmont microarchitectures. Eight pairs of MSRs are supported in the LBR stack.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 19-11.

19.6 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Processors based on the Goldmont microarchitecture extend the capabilities described in Section 19.5.2 with the following enhancements:

- Supports new LBR format encoding 00110b in IA32_PERF_CAPABILITIES[5:0].
- Size of LBR stack increased to 32. Each entry includes MSR_LASTBRANCH_x_FROM_IP (address 0x680..0x69f) and MSR_LASTBRANCH_x_TO_IP (address 0x6c0..0x6df).
- LBR call stack filtering supported. The layout of MSR_LBR_SELECT is described in Table 19-13.
- Elapsed cycle information is added to MSR_LASTBRANCH_x_TO_IP. Format is shown in Table 19-7.
- Misprediction info is reported in the upper bits of MSR_LASTBRANCH_x_FROM_IP. MISPREDE bit format is shown in Table 19-8.
- Streamlined Freeze_LBRs_On_PMI operation; see Section 19.12.2.
- LBR MSRs may be cleared when MWAIT is used to request a C-state that is numerically higher than C1; see Section 19.12.3.

Table 19-7. MSR_LASTBRANCH_x_TO_IP for the Goldmont Microarchitecture

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 19.4.8.1 for address format.
Cycle Count (Saturating)	63:48	R/W	Elapsed core clocks since last update to the LBR stack.

19.7 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Next generation Intel Atom processors are based on the Goldmont Plus microarchitecture. Processors based on the Goldmont Plus microarchitecture extend the capabilities described in Section 19.6 with the following changes:

- Enumeration of new LBR format: encoding 00111b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 19.4.8.1.
- Each LBR stack entry consists of three MSRs:
 - MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 19-9.
 - MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 19-9.
 - MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data. Layout is the same as Table 19-16.

19.8 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR INTEL® XEON PHI™ PROCESSOR 7200/5200/3200

The last branch record stack and top-of-stack (TOS) pointer MSRs are supported in the Intel® Xeon Phi™ processor 7200/5200/3200 series based on the Knights Landing microarchitecture. Eight pairs of MSRs are supported in the LBR stack, per thread:

- **Last Branch Record (LBR) Stack**
 - MSR_LASTBRANCH_0_FROM_IP (address 680H) through MSR_LASTBRANCH_7_FROM_IP (address 687H) store source addresses.
 - MSR_LASTBRANCH_0_TO_IP (address 6C0H) through MSR_LASTBRANCH_7_TO_IP (address 6C7H) store destination addresses.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The lowest significant 3 bits of the TOS Pointer MSR (MSR_LASTBRANCH_TOS, address 1C9H) contains a pointer to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded.

LBR filtering is supported. Filtering of LBRs based on a combination of CPL and branch type conditions is supported. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT. The layout of MSR_LBR_SELECT is described in Table 19-11.

The address format written in the FROM_IP/TO_IP MSRs may differ between processors. Software should query IA32_PERF_CAPABILITIES[5:0] and consult Section 19.4.8.1. The behavior of the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs corresponds to that of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

19.9 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

The processors based on Nehalem microarchitecture and Westmere microarchitecture support last branch interrupt and exception recording. These capabilities are similar to those found in Intel Core 2 processors and add additional capabilities:

- **Debug Trace and Branch Recording Control** — The IA32_DEBUGCTL MSR provides bit fields for software to configure mechanisms related to debug trace, branch recording, branch trace store, and performance counter operations. See Section 19.4.1 for a description of the flags. See Figure 19-11 for the MSR layout.
- **Last branch record (LBR) stack** — There are 16 MSR pairs that store the source and destination addresses related to recently executed branches. See Section 19.9.1.
- **Monitoring and single-stepping of branches, exceptions, and interrupts** — See Section 19.4.2 and Section 19.4.3. In addition, the ability to freeze the LBR stack on a PMI request is available.

- **Branch trace messages** — The IA32_DEBUGCTL MSR provides bit fields for software to enable each logical processor to generate branch trace messages. See Section 19.4.4. However, not all BTM messages are observable using the Intel® QPI link.
- **Last exception records** — See Section 19.13.3.
- **Branch trace store and CPL-qualified BTS** — See Section 19.4.6 and Section 19.4.5.
- **FREEZE_LBRS_ON_PMI flag (bit 11)** — see Section 19.4.7 for legacy Freeze_LBRs_On_PMI operation.
- **FREEZE_PERFMON_ON_PMI flag (bit 12)** — see Section 19.4.7 for legacy Freeze_PerfMon_On_PMI operation.
- **UNCORE_PMI_EN (bit 13)** — When set, this logical processor is enabled to receive an counter overflow interrupt form the uncore.
- **FREEZE_WHILE_SMM (bit 14)** — FREEZE_WHILE_SMM is supported if IA32_PERF_CAPABILITIES.FREEZE_WHILE_SMM[Bit 12] is reporting 1. See Section 19.4.1.

Processors based on Nehalem microarchitecture provide additional capabilities:

- **Independent control of uncore PMI** — The IA32_DEBUGCTL MSR provides a bit field (see Figure 19-11) for software to enable each logical processor to receive an uncore counter overflow interrupt.
- **LBR filtering** — Processors based on Nehalem microarchitecture support filtering of LBR based on combination of CPL and branch type conditions. When LBR filtering is enabled, the LBR stack only captures the subset of branches that are specified by MSR_LBR_SELECT.

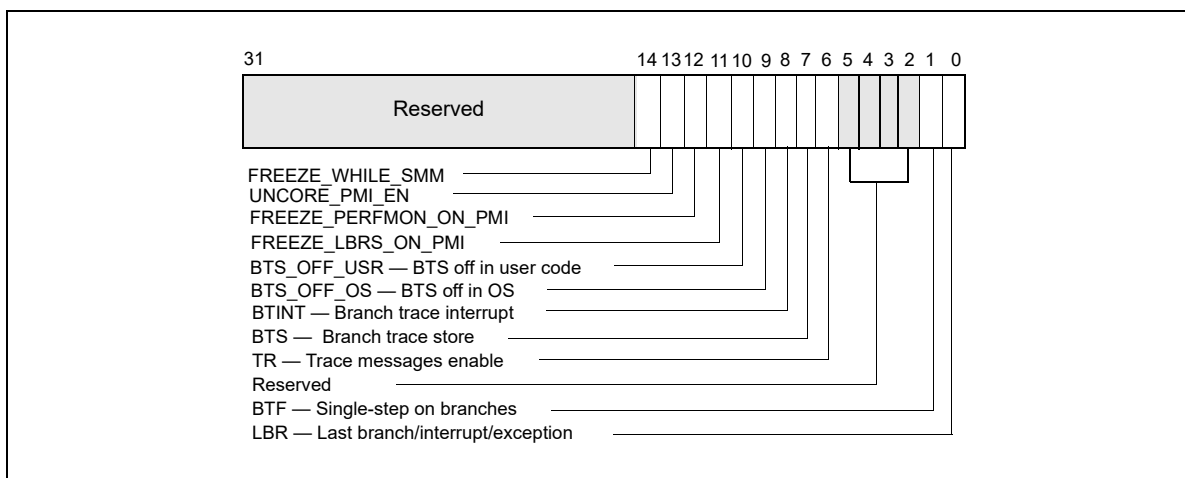


Figure 19-11. IA32_DEBUGCTL MSR for Processors Based on Nehalem Microarchitecture

19.9.1 LBR Stack

Processors based on Nehalem microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is shown in Table 19-8 and Table 19-9.

Table 19-8. MSR_LASTBRANCH_x_FROM_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the "branch from" address. See Section 19.4.8.1 for address format.
SIGN_EXT	62:48	R/W	Signed extension of bit 47 of this register.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

Table 19-9. MSR_LASTBRANCH_x_TO_IP

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch to” address. See Section 19.4.8.1 for address format
SIGN_EXT	63:48	R/W	Signed extension of bit 47 of this register.

Processors based on Nehalem microarchitecture have an LBR MSR Stack as shown in Table 19-10.

Table 19-10. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_1AH	16	0 to 15

19.9.2 Filtering of Last Branch Records

MSR_LBR_SELECT is cleared to zero at RESET, and LBR filtering is disabled, i.e., all branches will be captured. MSR_LBR_SELECT provides bit fields to specify the conditions of subsets of branches that will not be captured in the LBR. The layout of MSR_LBR_SELECT is shown in Table 19-11.

Table 19-11. MSR_LBR_SELECT for Nehalem Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

19.10 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SANDY BRIDGE MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 19.9, “Last Branch, Interrupt, and Exception Recording for Processors based on Nehalem Microarchitecture,” apply to processors based on Sandy Bridge microarchitecture. For processors based on Ivy Bridge microarchitecture, the same holds true.

One difference of note is that MSR_LBR_SELECT is shared between two logical processors in the same core. In Sandy Bridge microarchitecture, each logical processor has its own MSR_LBR_SELECT. The filtering semantics for “Near_ind_jmp” and “Near_rel_jmp” has been enhanced, see Table 19-12.

Table 19-12. MSR_LBR_SELECT for Sandy Bridge Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

19.11 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON HASWELL MICROARCHITECTURE

Generally, all of the last branch record, interrupt, and exception recording facility described in Section 19.10, “Last Branch, Interrupt, and Exception Recording for Processors based on Sandy Bridge Microarchitecture,” apply to next generation processors based on Haswell microarchitecture.

The LBR facility also supports an alternate capability to profile call stack profiles. Configuring the LBR facility to conduct call stack profiling is by writing 1 to the MSR_LBR_SELECT.EN_CALLSTACK[bit 9]; see Table 19-13. If MSR_LBR_SELECT.EN_CALLSTACK is clear, the LBR facility will capture branches normally as described in Section 19.10.

Table 19-13. MSR_LBR_SELECT for Haswell Microarchitecture

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches ending in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches ending in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps except near indirect calls and near returns
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps except near relative calls.
FAR_BRANCH	8	R/W	When set, do not capture far branches
EN_CALLSTACK ¹	9		Enable LBR stack to use LIFO filtering to capture Call stack profile
Reserved	63:10		Must be zero

NOTES:

1. Must set valid combination of bits 0-8 in conjunction with bit 9 (as described below), otherwise the contents of the LBR MSRs are undefined.

The call stack profiling capability is an enhancement of the LBR facility. The LBR stack is a ring buffer typically used to profile control flow transitions resulting from branches. However, the finite depth of the LBR stack often become less effective when profiling certain high-level languages (e.g., C++), where a transition of the execution flow is accompanied by a large number of leaf function calls, each of which returns an individual parameter to form the list

of parameters for the main execution function call. A long list of such parameters returned by the leaf functions would serve to flush the data captured in the LBR stack, often losing the main execution context.

When the call stack feature is enabled, the LBR stack will capture unfiltered call data normally, but as return instructions are executed the last captured branch record is flushed from the on-chip registers in a last-in first-out (LIFO) manner. Thus, branch information relative to leaf functions will not be captured, while preserving the call stack information of the main line execution path.

The configuration of the call stack facility is summarized below:

- Set IA32_DEBUGCTL.LBR (bit 0) to enable the LBR stack to capture branch records. The source and target addresses of the call branches will be captured in the 16 pairs of From/To LBR MSRs that form the LBR stack.
- Program the Top of Stack (TOS) MSR that points to the last valid from/to pair. This register is incremented by 1, modulo 16, before recording the next pair of addresses.
- Program the branch filtering bits of MSR_LBR_SELECT (bits 0:8) as desired.
- Program the MSR_LBR_SELECT to enable LIFO filtering of return instructions with:
 - The following bits in MSR_LBR_SELECT must be set to '1': JCC, NEAR_IND_JMP, NEAR_REL_JMP, FAR_BRANCH, EN_CALLSTACK;
 - The following bits in MSR_LBR_SELECT must be cleared: NEAR_REL_CALL, NEAR-IND_CALL, NEAR_RET;
 - At most one of CPL_EQ_0, CPL_NEQ_0 is set.

Note that when call stack profiling is enabled, “zero length calls” are excluded from writing into the LBRs. (A “zero length call” uses the attribute of the call instruction to push the immediate instruction pointer on to the stack and then pops off that address into a register. This is accomplished without any matching return on the call.)

19.11.1 LBR Stack Enhancement

Processors based on Haswell microarchitecture provide 16 pairs of MSR to record last branch record information. The layout of each MSR pair is enumerated by IA32_PERF_CAPABILITIES[5:0] = 04H, and is shown in Table 19-14 and Table 19-9.

Table 19-14. MSR_LASTBRANCH_x_FROM_IP with TSX Information

Bit Field	Bit Offset	Access	Description
Data	47:0	R/W	This is the “branch from” address. See Section 19.4.8.1 for address format.
SIGN_EXT	60:48	R/W	Signed extension of bit 47 of this register.
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region, or EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

19.12 LAST BRANCH, CALL STACK, INTERRUPT, AND EXCEPTION RECORDING FOR PROCESSORS BASED ON SKYLAKE MICROARCHITECTURE

Processors based on the Skylake microarchitecture provide a number of enhancement with storing last branch records:

- enumeration of new LBR format: encoding 00101b in IA32_PERF_CAPABILITIES[5:0] is supported, see Section 19.4.8.1.
- Each LBR stack entry consists of a triplets of MSRs:

- MSR_LASTBRANCH_x_FROM_IP, the layout is simplified, see Table 19-9.
- MSR_LASTBRANCH_x_TO_IP, the layout is the same as Table 19-9.
- MSR_LBR_INFO_x, stores branch prediction flag, TSX info, and elapsed cycle data.
- Size of LBR stack increased to 32.

Processors based on the Skylake microarchitecture support the same LBR filtering capabilities as described in Table 19-13.

Table 19-15. LBR Stack Size and TOS Pointer Range

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
06_4EH, 06_5EH	32	0 to 31

19.12.1 MSR_LBR_INFO_x MSR

The layout of each MSR_LBR_INFO_x MSR is shown in Table 19-16.

Table 19-16. MSR_LBR_INFO_x

Bit Field	Bit Offset	Access	Description
Cycle Count (saturating)	15:0	R/W	Elapsed core clocks since last update to the LBR stack.
Reserved	60:16	R/W	Reserved
TSX_ABORT	61	R/W	When set, indicates a TSX Abort entry LBR_FROM: EIP at the time of the TSX Abort LBR_TO: EIP of the start of HLE region OR EIP of the RTM Abort Handler
IN_TSX	62	R/W	When set, indicates the entry occurred in a TSX region.
MISPRED	63	R/W	When set, indicates either the target of the branch was mispredicted and/or the direction (taken/non-taken) was mispredicted; otherwise, the target branch was predicted.

19.12.2 Streamlined Freeze_LBRs_On_PMI Operation

The FREEZE_LBRs_ON_PMI feature causes the LBRs to be frozen on a hardware request for a PMI. This prevents the LBRs from being overwritten by new branches, allowing the PMI handler to examine the control flow that preceded the PMI generation. Architectural performance monitoring version 4 and above supports a streamlined FREEZE_LBRs_ON_PMI operation for PMI service routine that replaces the legacy FREEZE_LBRs_ON_PMI operation (see Section 19.4.7).

While the legacy FREEZE_LBRs_ON_PMI clear the LBR bit in the IA32_DEBUGCTL MSR on a PMI request, the streamlined FREEZE_LBRs_ON_PMI will set the LBR_FRZ bit in IA32_PERF_GLOBAL_STATUS. Branches will not cause the LBRs to be updated when LBR_FRZ is set. Software can clear LBR_FRZ at the same time as it clears overflow bits by setting the LBR_FRZ bit as well as the needed overflow bit when writing to IA32_PERF_GLOBAL_STATUS_RESET MSR.

This streamlined behavior avoids race conditions between software and processor writes to IA32_DEBUGCTL that are possible with FREEZE_LBRs_ON_PMI clearing of the LBR enable.

19.12.3 LBR Behavior and Deep C-State

When MWAIT is used to request a C-state that is numerically higher than C1, then LBR state may be initialized to zero depending on optimized “waiting” state that is selected by the processor. The affected LBR states include the FROM, TO, INFO, LAST_BRANCH, LER, and LBR_TOS registers. The LBR enable bit and LBR_FROZEN bit are not affected. The LBR-time of the first LBR record inserted after an exit from such a C-state request will be zero.

19.13 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PROCESSORS BASED ON INTEL NETBURST® MICROARCHITECTURE)

Pentium 4 and Intel Xeon processors based on Intel NetBurst microarchitecture provide the following methods for recording taken branches, interrupts, and exceptions:

- Store branch records in the last branch record (LBR) stack MSRs for the most recent taken branches, interrupts, and/or exceptions in MSRs. A branch record consists of a branch-from and a branch-to instruction address.
- Send the branch records out on the system bus as branch trace messages (BTMs).
- Log BTMs in a memory-resident branch trace store (BTS) buffer.

To support these functions, the processor provides the following MSRs and related facilities:

- **MSR_DEBUGCTLA MSR** — Enables last branch, interrupt, and exception recording; single-stepping on taken branches; branch trace messages (BTMs); and branch trace store (BTS). This register is named DebugCtlMSR in the P6 family processors.
- **Debug store (DS) feature flag (CPUID.01H:EDX.DS[21])** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer.
- **CPL-qualified debug store (DS) feature flag (CPUID.01H:ECX.DS_CPL[4])** — Indicates that the processor provides a CPL-qualified debug store (DS) mechanism, which allows software to selectively skip sending and storing BTMs, according to specified current privilege level settings, into a memory-resident BTS buffer.
- **IA32_MISC_ENABLE MSR** — Indicates that the processor provides the BTS facilities.
- **Last branch record (LBR) stack** — The LBR stack is a circular stack that consists of four MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. The LBR stack consists of 16 MSR pairs (MSR_LASTBRANCH_0_FROM_IP through MSR_LASTBRANCH_15_FROM_IP and MSR_LASTBRANCH_0_TO_IP through MSR_LASTBRANCH_15_TO_IP) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H].
- **Last branch record top-of-stack (TOS) pointer** — The TOS Pointer MSR contains a 2-bit pointer (0-3) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, models 0H-02H]. This pointer becomes a 4-bit pointer (0-15) for the Pentium 4 and Intel Xeon processor family [CPUID family 0FH, model 03H]. See also: Table 19-17, Figure 19-12, and Section 19.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **Last exception record** — See Section 19.13.3, “Last Exception Records.”

19.13.1 MSR_DEBUGCTLA MSR

The MSR_DEBUGCTLA MSR enables and disables the various last branch recording mechanisms described in the previous section. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode. A protected-mode operating system procedure is required to provide user access to this register. Figure 19-12 shows the flags in the MSR_DEBUGCTLA MSR. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. Each branch, interrupt, or exception is recorded as a 64-bit branch record. The processor clears this flag whenever a debug exception is generated (for example,

- when an instruction or data breakpoint or a single-step trap occurs). See Section 19.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 19.4.3, “Single-Stepping on Branches.”
 - **TR (trace message enable) flag (bit 2)** — When set, branch trace messages are enabled. Thereafter, when the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 19.4.4, “Branch Trace Messages.”

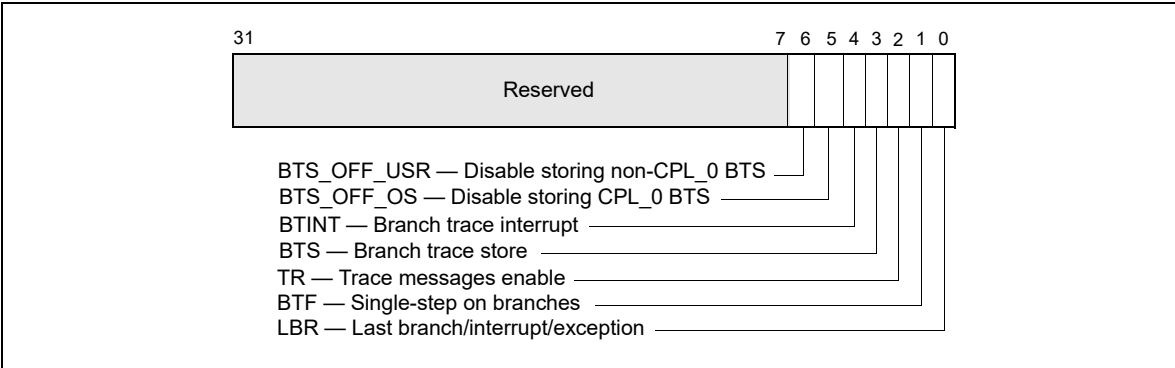


Figure 19-12. MSR_DEBUGCTLA MSR for Pentium 4 and Intel Xeon Processors

- **BTS (branch trace store) flag (bit 3)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 19.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 4)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 19.4.5, “Branch Trace Store (BTS).”
- **BTS_OFF_OS (disable ring 0 branch trace store) flag (bit 5)** — When set, enables the BTS facilities to skip sending/logging CPL_0 BTMs to the memory-resident BTS buffer. See Section 19.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”
- **BTS_OFF_USR (disable ring 0 branch trace store) flag (bit 6)** — When set, enables the BTS facilities to skip sending/logging non-CPL_0 BTMs to the memory-resident BTS buffer. See Section 19.13.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture.”

NOTE

The initial implementation of BTS_OFF_USR and BTS_OFF_OS in MSR_DEBUGCTLA is shown in Figure 19-12. The BTS_OFF_USR and BTS_OFF_OS fields may be implemented on other model-specific debug control register at different locations.

See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a detailed description of each of the last branch recording MSRs.

19.13.2 LBR Stack for Processors Based on Intel NetBurst® Microarchitecture

The LBR stack is made up of LBR MSRs that are treated by the processor as a circular stack. The TOS pointer (MSR_LASTBRANCH_TOS MSR) points to the LBR MSR (or LBR MSR pair) that contains the most recent (last) branch record placed on the stack. Prior to placing a new branch record on the stack, the TOS is incremented by 1. When the TOS pointer reaches its maximum value, it wraps around to 0. See Table 19-17 and Figure 19-12.

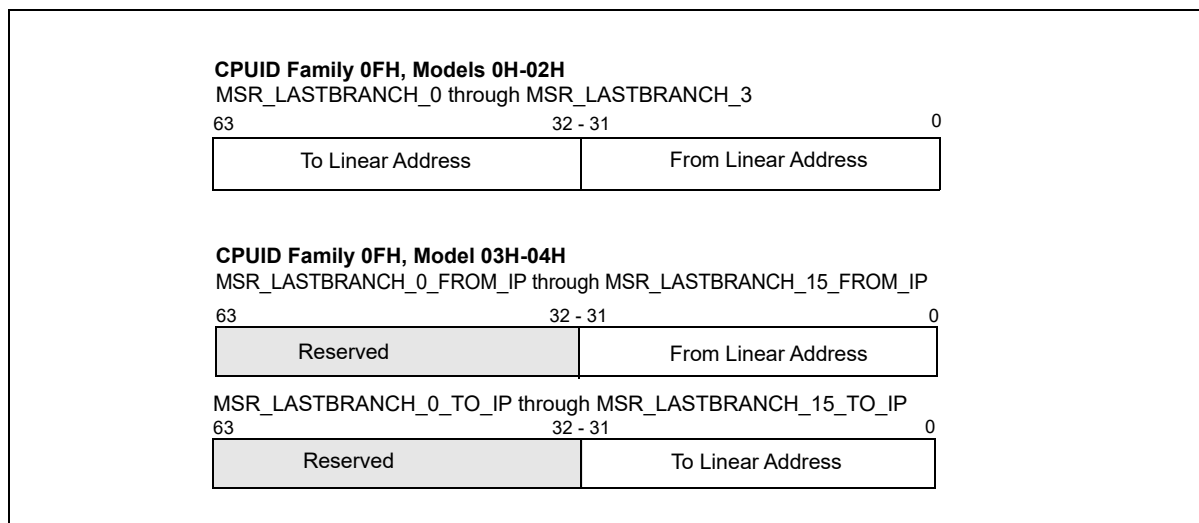
Table 19-17. LBR MSR Stack Size and TOS Pointer Range for the Pentium® 4 and the Intel® Xeon® Processor Family

DisplayFamily_DisplayModel	Size of LBR Stack	Range of TOS Pointer
Family 0FH, Models 0H-02H; MSRs at locations 1DBH-1DEH.	4	0 to 3
Family 0FH, Models; MSRs at locations 680H-68FH.	16	0 to 15
Family 0FH, Model 03H; MSRs at locations 6C0H-6CFH.	16	0 to 15

The registers in the LBR MSR stack and the MSR_LASTBRANCH_TOS MSR are read-only and can be read using the RDMSR instruction.

Figure 19-13 shows the layout of a branch record in an LBR MSR (or MSR pair). Each branch record consists of two linear addresses, which represent the “from” and “to” instruction pointers for a branch, interrupt, or exception. The contents of the from and to addresses differ, depending on the source of the branch:

- **Taken branch** — If the record is for a taken branch, the “from” address is the address of the branch instruction and the “to” address is the target instruction of the branch.
- **Interrupt** — If the record is for an interrupt, the “from” address the return instruction pointer (RIP) saved for the interrupt and the “to” address is the address of the first instruction in the interrupt handler routine. The RIP is the linear address of the next instruction to be executed upon returning from the interrupt handler.
- **Exception** — If the record is for an exception, the “from” address is the linear address of the instruction that caused the exception to be generated and the “to” address is the address of the first instruction in the exception handler routine.

**Figure 19-13. LBR MSR Branch Record Layout for the Pentium 4 and Intel® Xeon® Processor Family**

Additional information is saved if an exception or interrupt occurs in conjunction with a branch instruction. If a branch instruction generates a trap type exception, two branch records are stored in the LBR stack: a branch record for the branch instruction followed by a branch record for the exception.

If a branch instruction is immediately followed by an interrupt, a branch record is stored in the LBR stack for the branch instruction followed by a record for the interrupt.

19.13.3 Last Exception Records

The Pentium 4, Intel Xeon, Pentium M, Intel® Core™ Solo, Intel® Core™ Duo, Intel® Core™2 Duo, Intel® Core™ i7 and Intel Atom® processors provide two MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate the functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in the P6 family processors.

The MSR_LER_TO_LIP and MSR_LER_FROM_LIP MSRs contain a branch record for the last branch that the processor took prior to an exception or interrupt being generated.

19.14 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS)

Intel Core Solo and Intel Core Duo processors provide last branch interrupt and exception recording. This capability is almost identical to that found in Pentium 4 and Intel Xeon processors. There are differences in the stack and in some MSR names and locations.

Note the following:

- **IA32_DEBUGCTL MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. IA32_DEBUGCTL MSR is located at register address 01D9H.

See Figure 19-14 for the layout and the entries below for a description of the flags:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” below.
- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 19.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
- **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception; it sends the branch record out on the system bus as a branch trace message (BTM). See Section 19.4.4, “Branch Trace Messages,” for more information about the TR flag.
- **BTS (branch trace store) flag (bit 7)** — When set, the flag enables BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 19.4.9, “BTS and DS Save Area.”
- **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 19.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

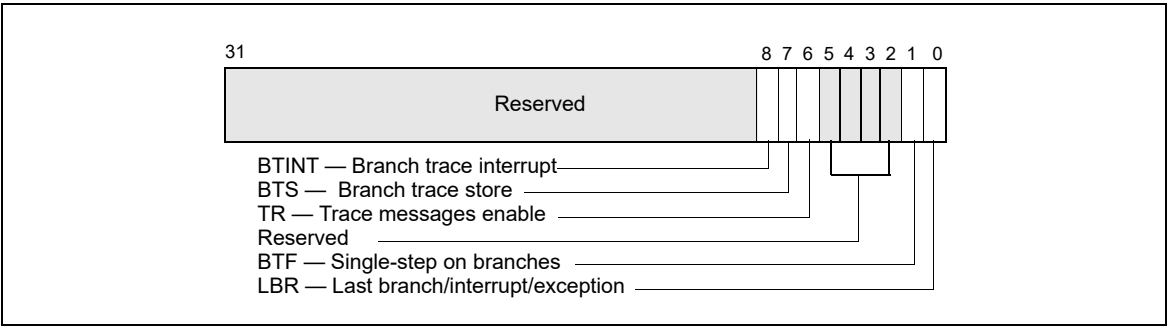


Figure 19-14. IA32_DEBUGCTL MSR for Intel® Core™ Solo and Intel® Core™ Duo Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 19.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address (MSR addresses start at 40H). See Figure 19-15.

- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Intel Core Solo and Intel Core Duo processors, this MSR is located at register address 01C9H.

For compatibility, the Intel Core Solo and Intel Core Duo processors provide two 32-bit MSRs (the MSR_LER_TO_LIP and the MSR_LER_FROM_LIP MSRs) that duplicate functions of the LastExceptionToIP and LastExceptionFromIP MSRs found in P6 family processors.

For details, see Section 19.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture,” and Section 2.20, “MSRs In Intel® Core™ Solo and Intel® Core™ Duo Processors,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

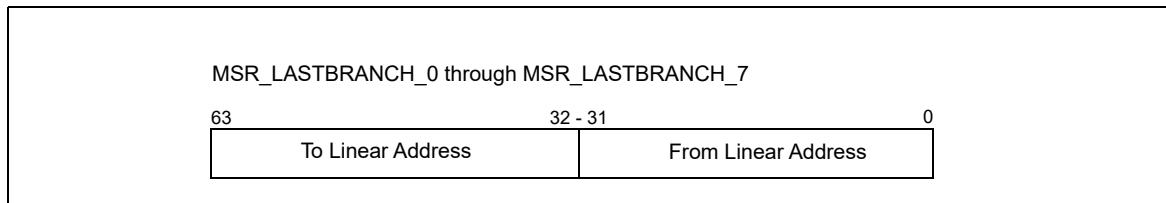


Figure 19-15. LBR Branch Record Layout for the Intel® Core™ Solo and Intel® Core™ Duo Processor

19.15 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (PENTIUM M PROCESSORS)

Like the Pentium 4 and Intel Xeon processor family, Pentium M processors provide last branch interrupt and exception recording. The capability operates almost identically to that found in Pentium 4 and Intel Xeon processors. There are differences in the shape of the stack and in some MSR names and locations. Note the following:

- **MSR_DEBUGCTLB MSR** — Enables debug trace interrupt, debug trace store, trace messages enable, performance monitoring breakpoint flags, single stepping on branches, and last branch. For Pentium M processors, this MSR is located at register address 01D9H. See Figure 19-16 and the entries below for a description of the flags.
 - **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records a running trace of the most recent branches, interrupts, and/or exceptions taken by the processor (prior to a debug exception being generated) in the last branch record (LBR) stack. For more information, see the “Last Branch Record (LBR) Stack” bullet below.
 - **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag rather than a “single-step on instructions” flag. This mechanism allows single-stepping the processor on taken branches. See Section 19.4.3, “Single-Stepping on Branches,” for more information about the BTF flag.
 - **PBi (performance monitoring/breakpoint pins) flags (bits 5-2)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
 - **TR (trace message enable) flag (bit 6)** — When set, branch trace messages are enabled. When the processor detects a taken branch, interrupt, or exception, it sends the branch record out on the system bus as a branch trace message (BTM). See Section 19.4.4, “Branch Trace Messages,” for more information about the TR flag.
 - **BTS (branch trace store) flag (bit 7)** — When set, enables the BTS facilities to log BTMs to a memory-resident BTS buffer that is part of the DS save area. See Section 19.4.9, “BTS and DS Save Area.”
 - **BTINT (branch trace interrupt) flag (bits 8)** — When set, the BTS facilities generate an interrupt when the BTS buffer is full. When clear, BTMs are logged to the BTS buffer in a circular fashion. See Section 19.4.5, “Branch Trace Store (BTS),” for a description of this mechanism.

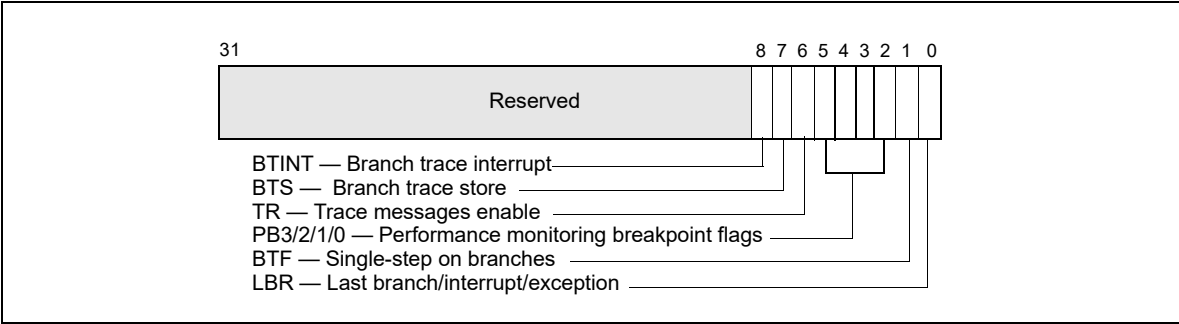


Figure 19-16. MSR_DEBUGCTLB MSR for Pentium M Processors

- **Debug store (DS) feature flag (bit 21), returned by the CPUID instruction** — Indicates that the processor provides the debug store (DS) mechanism, which allows BTMs to be stored in a memory-resident BTS buffer. See Section 19.4.5, “Branch Trace Store (BTS).”
- **Last Branch Record (LBR) Stack** — The LBR stack consists of 8 MSRs (MSR_LASTBRANCH_0 through MSR_LASTBRANCH_7); bits 31-0 hold the ‘from’ address, bits 63-32 hold the ‘to’ address. For Pentium M Processors, these pairs are located at register addresses 040H-047H. See Figure 19-17.
- **Last Branch Record Top-of-Stack (TOS) Pointer** — The TOS Pointer MSR contains a 3-bit pointer (bits 2-0) to the MSR in the LBR stack that contains the most recent branch, interrupt, or exception recorded. For Pentium M Processors, this MSR is located at register address 01C9H.

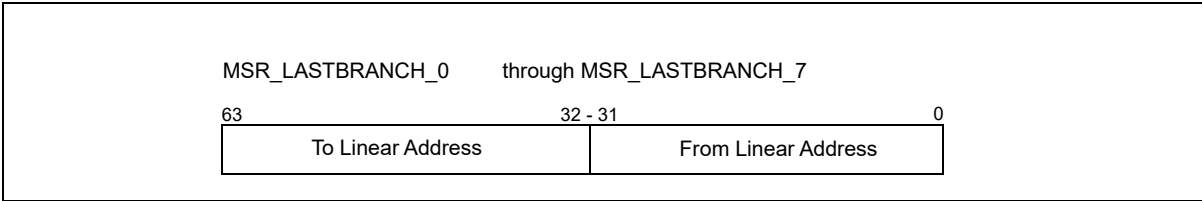


Figure 19-17. LBR Branch Record Layout for the Pentium M Processor

For more detail on these capabilities, see Section 19.13.3, “Last Exception Records,” and Section 2.21, “MSRs In the Pentium M Processor,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

19.16 LAST BRANCH, INTERRUPT, AND EXCEPTION RECORDING (P6 FAMILY PROCESSORS)

The P6 family processors provide five MSRs for recording the last branch, interrupt, or exception taken by the processor: DEBUGCTLMR, LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP. These registers can be used to collect last branch records, to set breakpoints on branches, interrupts, and exceptions, and to single-step from one branch to the next.

See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a detailed description of each of the last branch recording MSRs.

19.16.1 DEBUGCTLMR Register

The version of the DEBUGCTLMR register found in the P6 family processors enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.

A protected-mode operating system procedure is required to provide user access to this register. Figure 19-18 shows the flags in the DEBUGCTLMSR register for the P6 family processors. The functions of these flags are as follows:

- **LBR (last branch/interrupt/exception) flag (bit 0)** — When set, the processor records the source and target addresses (in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs) for the last branch and the last exception or interrupt taken by the processor prior to a debug exception being generated. The processor clears this flag whenever a debug exception, such as an instruction or data breakpoint or single-step trap occurs.

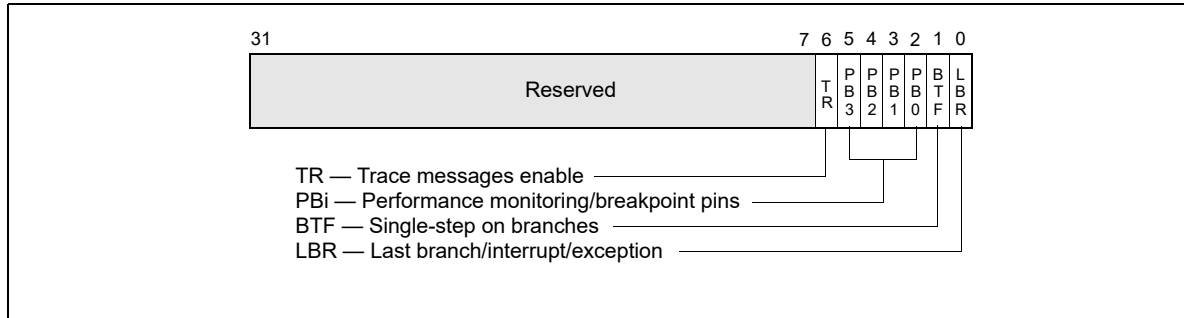


Figure 19-18. DEBUGCTLMSR Register (P6 Family Processors)

- **BTF (single-step on branches) flag (bit 1)** — When set, the processor treats the TF flag in the EFLAGS register as a “single-step on branches” flag. See Section 19.4.3, “Single-Stepping on Branches.”
- **PBi (performance monitoring/breakpoint pins) flags (bits 2 through 5)** — When these flags are set, the performance monitoring/breakpoint pins on the processor (BP0#, BP1#, BP2#, and BP3#) report breakpoint matches in the corresponding breakpoint-address registers (DR0 through DR3). The processor asserts then deasserts the corresponding BPi# pin when a breakpoint match occurs. When a PBi flag is clear, the performance monitoring/breakpoint pins report performance events. Processor execution is not affected by reporting performance events.
- **TR (trace message enable) flag (bit 6)** — When set, trace messages are enabled as described in Section 19.4.4, “Branch Trace Messages.” Setting this flag greatly reduces the performance of the processor. When trace messages are enabled, the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are undefined.

19.16.2 Last Branch and Last Exception MSRs

The LastBranchToIP and LastBranchFromIP MSRs are 32-bit registers for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated. When a branch occurs, the processor loads the address of the branch instruction into the LastBranchFromIP MSR and loads the target address for the branch into the LastBranchToIP MSR.

When an interrupt or exception occurs (other than a debug exception), the address of the instruction that was interrupted by the exception or interrupt is loaded into the LastBranchFromIP MSR and the address of the exception or interrupt handler that is called is loaded into the LastBranchToIP MSR.

The LastExceptionToIP and LastExceptionFromIP MSRs (also 32-bit registers) record the instruction pointers for the last branch that the processor took prior to an exception or interrupt being generated. When an exception or interrupt occurs, the contents of the LastBranchToIP and LastBranchFromIP MSRs are copied into these registers before the to and from addresses of the exception or interrupt are recorded in the LastBranchToIP and LastBranchFromIP MSRs.

These registers can be read using the RDMSR instruction.

Note that the values stored in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into the current code segment, as opposed to linear addresses, which are saved in last branch records for the Pentium 4 and Intel Xeon processors.

19.16.3 Monitoring Branches, Exceptions, and Interrupts

When the LBR flag in the DEBUGCTLMSR register is set, the processor automatically begins recording branches that it takes, exceptions that are generated (except for debug exceptions), and interrupts that are serviced. Each time a branch, exception, or interrupt occurs, the processor records the to and from instruction pointers in the LastBranchToIP and LastBranchFromIP MSRs. In addition, for interrupts and exceptions, the processor copies the contents of the LastBranchToIP and LastBranchFromIP MSRs into the LastExceptionToIP and LastExceptionFromIP MSRs prior to recording the to and from addresses of the interrupt or exception.

When the processor generates a debug exception (#DB), it automatically clears the LBR flag before executing the exception handler, but does not touch the last branch and last exception MSRs. The addresses for the last branch, interrupt, or exception taken are thus retained in the LastBranchToIP and LastBranchFromIP MSRs and the addresses of the last branch prior to an interrupt or exception are retained in the LastExceptionToIP, and LastExceptionFromIP MSRs.

The debugger can use the last branch, interrupt, and/or exception addresses in combination with code-segment selectors retrieved from the stack to reset breakpoints in the breakpoint-address registers (DR0 through DR3), allowing a backward trace from the manifestation of a particular bug toward its source. Because the instruction pointers recorded in the LastBranchToIP, LastBranchFromIP, LastExceptionToIP, and LastExceptionFromIP MSRs are offsets into a code segment, software must determine the segment base address of the code segment associated with the control transfer to calculate the linear address to be placed in the breakpoint-address registers. The segment base address can be determined by reading the segment selector for the code segment from the stack and using it to locate the segment descriptor for the segment in the GDT or LDT. The segment base address can then be read from the segment descriptor.

Before resuming program execution from a debug-exception handler, the handler must set the LBR flag again to re-enable last branch and last exception/interrupt recording.

19.17 TIME-STAMP COUNTER

The Intel 64 and IA-32 architectures (beginning with the Pentium processor) define a time-stamp counter mechanism that can be used to monitor and identify the relative time occurrence of processor events. The counter's architecture includes the following components:

- **TSC flag** — A feature bit that indicates the availability of the time-stamp counter. The counter is available in an if the function CPUID.01H:EDX.TSC[4] = 1.
- **IA32_TIME_STAMP_COUNTER MSR** (called TSC MSR in P6 family and Pentium processors) — The MSR used as the counter.
- **RDTSC instruction** — An instruction used to read the time-stamp counter.
- **TSD flag** — A control register flag is used to enable or disable the time-stamp counter (enabled if CR4.TSD[bit 2] = 1).

The time-stamp counter (as implemented in the P6 family, Pentium, Pentium M, Pentium 4, Intel Xeon, Intel Core Solo and Intel Core Duo processors and later processors) is a 64-bit counter that is set to 0 following a RESET of the processor. Following a RESET, the counter increments even when the processor is halted by the HLT instruction or the external STPCLK# pin. Note that the assertion of the external DPSLP# pin may cause the time-stamp counter to stop.

Processor families increment the time-stamp counter differently:

- For Pentium M processors (family [06H], models [09H, 0DH]); for Pentium 4 processors, Intel Xeon processors (family [0FH], models [00H, 01H, or 02H]); and for P6 family processors: the time-stamp counter increments with every internal processor clock cycle.

The internal processor clock cycle is determined by the current core-clock to bus-clock ratio. Intel® SpeedStep® technology transitions may also impact the processor clock.

- For Pentium 4 processors, Intel Xeon processors (family [0FH], models [03H and higher]); for Intel Core Solo and Intel Core Duo processors (family [06H], model [0EH]); for the Intel Xeon processor 5100 series and Intel Core 2 Duo processors (family [06H], model [0FH]); for Intel Core 2 and Intel Xeon processors (family [06H], DisplayModel [17H]); for Intel Atom processors (family [06H], DisplayModel [1CH]): the time-stamp counter increments at a constant rate. That rate may be set by the maximum core-clock to bus-clock ratio of the

processor or may be set by the maximum resolved frequency at which the processor is booted. The maximum resolved frequency may differ from the processor base frequency, see Section 21.7.2 for more detail. On certain processors, the TSC frequency may not be the same as the frequency in the brand string.

The specific processor configuration determines the behavior. Constant TSC behavior ensures that the duration of each clock tick is uniform and supports the use of the TSC as a wall clock timer even if the processor core changes frequency. This is the architectural behavior moving forward.

NOTE

To determine average processor clock frequency, Intel recommends the use of performance monitoring logic to count processor core clocks over the period of time for which the average is required. See Section 21.6.4.5, “Counting Clocks on systems with Intel® Hyper-Threading Technology in Processors Based on Intel NetBurst® Microarchitecture,” and <https://perfmon-events.intel.com/> for more information.

The RDTSC instruction reads the time-stamp counter and is guaranteed to return a monotonically increasing unique value whenever executed, except for a 64-bit counter wraparound. Intel guarantees that the time-stamp counter will not wraparound within 10 years after being reset. The period for counter wrap is longer for Pentium 4, Intel Xeon, P6 family, and Pentium processors.

Normally, the RDTSC instruction can be executed by programs and procedures running at any privilege level and in virtual-8086 mode. The TSD flag allows use of this instruction to be restricted to programs and procedures running at privilege level 0. A secure operating system would set the TSD flag during system initialization to disable user access to the time-stamp counter. An operating system that disables user access to the time-stamp counter should emulate the instruction through a user-accessible programming interface.

The RDTSC instruction is not serializing or ordered with other instructions. It does not necessarily wait until all previous instructions have been executed before reading the counter. Similarly, subsequent instructions may begin execution before the RDTSC instruction operation is performed.

The RDMSR and WRMSR instructions read and write the time-stamp counter, treating the time-stamp counter as an ordinary MSR (address 10H). In the Pentium 4, Intel Xeon, and P6 family processors, all 64-bits of the time-stamp counter are read using RDMSR (just as with RDTSC). When WRMSR is used to write the time-stamp counter on processors before family [0FH], models [03H, 04H]: only the low-order 32-bits of the time-stamp counter can be written (the high-order 32 bits are cleared to 0). For family [0FH], models [03H, 04H, 06H]; for family [06H]], model [0EH, 0FH]; for family [06H]], DisplayModel [17H, 1AH, 1CH, 1DH]: all 64 bits are writable.

19.17.1 Invariant TSC

The time stamp counter in newer processors may support an enhancement, referred to as invariant TSC. Processor's support for invariant TSC is indicated by CPUID.80000007H:EDX[8].

The invariant TSC will run at a constant rate in all ACPI P-, C-, and T-states. This is the architectural behavior moving forward. On processors with invariant TSC support, the OS may use the TSC for wall clock timer services (instead of ACPI or HPET timers). TSC reads are much more efficient and do not incur the overhead associated with a ring transition or access to a platform resource.

19.17.2 IA32_TSC_AUX Register and RDTSCP Support

Processors based on Nehalem microarchitecture provide an auxiliary TSC register, IA32_TSC_AUX that is designed to be used in conjunction with IA32_TSC. IA32_TSC_AUX provides a 32-bit field that is initialized by privileged software with a signature value (for example, a logical processor ID).

The primary usage of IA32_TSC_AUX in conjunction with IA32_TSC is to allow software to read the 64-bit time stamp in IA32_TSC and signature value in IA32_TSC_AUX with the instruction RDTSCP in an atomic operation. RDTSCP returns the 64-bit time stamp in EDX:EAX and the 32-bit TSC_AUX signature value in ECX. The atomicity of RDTSCP ensures that no context switch can occur between the reads of the TSC and TSC_AUX values.

Support for RDTSCP is indicated by CPUID.80000001H:EDX[27]. As with RDTSC instruction, non-ring 0 access is controlled by CR4.TSD (Time Stamp Disable flag).

User mode software can use RDTSCP to detect if CPU migration has occurred between successive reads of the TSC. It can also be used to adjust for per-CPU differences in TSC values in a NUMA system.

19.17.3 Time-Stamp Counter Adjustment

Software can modify the value of the time-stamp counter (TSC) of a logical processor by using the WRMSR instruction to write to the IA32_TIME_STAMP_COUNTER MSR (address 10H). Because such a write applies only to that logical processor, software seeking to synchronize the TSC values of multiple logical processors must perform these writes on each logical processor. It may be difficult for software to do this in a way that ensures that all logical processors will have the same value for the TSC at a given point in time.

The synchronization of TSC adjustment can be simplified by using the 64-bit IA32_TSC_ADJUST MSR (address 3BH). Like the IA32_TIME_STAMP_COUNTER MSR, the IA32_TSC_ADJUST MSR is maintained separately for each logical processor. A logical processor maintains and uses the IA32_TSC_ADJUST MSR as follows:

- On RESET, the value of the IA32_TSC_ADJUST MSR is 0.
- If an execution of WRMSR to the IA32_TIME_STAMP_COUNTER MSR adds (or subtracts) value X from the TSC, the logical processor also adds (or subtracts) value X from the IA32_TSC_ADJUST MSR.
- If an execution of WRMSR to the IA32_TSC_ADJUST MSR adds (or subtracts) value X from that MSR, the logical processor also adds (or subtracts) value X from the TSC.

Unlike the TSC, the value of the IA32_TSC_ADJUST MSR changes only in response to WRMSR (either to the MSR itself, or to the IA32_TIME_STAMP_COUNTER MSR). Its value does not otherwise change as time elapses. Software seeking to adjust the TSC can do so by using WRMSR to write the same value to the IA32_TSC_ADJUST MSR on each logical processor.

Processor support for the IA32_TSC_ADJUST MSR is indicated by CPUID.07H.00H:EBX.TSC_ADJUST (bit 1).

19.17.4 Invariant Time-Keeping

The invariant TSC is based on the invariant timekeeping hardware (called Always Running Timer or ART), that runs at the core crystal clock frequency. The ratio defined by CPUID.15H expresses the frequency relationship between the ART hardware and TSC.

If CPUID.15H:EBX[31:0] != 0 and CPUID.80000007H:EDX.TSC_INVARIANT = 1, the following linearity relationship holds between TSC and the ART hardware:

$$\text{TSC\_Value} = (\text{ART\_Value} * \text{CPUID.15H:EBX}[31:0]) / \text{CPUID.15H:EAX}[31:0] + K$$

Where 'K' is an offset that can be adjusted by a privileged agent¹.

When ART hardware is reset, both invariant TSC and K are also reset.

19.18 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) MONITORING FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of monitoring capabilities including Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring (MBM). The Intel® Xeon® processor E5 v3 family introduced resource monitoring capability in each logical processor to measure specific platform shared resource metrics, for example, L3 cache occupancy. The programming interface for these monitoring features is described in this section. Two features within the monitoring feature set provided are described - Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.

Cache Monitoring Technology (CMT) allows an Operating System, Hypervisor or similar system management agent to determine the usage of cache by applications running on the platform. The initial implementation is directed at L3 cache monitoring (currently the last level cache in most server platforms).

1. IA32_TSC_ADJUST MSR and the TSC-offset field in the VM execution controls of VMCS are some of the common interfaces that privileged software can use to manage the time stamp counter for keeping time

Memory Bandwidth Monitoring (MBM), introduced in the Intel® Xeon® processor E5 v4 family, builds on the CMT infrastructure to allow monitoring of bandwidth from one level of the cache hierarchy to the next - in this case focusing on the L3 cache, which is typically backed directly by system memory. As a result of this implementation, memory bandwidth can be monitored.

The monitoring mechanisms described provide the following key shared infrastructure features:

- A mechanism to enumerate the presence of the monitoring capabilities within the platform (via a CPUID feature bit).
- A framework to enumerate the details of each sub-feature (including CMT and MBM, as discussed later, via CPUID leaves and sub-leaves).
- A mechanism for the OS or Hypervisor to indicate a software-defined ID for each of the software threads (applications, virtual machines, etc.) that are scheduled to run on a logical processor. These identifiers are known as Resource Monitoring IDs (RMIDs).
- Mechanisms in hardware to monitor cache occupancy and bandwidth statistics as applicable to a given product generation on a per software-id basis.
- Mechanisms for the OS or Hypervisor to read back the collected metrics such as L3 occupancy or Memory Bandwidth for a given software ID at any point during runtime.

19.18.1 Overview of Cache Monitoring Technology and Memory Bandwidth Monitoring

The shared resource monitoring features described in this chapter provide a layer of abstraction between applications and logical processors through the use of **Resource Monitoring IDs** (RMIDs). Each logical processor in the system can be assigned an RMID independently, or multiple logical processors can be assigned to the same RMID value (e.g., to track an application with multiple threads). For each logical processor, only one RMID value is active at a time. This is enforced by the IA32_PQR_ASSOC MSR, which specifies the active RMID of a logical processor. Writing to this MSR by software changes the active RMID of the logical processor from an old value to a new value.

The underlying platform shared resource monitoring hardware tracks cache metrics such as cache utilization and misses as a result of memory accesses according to the RMIDs and reports monitored data via a counter register (IA32_QM_CTR). The specific event types supported vary by generation and can be enumerated via CPUID. To read back monitored data, software configures an event selection MSR (IA32_QM_EVTSEL) to specify which metric is to be reported and the specific RMID for which the data should be returned.

Processor support of the monitoring framework and sub-features such as CMT is reported via the CPUID instruction. The resource type available to the monitoring framework is enumerated via a new leaf function in CPUID. Reading and writing to the monitoring MSRs requires the RDMSR and WRMSR instructions.

The Cache Monitoring Technology feature set provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the CMT feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- CMT-specific event codes to read occupancy for a given level of the cache hierarchy.

The Memory Bandwidth Monitoring feature provides the following unique mechanisms:

- A mechanism to enumerate the presence and details of the MBM feature as applicable to a given level of the cache hierarchy, independent of other monitoring features.
- MBM-specific event codes to read bandwidth out to the next level of the hierarchy and various sub-event codes to read more specific metrics as discussed later (e.g., total bandwidth vs. bandwidth only from local memory controllers on the same package).

19.18.2 Enabling Monitoring: Usage Flow

Figure 19-19 illustrates the key steps for OS/VMM to detect support of shared resource monitoring features such as CMT and enable resource monitoring for available resource types and monitoring events.

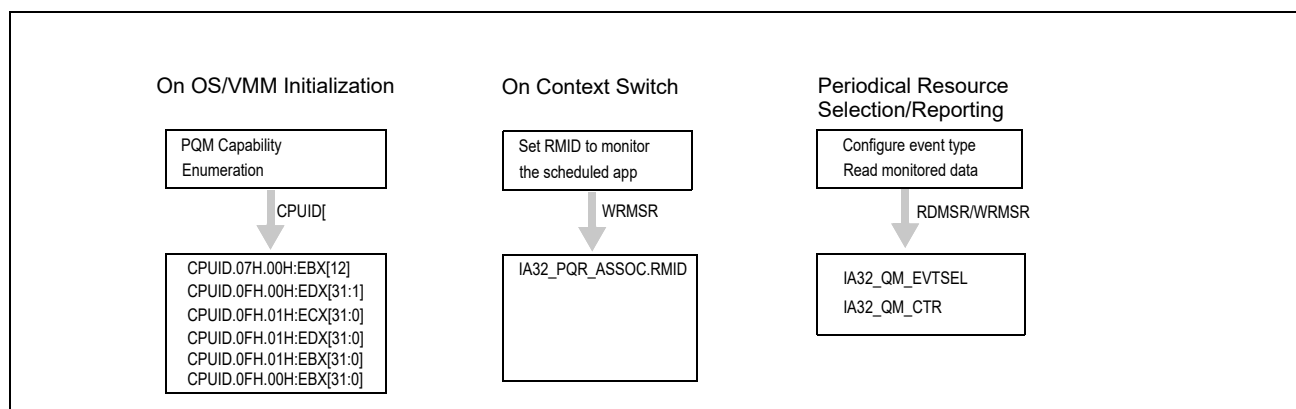


Figure 19-19. Platform Shared Resource Monitoring Usage Flow

19.18.3 Enumeration and Detecting Support of Cache Monitoring Technology and Memory Bandwidth Monitoring

Software can query processor support of shared resource monitoring features capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_M[12] reports 1, the processor provides the following programming interfaces for shared resource monitoring, including Cache Monitoring Technology:

- CPUID.0FH (Shared Resource Monitoring Enumeration leaf) provides information on available resource types (see Section 19.18.4), and monitoring capabilities for each resource type (see Section 19.18.5). Note CMT and MBM capabilities are enumerated as separate event vectors using shared enumeration infrastructure under a given resource type.
- IA32_PQR_ASSOC.RMID: The per-logical-processor MSR, IA32_PQR_ASSOC, that OS/VMM can use to assign an RMID to each logical processor, see Section 19.18.6.
- IA32_QM_EVTSEL: This MSR specifies an Event ID (EvtID) and an RMID which the platform uses to look up and provide monitoring data in the monitoring counter, IA32_QM_CTR, see Section 19.18.7.
- IA32_QM_CTR: This MSR reports monitored resource data when available along with bits to allow software to check for error conditions and verify data validity.

Software must follow the following sequence of enumeration to discover Cache Monitoring Technology capabilities:

1. Use CPUID.00H to discover the "cpuid_maxLeaf" supported in the processor;
2. If cpuid_maxLeaf >= 7, then verify CPUID.07H.00H:EBX.RDT_M[12] is set;
3. If CPUID.07H.00H:EBX.RDT_M[12] = 1, then use CPUID.0FH.00H to query available resource types that support monitoring;
4. If CPUID.0FH.00H:EDX.CMT_L3[1] = 1, then use CPUID.0FH.01H to query the specific capabilities of L3 Cache Monitoring Technology (CMT) and Memory Bandwidth Monitoring.
5. If CPUID.0FH.00H:EDX reports additional resource types supporting monitoring, then use CPUID.0FH.ResID for a corresponding resource type ID (ResID) as enumerated by the bit position of CPUID.0FH.00H:EDX.

19.18.4 Monitoring Resource Type and Capability Enumeration

CPUID.0FH (Shared Resource Monitoring Enumeration leaf) provides one sub-leaf (sub-function 0) that reports shared enumeration infrastructure, and one or more sub-functions that report feature-specific enumeration data:

- Monitoring leaf sub-function 0 enumerates available resources that support monitoring, i.e., use CPUID.0FH.00H. In the initial implementation, L3 cache is the only resource type available. Each supported resource type is represented by a bit in CPUID.0FH.00H:EDX[31:1]. The bit position corresponds to the sub-leaf index (ResID) that software must use to query details of the monitoring capability of that resource type (see Figure 19-21 and Figure 19-22). Reserved bits of CPUID.0FH.00H:EDX[31:2] correspond to unsupported

sub-leaves of the CPUID.0FH leaf. Additionally, CPUID.0FH.00H:EBX reports the highest RMID value of any resource type that supports monitoring in the processor.

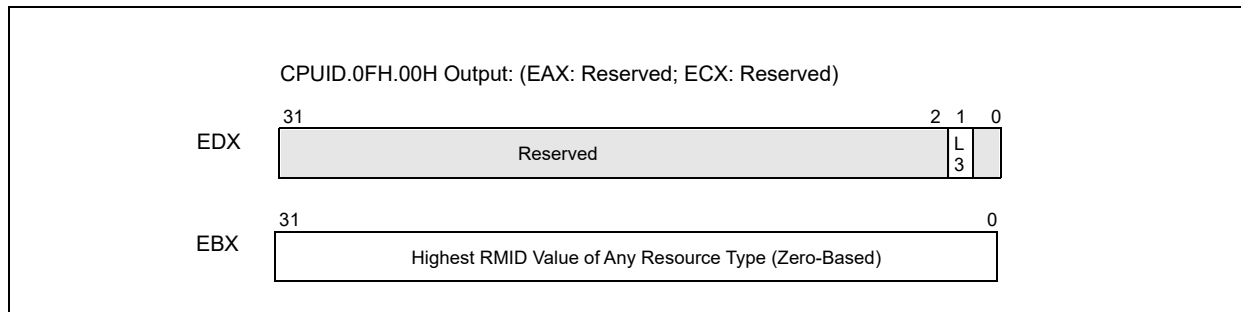


Figure 19-20. CPUID.0FH.00H Monitoring Resource Type Enumeration

19.18.5 Feature-Specific Enumeration

Each additional sub-leaf of CPUID.0FH.ResID enumerates the specific details for software to program monitoring MSRs using the resource type associated with the given ResID.

Note that in future Monitoring implementations the meanings of the returned registers may vary in other sub-leaves that are not yet defined. The registers will be specified and defined on a per-ResID basis.

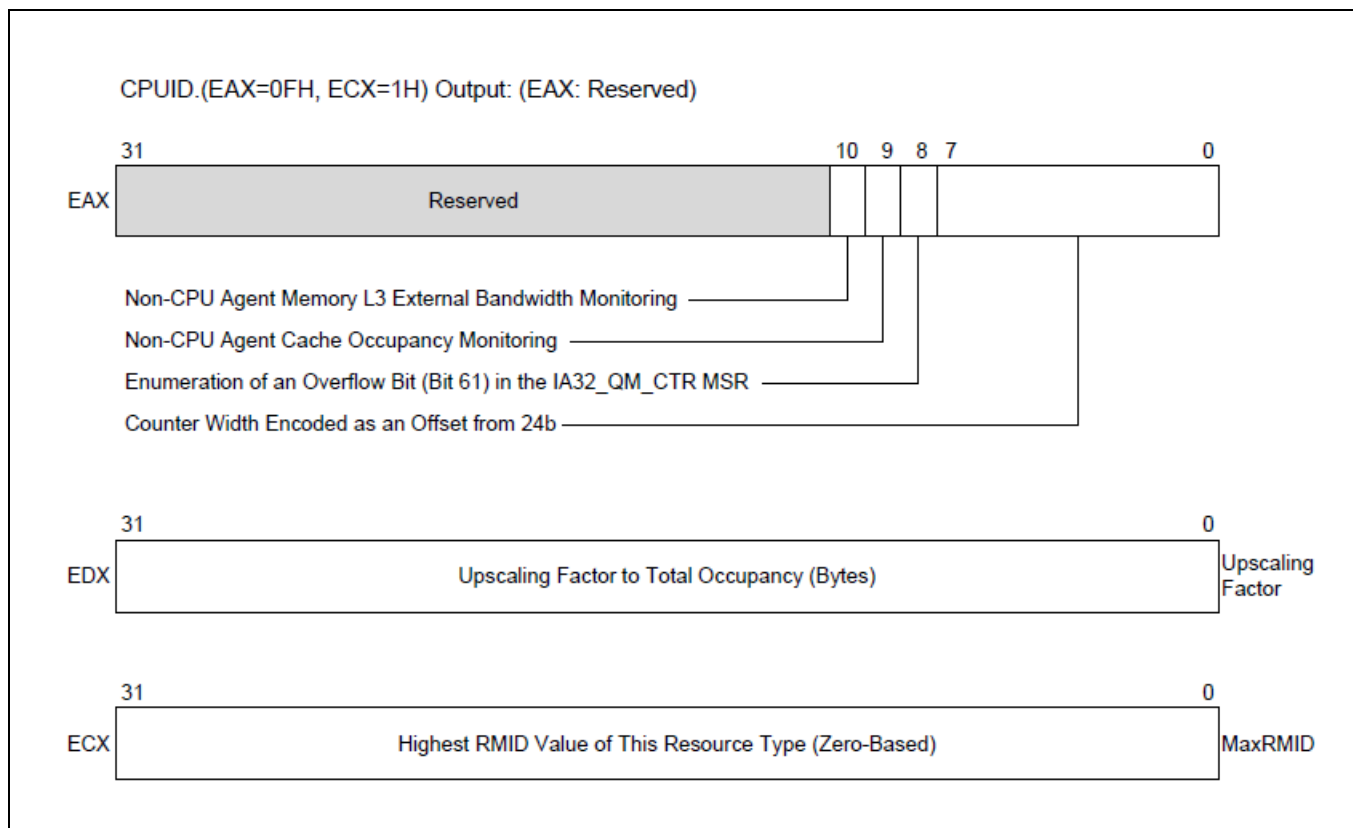


Figure 19-21. L3 Cache Monitoring Capability Enumeration Data (CPUID.0FH.01H)

CPUID.0FH.01H:EAX[7:0]: Encode counter width as offset from 24b. See Section 19.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[8]: If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR. See Section 19.18.5.2 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[9]: If 1, indicates the presence of non-CPU agent Intel RDT CMT support. See Section 19.20 for details. Bits 31:11 of EAX are reserved.

CPUID.0FH.01H:EAX[10]: If 1, indicates the presence of non-CPU agent Intel RDT MBM support. See Section 19.20 for details. Bits 31:11 of EAX are reserved.

For each supported Cache Monitoring resource type, hardware supports only a finite number of RMIDs. CPUID.0FH.01H:ECX enumerates the highest RMID value that can be monitored with this resource type, see Figure 19-21.

CPUID.0FH.01H:EDX specifies a bit vector that is used to look up the EventID (See Figure 19-22 and Table 19-18) that software must program with IA32_QM_EVTSEL in order to retrieve event data. After software configures IA32_QMEVTSEL with the desired RMID and EventID, it can read the resulting data from IA32_QM_CTR. The raw numerical value reported from IA32_QM_CTR can be converted to the final value (occupancy in bytes or bandwidth in bytes per sampled time period) by multiplying the counter value by the value from CPUID.0FH.01H:EBX, see Figure 19-21.

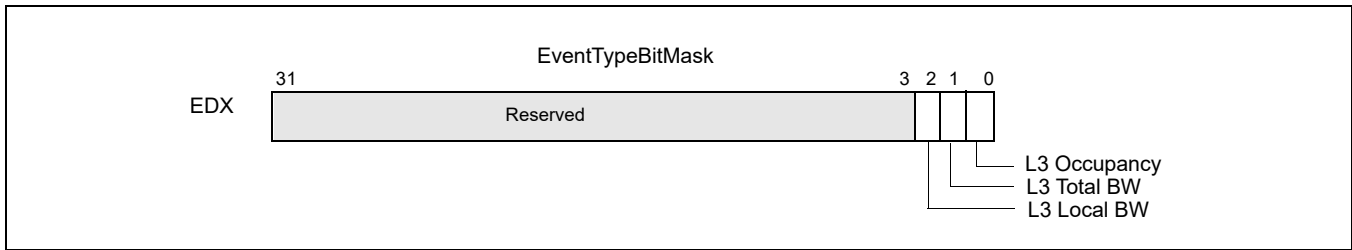


Figure 19-22. L3 Cache Monitoring Capability Enumeration Event Type Bit Vector (CPUID.0FH.01H)

19.18.5.1 Cache Monitoring Technology

On processors for which Cache Monitoring Technology supports the L3 cache occupancy event, CPUID.0FH.01H:EDX returns with bit 0 set. The corresponding event ID is shown in Table 19-18. The L3 occupancy data accumulated in the IA32_QM_CTR MSR can be converted to total occupancy (in bytes) by multiplying with CPUID.0FH.01H:EBX.

Event codes for Cache Monitoring Technology are discussed in the next section.

19.18.5.2 Memory Bandwidth Monitoring

On processors that monitoring supports Memory Bandwidth Monitoring using ResID=1 (L3), two additional bits are defined in the vector at CPUID.0FH.01H:EDX:

- CPUID.0FH.01H:EDX[1]: indicates the L3 total external bandwidth monitoring event is supported if set. This event monitors the L3 total external bandwidth to the next level of the cache hierarchy, including all demand and prefetch misses from the L3 to the next hierarchy of the memory system. In most platforms, this represents memory bandwidth.
- CPUID.0FH.01H:EDX[2]: indicates L3 local memory bandwidth monitoring event is supported if set. This event monitors the L3 external bandwidth satisfied by the local memory. In most platforms that support this event, L3 requests are likely serviced by a memory system with non-uniform memory architecture. This allows bandwidth to off-package memory resources to be tracked by subtracting local from total bandwidth (for instance, bandwidth over QPI to a memory controller on another physical processor could be tracked by subtraction). Note that it is not possible to read the local and total bandwidth atomically; multiple operations are needed. Because of this, it is possible for the counters to change in between the two reads.

The corresponding Event ID is shown in Table 19-18. The L3 bandwidth data accumulated in IA32_QM_CTR can be converted to total bandwidth (in bytes) using CPUID.0FH.01H:EBX.

Table 19-18. Monitoring Supported Event IDs

Event Type	Event ID	Context
L3 Cache Occupancy	01H	Cache Monitoring Technology
L3 Total External Bandwidth	02H	MBM
L3 Local External Bandwidth	03H	MBM
Reserved	All other event codes	N/A

A field is added to CPUID to enumerate the MBM counter width in platforms that support the extensible MBM counter width feature.

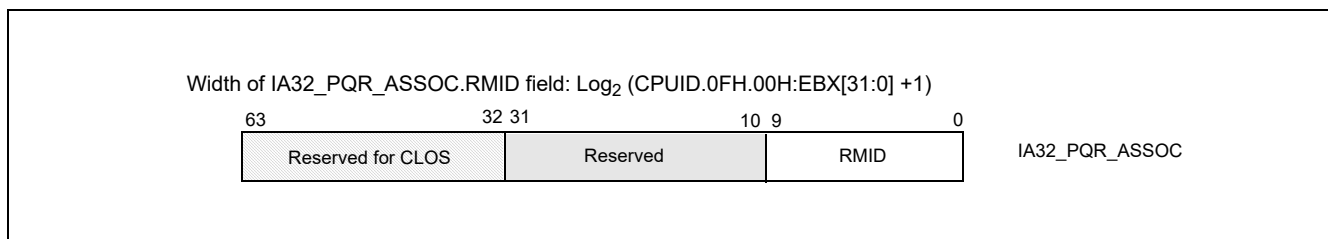
- CPUID.0FH.01H:EAX[7:0]: Encode counter width as offset from 24b in bits[7:0]. In EAX bits 7:0, the counter width is encoded as an offset from 24b. A value of zero in this field means 24-bit counters are supported. A value of 8 indicates that 32-bit counters are supported, as in the 3rd generation Intel Xeon Scalable Processor Family. With this enumerable counter width, the requirement that software polls at 1Hz is removed. Software may poll at a varying rate with a reduced risk of rollover. Under typical conditions, rollover will likely require hundreds of seconds (though this value is not explicitly specified and may vary and decrease in future processor generations as memory bandwidths increase). Suppose software seeks to ensure that rollover does not occur more than once between samples. In that case, sampling at 1Hz while consuming the enumerated counter widths' worth of data will provide this guarantee for a specific platform and counter width under all conditions.
- CPUID.0FH.01H:EAX[8]: Enumeration of the presence of an overflow bit in the IA32_QM_CTR MSR via EAX bit[8]. Software that uses the MBM event retrieval MSR interface should be updated to comprehend this new format, which enables up to 62-bit MBM counters to be provided by future platforms. Higher-level software that consumes the resulting bandwidth values is not expected to be affected. An overflow bit is defined in the IA32_QM_CTR MSR, bit 61, if CPUID.0FH.01H:EAX[8] is set. This rollover bit will be set on overflow of the MBM counters and reset upon read. Current processors do not support this capability.

19.18.6 Monitoring Resource RMID Association

After Monitoring and sub-features have been enumerated, software can begin using the monitoring features. The first step is to associate a given software thread (or multiple threads as part of an application, VM, group of applications or other abstraction) with an RMID.

Note that the process of associating an RMID with a given software thread is the same for all shared resource monitoring features (CMT, MBM), and a given RMID number has the same meaning from the viewpoint of any logical processors in a package. Stated another way, a thread may be associated in a 1:1 mapping with an RMID, and that RMID may allow cache occupancy, memory bandwidth information or other monitoring data to be read back later with monitoring event codes (retrieving data is discussed in a previous section).

The association of an application thread with an RMID requires an OS to program the per-logical-processor MSR IA32_PQR_ASSOC at context swap time (updates may also be made at any other arbitrary points during program execution such as application phase changes). The IA32_PQR_ASSOC MSR specifies the active RMID that monitoring hardware will use to tag internal operations, such as L3 cache requests. The layout of the MSR is shown in Figure 19-23. Software specifies the active RMID to monitor in the IA32_PQR_ASSOC.RMID field. The width of the RMID field can vary from one implementation to another, and is derived from $\text{Ceil}(\text{LOG}_2(1 + \text{CPUID.0FH.00H:EBX}[31:0]))$. The value of IA32_PQR_ASSOC after power-on is 0.

**Figure 19-23. IA32_PQR_ASSOC MSR**

In the initial implementation, the width of the RMID field is up to 10 bits wide, zero-referenced and fully encoded. However, software must use CPUID to query the maximum RMID supported by the processor. If a value larger than the maximum RMID is written to IA32_PQR_ASSOC.RMID, a #GP(0) fault will be generated.

RMIDs have a global scope within the physical package- if an RMID is assigned to one logical processor then the same RMID can be used to read multiple thread attributes later (for example, L3 cache occupancy or external bandwidth from the L3 to the next level of the cache hierarchy). In a multiple LLC platform the RMIDs are to be reassigned by the OS or VMM scheduler when an application is migrated across LLCs.

Note that in a situation where Monitoring supports multiple resource types, some upper range of RMIDs (e.g., RMID 31) may only be supported by one resource type but not by another resource type.

19.18.7 Monitoring Resource Selection and Reporting Infrastructure

The reporting mechanism for Cache Monitoring Technology and other related features is architecturally exposed as an MSR pair that can be programmed and read to measure various metrics such as the L3 cache occupancy (CMT) and bandwidths (MBM) depending on the level of Monitoring support provided by the platform. Data is reported back on a per-RMID basis. These events do not trigger based on event counts or trigger APIC interrupts (e.g., no Performance Monitoring Interrupt occurs based on counts). Rather, they are used to sample counts explicitly.

The MSR pair for the shared resource monitoring features (CMT, MBM) is separate from and not shared with architectural PerfMon counters, meaning software can use these monitoring features simultaneously with the PerfMon counters.

Access to the aggregated monitoring information is accomplished through the following programmable monitoring MSRs:

- **IA32_QM_EVTSEL:** This MSR provides a role similar to the event select MSRs for programmable performance monitoring described in Chapter 18. The simplified layout of the MSR is shown in Figure 19-24. IA32_QM_EVTSEL.EvtID (bits 7:0) specifies an event code of a supported resource type for hardware to report monitored data associated with IA32_QM_EVTSEL.RMID (bits 41:32). Software can configure IA32_QM_EVTSEL.RMID with any RMID that is active within the physical processor. The width of IA32_QM_EVTSEL.RMID matches that of IA32_PQR_ASSOC.RMID. Supported event codes for the IA32_QM_EVTSEL register are shown in Table 19-18. Note that valid event codes may not necessarily map directly to the bit position used to enumerate support for the resource via CPUID.

Software can program an RMID / Event ID pair into the IA32_QM_EVTSEL MSR bit field to select an RMID to read a particular counter for a given resource. The currently supported list of Monitoring Event IDs is discussed in Section 19.18.5, which covers feature-specific details.

Thread access to the IA32_QM_EVTSEL and IA32_QM_CTR MSR pair should be serialized (that is, treated as a critical section under lock) to avoid situations where one thread changes the RMID/EvtID just before another thread reads monitoring data from IA32_QM_CTR.

- **IA32_QM_CTR:** This MSR reports monitored data when available. It contains three bit fields. If software configures an unsupported RMID or event type in IA32_QM_EVTSEL, then IA32_QM_CTR.Error (bit 63) will be set, indicating there is no valid data to report. If IA32_QM_CTR.Unavailable (bit 62) is set, it indicates monitored data for the RMID is not available, and IA32_QM_CTR.data (bits 61:0) should be ignored. Therefore, IA32_QM_CTR.data (bits 61:0) is valid only if bits 63 and 62 are both clear. The IA32_QM_CTR.Overflow (bit 61) is present if CPUID.0FH.01H:EAX[8] is set. This bit is set on overflow of the MBM counters and will reset upon read. For Cache Monitoring Technology, software can convert IA32_QM_CTR.data into cache occupancy or bandwidth metrics expressed in bytes by multiplying with the conversion factor from CPUID.0FH.01H:EBX.

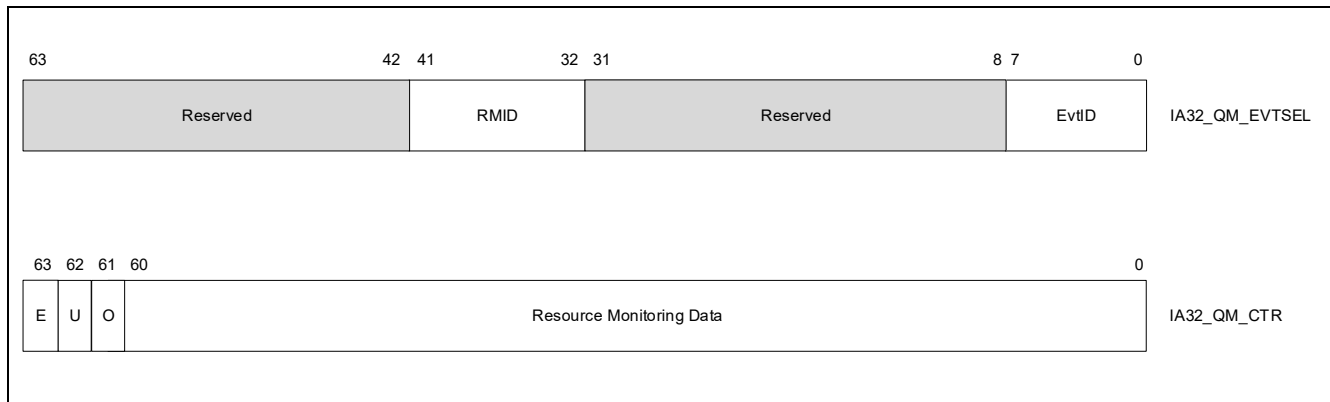


Figure 19-24. IA32_QM_EVTSEL and IA32_QM_CTR MSRs

19.18.8 Monitoring Programming Considerations

Figure 19-25 illustrates how system software can program IA32_QM_EVTSEL and IA32_QM_CTR to perform resource monitoring.

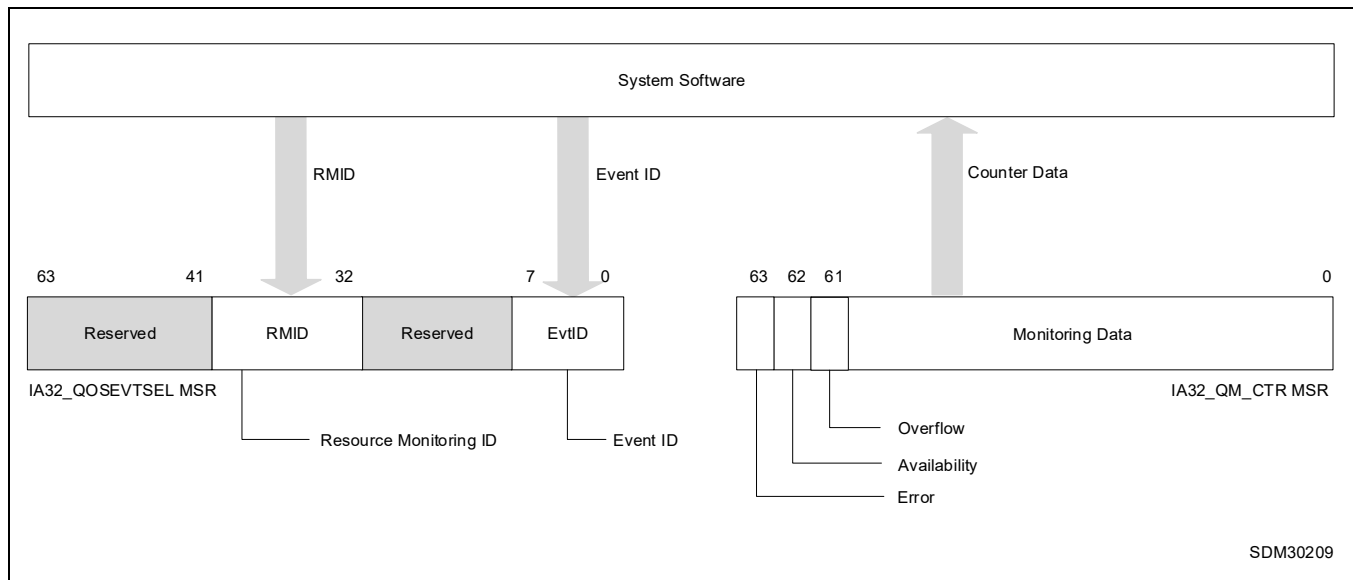


Figure 19-25. Software Usage of Cache Monitoring Resources

Though the field provided in IA32_QM_CTR allows for up to 62 bits of data to be returned, often a subset of bits are used. With Cache Monitoring Technology for instance, the number of bits used is the base-two logarithm of the total cache size divided by the Upscaling Factor from CPUID.

In Memory Bandwidth Monitoring, the initial counter size is 24 bits, and retrieving the value at 1Hz or faster is sufficient to ensure at most one rollover per sampling period. Any changes to counter width are enumerated to software; see Section 19.18.5.2 for details.

19.18.8.1 Monitoring Dynamic Configuration

Both the IA32_QM_EVTSEL and IA32_PQR_ASSOC registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,

- An RMID exceeding the maximum RMID is used.

19.18.8.2 Monitoring Operation With Power Saving Features

Some advanced power management features such as deep package C-states may shrink the L3 cache and cause CMT occupancy count to be reduced. MBM bandwidth counts may increase due to flushing cached data out of L3.

19.18.8.3 Monitoring Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and monitoring counter are unmodified across an SMI delivery. Thus, the execution of SMM handler code and SMM handler's data can manifest as spurious contribution in the monitored data.

It is possible for an SMM handler to minimize the impact on of spurious contribution in the QOS monitoring counters by reserving a dedicated RMID for monitoring the SMM handler. Such an SMM handler can save the previously configured QOS Monitoring state immediately upon entering SMM, and restoring the QOS monitoring state back to the prev-SMM RMID upon exit.

19.18.8.4 Monitoring Operation with RAS Features

In general, the Reliability, Availability, and Serviceability (RAS) features present in Intel Platforms are not expected to significantly affect shared resource monitoring counts. In cases where software RAS features cause memory copies or cache accesses, these may be tracked and may influence the shared resource monitoring counter values.

19.19 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) ALLOCATION FEATURES

The Intel Resource Director Technology (Intel RDT) feature set provides a set of allocation (resource control) capabilities including Cache Allocation Technology (CAT) and Code and Data Prioritization (CDP). The Intel Xeon processor E5 v4 family (and a subset of communication-focused processors in the Intel Xeon E5 v3 family) introduce capabilities to configure and make use of the Cache Allocation Technology (CAT) mechanisms on the L3 cache. Certain Intel Atom processors also provide support for control over the L2 cache, with capabilities as described below. The programming interface for Cache Allocation Technology and for the more general allocation capabilities are described in the rest of this chapter. The CAT and CDP capabilities, where architecturally supported, may be detected and enumerated in software using the CPUID instruction, as described in this chapter.

The Intel Xeon Scalable Processor Family introduces the Memory Bandwidth Allocation (MBA) feature which provides indirect control over the memory bandwidth available to CPU cores, and is discussed later in this chapter.

19.19.1 Introduction to Cache Allocation Technology (CAT)

Cache Allocation Technology enables an Operating System (OS), Hypervisor /Virtual Machine Manager (VMM) or similar system service management agent to specify the amount of cache space into which an application can fill (as a hint to hardware - certain features such as power management may override CAT settings). Specialized user-level implementations with minimal OS support are also possible, though not necessarily recommended (see notes below for OS/Hypervisor with respect to ring 3 software and virtual guests). Depending on the processor family, L2 or L3 cache allocation capability may be provided, and the technology is designed to scale across multiple cache levels and technology generations.

Software can determine which levels are supported in a given platform programmatically using CPUID as described in the following sections.

The CAT mechanisms defined in this document provide the following key features:

- A mechanism to enumerate platform Cache Allocation Technology capabilities and available resource types that provides CAT control capabilities. For implementations that support Cache Allocation Technology, CPUID provides enumeration support to query which levels of the cache hierarchy are supported and specific CAT capabilities, such as the max allocation bitmask size.

- A mechanism for the OS or Hypervisor to configure the amount of a resource available to a particular Class of Service via a list of allocation bitmasks.
- Mechanisms for the OS or Hypervisor to signal the Class of Service to which an application belongs.
- Hardware mechanisms to guide the LLC fill policy when an application has been designated to belong to a specific Class of Service.

Note that for many usages, an OS or Hypervisor may not want to expose Cache Allocation Technology mechanisms to Ring3 software or virtualized guests.

The Cache Allocation Technology feature enables more cache resources (i.e., cache space) to be made available for high priority applications based on guidance from the execution environment as shown in Figure 19-26. The architecture also allows dynamic resource reassignment during runtime to further optimize the performance of the high priority application with minimal degradation to the low priority app. Additionally, resources can be rebalanced for system throughput benefit across uses cases of OSes, VMMs, containers, and other scenarios by managing the CPUID and MSR interfaces. This section describes the hardware and software support required in the platform including what is required of the execution environment (i.e., OS/VMM) to support such resource control. Note that in Figure 19-26 the L3 Cache is shown as an example resource.

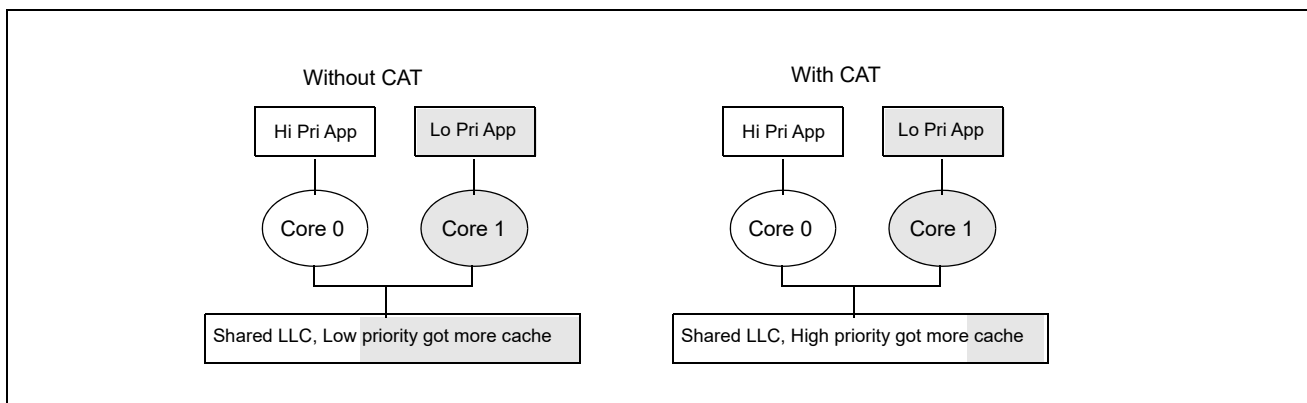


Figure 19-26. Cache Allocation Technology Enables Allocation of More Resources to High Priority Applications

19.19.2 Cache Allocation Technology Architecture

The fundamental goal of Cache Allocation Technology is to enable resource allocation based on application priority or Class of Service (CLOS). The processor exposes a set of Classes of Service into which applications (or individual threads) can be assigned. Cache allocation for the respective applications or threads is then restricted based on the class with which they are associated. Each Class of Service can be configured using capacity bitmasks (CBMs) which represent capacity and indicate the degree of overlap and isolation between classes. For each logical processor there is a register exposed (referred to here as the IA32_PQR_ASSOC MSR or PQR) to allow the OS/VMM to specify a CLOS when an application, thread or VM is scheduled.

The usage of Classes of Service (CLOS) are consistent across resources and a CLOS may have multiple resource control attributes attached, which reduces software overhead at context swap time. Rather than adding new types of CLOS tags per resource for instance, the CLOS management overhead is constant. Cache allocation for the indicated application/thread/container/VM is then controlled automatically by the hardware based on the class and the bitmask associated with that class. Bitmasks are configured via the IA32_resourceType_MASK_n MSRs, where resourceType indicates a resource type (e.g., "L3" for the L3 cache) and "n" indicates a CLOS number.

The basic ingredients of Cache Allocation Technology are as follows:

- An architecturally exposed mechanism using CPUID to indicate whether CAT is supported, and what resource types are available which can be controlled.
- For each available resourceType, CPUID also enumerates the total number of Classes of Services and the length of the capacity bitmasks that can be used to enforce cache allocation to applications on the platform.
- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to configure the behavior of different classes of service using the bitmasks available.

- An architecturally exposed mechanism to allow the execution environment (OS/VMM) to assign a CLOS to an executing software thread (i.e., associating the active CR3 of a logical processor with the CLOS in IA32_PQR_ASSOC).
- Implementation-dependent mechanisms to indicate which CLOS is associated with a memory access and to enforce the cache allocation on a per CLOS basis.

A capacity bitmask (CBM) provides a hint to the hardware indicating the cache space an application should be limited to as well as providing an indication of overlap and isolation in the CAT-capable cache from other applications contending for the cache. The bit length of the capacity mask available generally depends on the configuration of the cache and is specified in the enumeration process for CAT in CPUID (this may vary between models in a processor family as well). Similarly, other parameters such as the number of supported CLOS may vary for each resource type, and these details can be enumerated via CPUID.

	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A	A	A	A	A	Default Bitmask
CLOS1	A	A	A	A	A	A	A	A	
CLOS2	A	A	A	A	A	A	A	A	
CLOS3	A	A	A	A	A	A	A	A	

	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A	A	A	A	A	Overlapped Bitmask
CLOS1					A	A	A	A	
CLOS2							A	A	
CLOS3								A	

	M7	M6	M5	M4	M3	M2	M1	M0	
CLOS0	A	A	A	A					Isolated Bitmask
CLOS1					A	A			
CLOS2							A		
CLOS3								A	

Figure 19-27. Examples of Cache Capacity Bitmasks

Sample cache capacity bitmasks for a bit length of 8 are shown in Figure 19-27. Note that all (and only) contiguous '1' combinations are allowed (e.g., FFFFH, 0FF0H, 003CH, etc.), unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. Attempts to program a value without contiguous '1's (including zero) will result in a general protection fault (#GP(0)). It is generally expected that in way-based implementations, one capacity mask bit corresponds to some number of ways in cache, but the specific mapping is implementation-dependent. In all cases, a mask bit set to '1' specifies that a particular Class of Service can allocate into the cache subset represented by that bit. A value of '0' in a mask bit specifies that a Class of Service cannot allocate into the given cache subset. In general, allocating more cache to a given application is usually beneficial to its performance.

Figure 19-27 also shows three examples of sets of Cache Capacity Bitmasks. For simplicity these are represented as 8-bit vectors, though this may vary depending on the implementation and how the mask is mapped to the available cache capacity. The first example shows the default case where all 4 Classes of Service (the total number of CLOS are implementation-dependent) have full access to the cache. The second case shows an overlapped case,

which would allow some lower-priority threads to share cache space with the highest priority threads. The third case shows various non-overlapped partitioning schemes. As a matter of software policy for extensibility, CLOS0 should typically be considered and configured as the highest priority CLOS, followed by CLOS1, and so on, though there is no hardware restriction enforcing this mapping. When the system boots all threads are initialized to CLOS0, which has full access to the cache by default.

Though the representation of the CBMs looks similar to a way-based mapping they are independent of any specific enforcement implementation (e.g., way partitioning.) Rather, this is a convenient manner to represent capacity, overlap, and isolation of cache space. For example, executing a POPCNT instruction (population count of set bits) on the capacity bitmask can provide the fraction of cache space that a class of service can allocate into. In addition to the fraction, the exact location of the bits also shows whether the class of service overlaps with other classes of service or is entirely isolated in terms of cache space used.

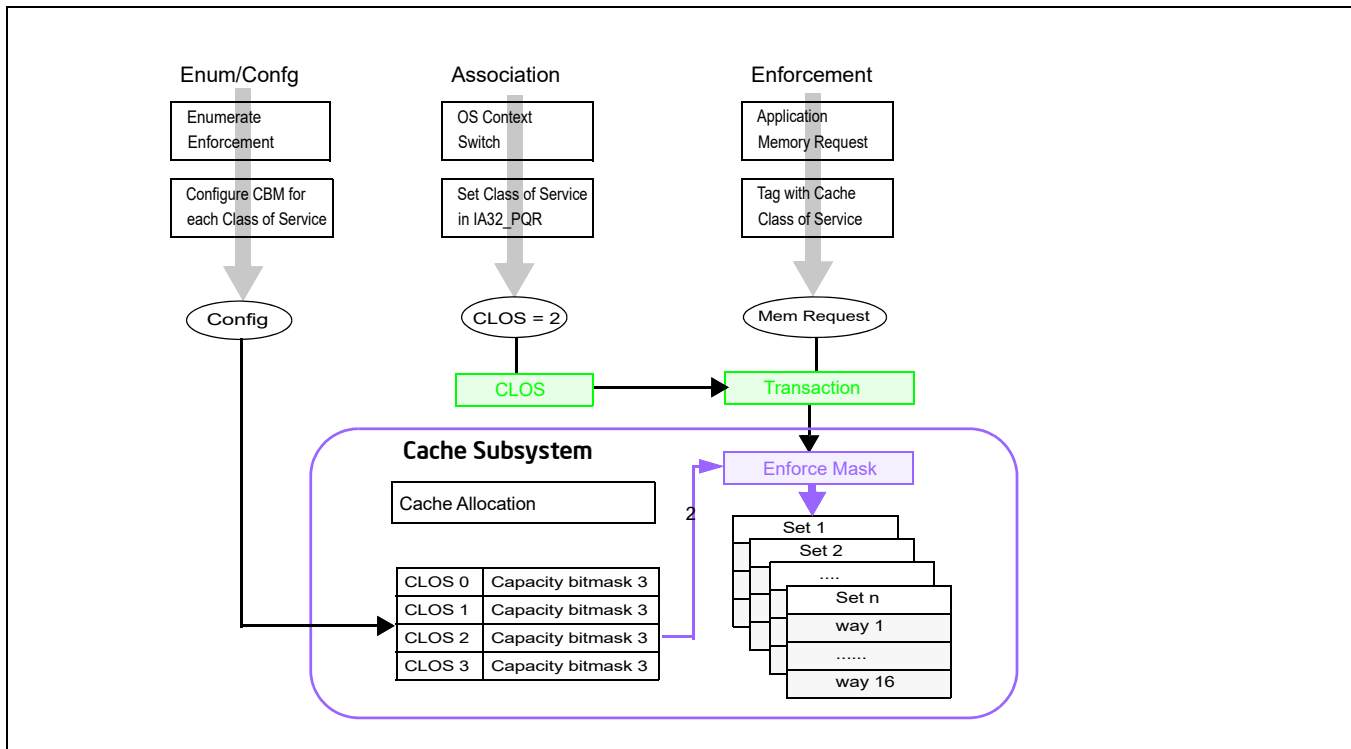


Figure 19-28. Class of Service and Cache Capacity Bitmasks

Figure 19-28 shows how the Cache Capacity Bitmasks and the per-logical-processor Class of Service are logically used to enable Cache Allocation Technology. All (and only) contiguous 1's in the CBM are permitted, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type. The length of a CBM may vary from resource to resource or between processor generations and can be enumerated using CPUID. From the available mask set and based on the goals of the OS/VMM (shared or isolated cache, etc.) bitmasks are selected and associated with different classes of service. For the available Classes of Service the associated CBMs can be programmed via the global set of CAT configuration registers (in the case of L3 CAT, via the IA32_L3_MASK_n MSRs, where "n" is the Class of Service, starting from zero). In all architectural implementations supporting CPUID it is possible to change the CBMs dynamically, during program execution, unless stated otherwise by Intel.

The currently running application's Class of Service is communicated to the hardware through the per-logical-processor PQR MSR (IA32_PQR_ASSOC MSR). When the OS schedules an application thread on a logical processor, the application thread is associated with a specific CLOS (i.e., the corresponding CLOS in the PQR) and all requests to the CAT-capable resource from that logical processor are tagged with that CLOS (in other words, the application thread is configured to belong to a specific CLOS). The cache subsystem uses this tagged request information to enforce QoS. The capacity bitmask may be mapped into a way bitmask (or a similar enforcement entity based on the implementation) at the cache before it is applied to the allocation policy. For example, the capacity bitmask can

be an 8-bit mask and the enforcement may be accomplished using a 16-way bitmask for a cache enforcement implementation based on way partitioning.

The following sections describe extensions of CAT such as Code and Data Prioritization (CDP), followed by details on specific features such as L3 CAT, L3 CDP, L2 CAT, and L2 CDP. Depending on the specific processor a mix of features may be supported, and CPUID provides enumeration capabilities to enable software to dynamically detect the set of supported features.

19.19.3 Code and Data Prioritization (CDP) Technology

Code and Data Prioritization Technology is an extension of CAT. CDP enables isolation and separate prioritization of code and data fetches to the L2 or L3 cache in a software configurable manner, depending on hardware support, which can enable workload prioritization and tuning of cache capacity to the characteristics of the workload. CDP extends Cache Allocation Technology (CAT) by providing separate code and data masks per Class of Service (CLOS). Support for the L2 CDP feature and the L3 CDP features are separately enumerated (via CPUID) and separately controlled (via remapping the L2 CAT MSRs or L3 CAT MSRs respectively). Section 19.19.6.3 and Section 19.19.7 provide details on enumerating, controlling, and enabling L3 and L2 CDP respectively, while this section provides a general overview.

The L3 CDP feature was first introduced on the Intel Xeon E5 v4 family of server processors, as an extension to L3 CAT. The L2 CDP feature is first introduced on future Intel Atom family processors, as an extension to L2 CAT.

By default, CDP is disabled on the processor. If the CAT MSRs are used without enabling CDP, the processor operates in a traditional CAT-only mode. When CDP is enabled:

- The CAT mask MSRs are re-mapped into interleaved pairs of mask MSRs for data or code fetches (see Figure 19-29).
- The range of CLOS for CAT is re-indexed, with the lower-half of the CLOS range available for CDP.

Using the CDP feature, virtual isolation between code and data can be configured on the L2 or L3 cache if desired, similar to how some processor cache levels provide separate L1 data and L1 instruction caches.

Like the CAT feature, CDP may be dynamically configured by privileged software at any point during normal system operation, including dynamically enabling or disabling the feature provided that certain software configuration requirements are met (see Section 19.19.5).

An example of the operating mode of CDP is shown in Figure 19-29. Shown at the top are traditional CAT usage models where capacity masks map 1:1 with a CLOS number to enable control over the cache space which a given CLOS (and thus applications, threads or VMs) may occupy. Shown at the bottom are example mask configurations where CDP is enabled, and each CLOS number maps 1:2 to two masks, one for code and one for data. This enables code and data to be either overlapped or isolated to varying degrees either globally or on a per-CLOS basis, depending on application and system needs.

Example of CAT-Only Usage - 16 bit Capacity Masks															
CLOS0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CLOS1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
CLOS2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
CLOS3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

Traditional CAT

Example of Code/Data Prioritization Usage - 16 bit Capacity Masks															
CLOS0.Data	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CLOS0.Code	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
CLOS1.Data	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
CLOS1.Code	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
Other CLOS.Data	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Other CLOS.Code	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

CAT with CDP

Figure 19-29. Code and Data Capacity Bitmasks of CDP

When CDP is enabled, the existing mask space for CAT-only operation is split. As an example if the system supports 16 CAT-only CLOS, when CDP is enabled the same MSR interfaces are used, however half of the masks correspond to code, half correspond to data, and the effective number of CLOS is reduced by half. Code/Data masks are defined per-CLOS and interleaved in the MSR space as described in subsequent sections.

In cases where CPUID exposes a non-even number of supported Classes of Service for the CAT or CDP features, software using CDP should use the lower matched pairs of code/data masks, and any upper unpaired masks should not be used. As an example, if CPUID exposes 5 CLOS, when CDP is enabled then two code/data pairs are available (masks 0/1 for CLOS[0] data/code and masks 2/3 for CLOS[1] data/code), however the upper un-paired mask should not be used (mask 4 in this case) or undefined behavior may result.

19.19.4 Enabling Cache Allocation Technology Usage Flow

Figure 19-30 illustrates the key steps for OS/VMM to detect support of Cache Allocation Technology and enable priority-based resource allocation for a CAT-capable resource.

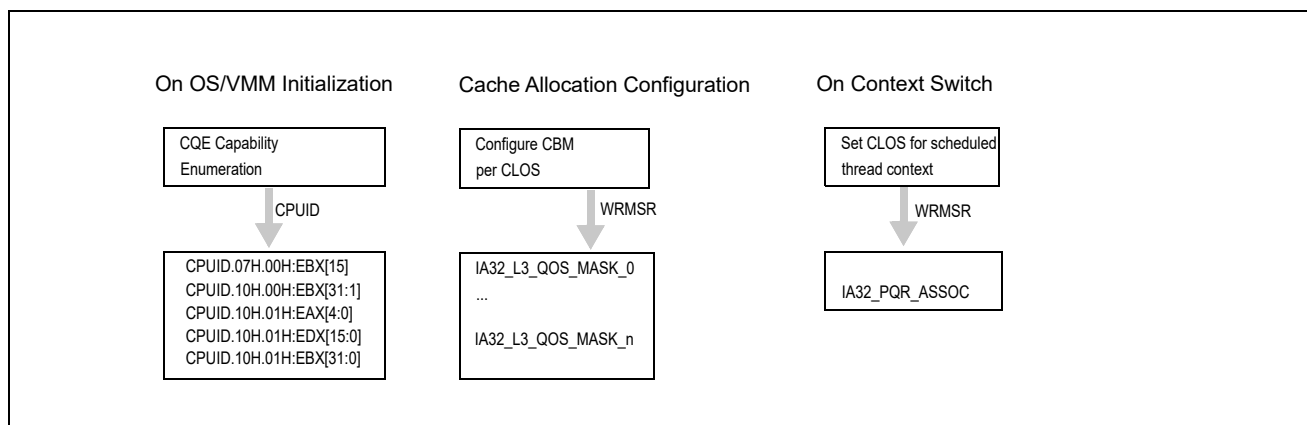


Figure 19-30. Cache Allocation Technology Usage Flow

Enumeration and configuration of L2 CAT is similar to L3 CAT, however CPUID details and MSR addresses differ. Common CLOS are used across the features.

19.19.4.1 Enumeration and Detection Support of Cache Allocation Technology

Software can query processor support of CAT capabilities by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_A[15] reports 1, the processor supports software control over shared processor resources. Software must use CPUID.10H to enumerate additional details of available resource types, classes of services and capability bitmasks. The programming interfaces provided by Cache Allocation Technology include:

- CPUID.10H (Cache Allocation Technology Enumeration leaf) and its sub-functions provide information on available resource types, and CAT capability for each resource type (see Section 19.19.4.2).
- IA32_L3_MASK_n: A range of MSRs is provided for each resource type, each MSR within that range specifying a software-configured capacity bitmask for each class of service. For L3 with Cache Allocation support, the CBM is specified using one of the IA32_L3_QOS_MASK_n MSR, where 'n' corresponds to a number within the supported range of CLOS, i.e., the range between 0 and CPUID.10H.ResID:EDX[15:0], inclusive. See Section 19.19.4.3 for details.
- IA32_L2_MASK_n: A range of MSRs is provided for L2 Cache Allocation Technology, enabling software control over the amount of L2 cache available for each CLOS. Similar to L3 CAT, a CBM is specified for each CLOS using the set of registers, IA32_L2_QOS_MASK_n MSR, where 'n' ranges from zero to the maximum CLOS number reported for L2 CAT in CPUID. See Section 19.19.4.3 for details.

The L2 mask MSRs are scoped at the same level as the L2 cache (similarly, the L3 mask MSRs are scoped at the same level as the L3 cache). Software may determine which logical processors share an MSR (for instance local to a core, or shared across multiple cores) by performing a write to one of these MSRs and noting which logical threads observe the change. Example flows for a similar method to determine register scope are described in Section 17.5.2, “System Software Recommendation for Managing CMCI and Machine Check Resources.” Software may also use CPUID.04H to determine the maximum number of logical processor IDs that may share a given level of the cache.

- IA32_PQR_ASSOC.CLOS: The IA32_PQR_ASSOC MSR provides a CLOS field that OS/VMM can use to assign a logical processor to an available CLOS. The set of CLOS are common across all allocation features, meaning that multiple features may be supported in the same processor without additional software CLOS management overhead at context swap time. See Section 19.19.4.4 for details.

19.19.4.2 Cache Allocation Technology: Resource Type and Capability Enumeration

CPUID.10H (Cache Allocation Technology Enumeration leaf) provides two or more sub-functions:

- CAT Enumeration leaf sub-function 0 enumerates available resource types that support allocation control, i.e., by using CPUID.10H.00H. Each supported resource type is represented by a bit field in CPUID.10H.00H:EBX[31:1]. The bit position of each set bit corresponds to a Resource ID (ResID), for instance ResID=1 is used to indicate L3 CAT support, and ResID=2 indicates L2 CAT support. The ResID is also the sub-leaf index that software must use to query details of the CAT capability of that resource type (see Figure 19-31).

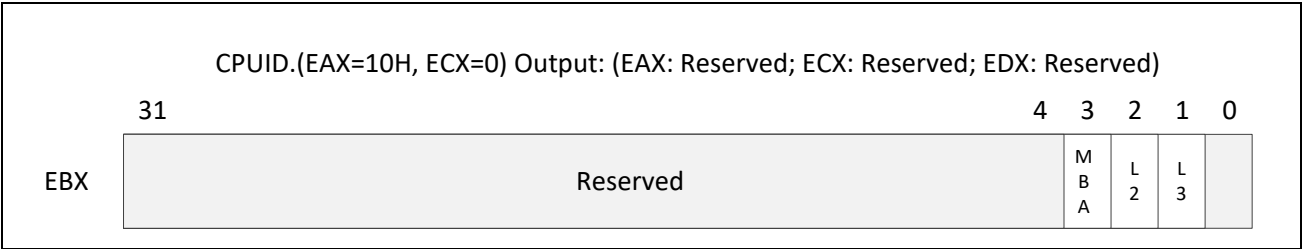


Figure 19-31. CPUID.10H.00H Available Resource Type Identification

- For ECX>0, EAX[4:0] reports the length of the capacity bitmask (ECX=1 or 2 for L3 CAT or L2 CAT respectively). Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
- Sub-leaves of CPUID.10H with a non-zero ECX input matching a supported ResID enumerate the specific enforcement details of the corresponding ResID. The capabilities enumerated include the length of the capacity bitmasks and the number of Classes of Service for a given ResID. Software should query the capability of each available ResID that supports CAT from a sub-leaf of leaf 10H using the sub-leaf index reported by the corresponding non-zero bit in CPUID.10H.00H:EBX[31:1] in order to obtain additional feature details.

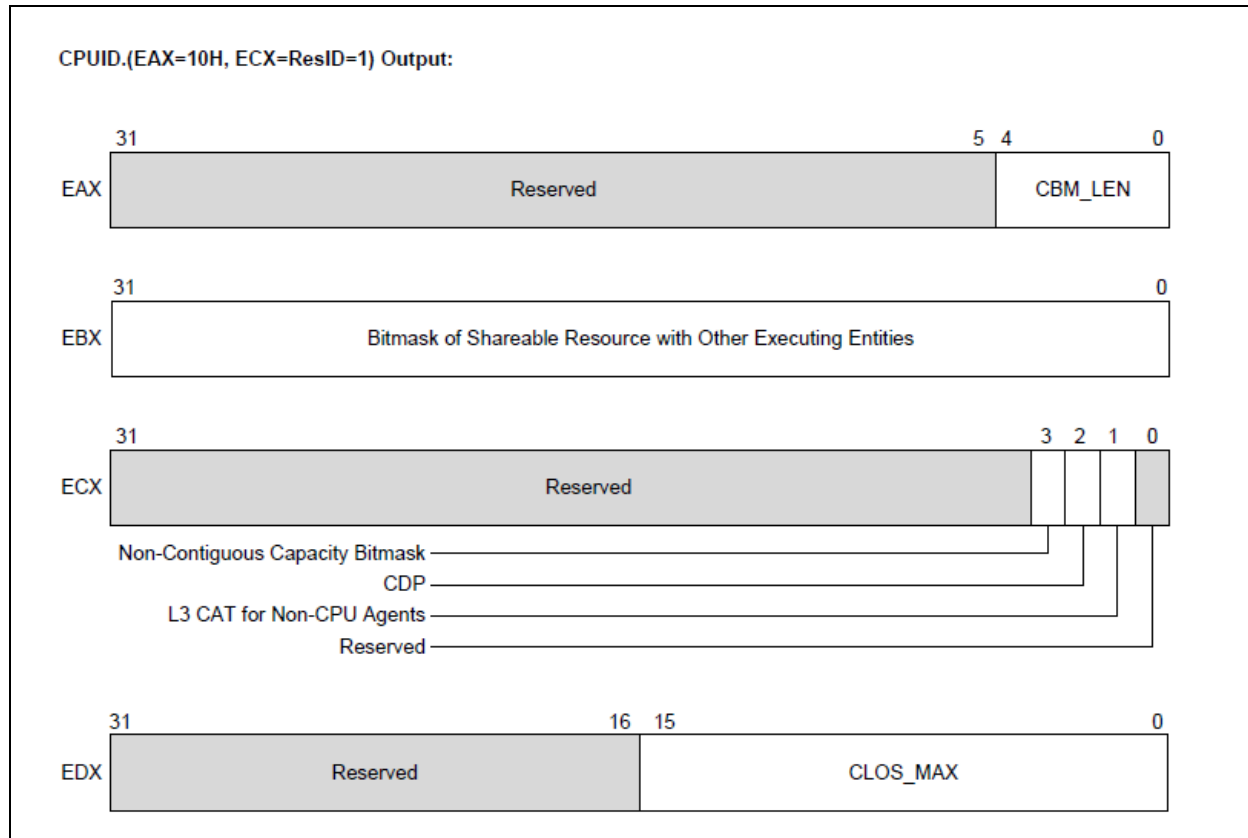


Figure 19-32. L3 Cache Allocation Technology and CDP Enumeration

- CAT capability for L3 is enumerated by CPUID.10H.01H, see Figure 19-32. The specific CAT capabilities reported by CPUID.10H.01H are:
 - CPUID.10H.ResID=1:EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
 - CPUID.10H.01H:EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L3 allocation may be used by other entities in the platform (e.g., an integrated graphics engine or hardware units outside the processor core and have direct access to L3). Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
 - CPUID.10H.01H:ECX[1]: If 1, indicates L3 CAT for non-CPU agents is supported. Bits 0 and 31:4 of ECX are reserved. See section 18.20 for details.
 - CPUID.10H.01H:ECX.CDP[2]: If 1, indicates L3 Code and Data Prioritization Technology is supported (see Section 19.19.5). Bits 0 and 31:4 of ECX are reserved.

- CPUID.10H.01H:ECX[3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L3_MASK_n registers do not have to be contiguous. Bits 0 and 31:4 of ECX are reserved.
- CPUID.10H.01H:EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

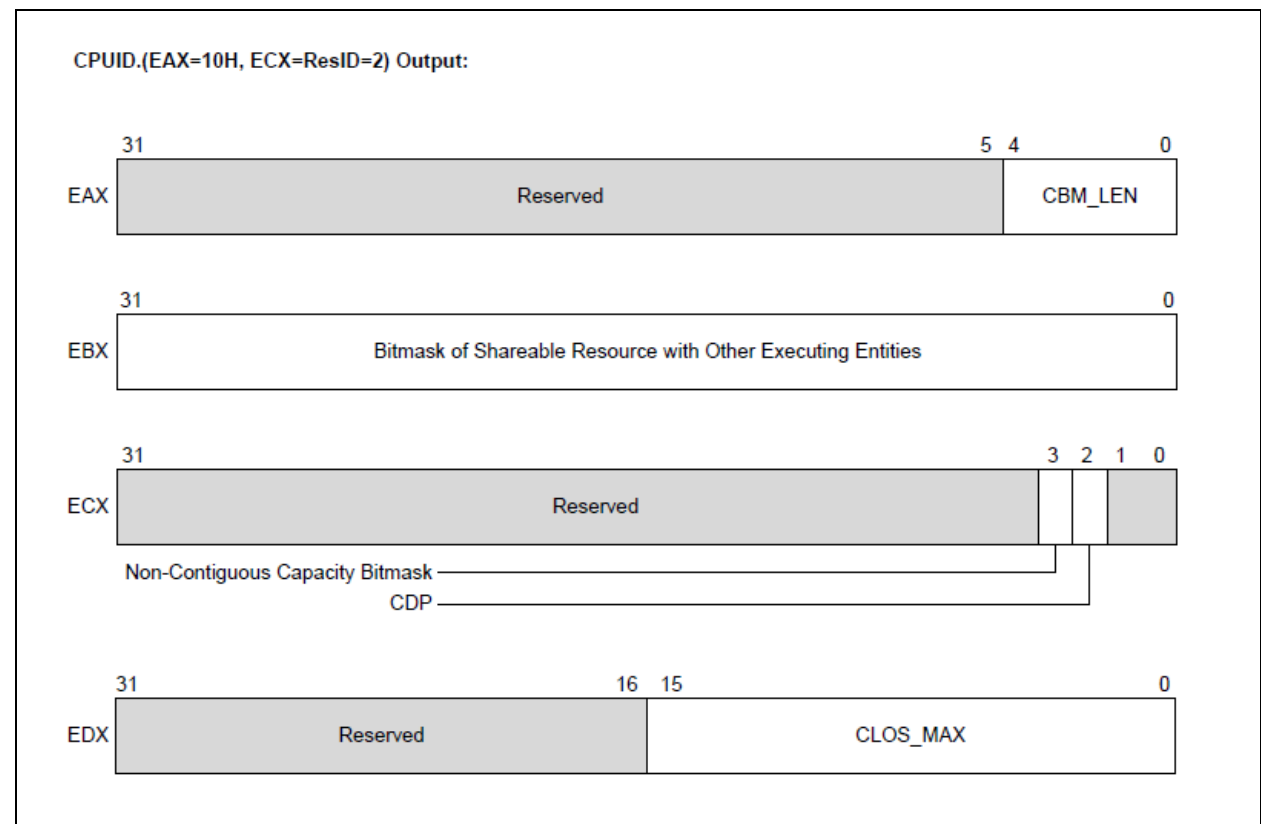


Figure 19-33. L2 Cache Allocation Technology

- CAT capability for L2 is enumerated by CPUID.10H.02H, see Figure 19-33. The specific CAT capabilities reported by CPUID.10H.02H are:
 - CPUID.10H.ResID=2:EAX[4:0] reports the length of the capacity bitmask. Add one to the return value to get the result, e.g., a value of 15 corresponds to the capacity bitmask having length of 16 bits. Bits 31:5 of EAX are reserved.
 - CPUID.10H.02H:EBX[31:0] reports a bit mask. Each set bit within the length of the CBM indicates the corresponding unit of the L2 allocation may be used by other entities in the platform. Each cleared bit within the length of the CBM indicates the corresponding allocation unit can be configured to implement a priority-based allocation scheme chosen by an OS/VMM without interference with other hardware agents in the system. Bits outside the length of the CBM are reserved.
 - CPUID.10H.02H:ECX.CDP[2]: If 1, indicates L2 Code and Data Prioritization Technology is supported (see Section 17.19.6). Bits 1:0 and 31:4 of ECX are reserved.
 - CPUID.10H.02H:ECX[3]: If 1, indicates non-contiguous capacity bitmask is supported. The bits which are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 1:0 and 31:4 of ECX are reserved.

- CPUID.10H.02H:EDX[15:0] reports the maximum CLOS supported for the resource (CLOS are zero-referenced, meaning a reported value of '15' would indicate 16 total supported CLOS). Bits 31:16 are reserved.

A note on migration of Classes of Service (CLOS): Software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the performance of the Cache Allocation Technology feature may result if CLOS are migrated frequently. This is aligned with the industry-standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and CLOS migration across processor logical threads and processor cores.

19.19.4.3 Cache Allocation Technology: Cache Mask Configuration

After determining the length of the capacity bitmasks (CBM) and number of CLOS supported using CPUID (see Section 19.19.4.2), each CLOS needs to be programmed with a CBM to dictate its available cache via a write to the corresponding IA32_resourceType_MASK_n register, where 'n' corresponds to a number within the supported range of CLOS, i.e., the range between 0 and CPUID.10H.ResID:EDX[15:0], inclusive, and 'resourceType' corresponds to a specific resource as enumerated by the set bits of CPUID.10H.00H:EBX[31:1], for instance, 'L2' or 'L3' cache.

A hierarchy of MSRs is reserved for Cache Allocation Technology registers of the form IA32_resourceType_MASK_n:

- From 0C90H through 0D8FH (inclusive), providing support for multiple sub-ranges to support varying resource types. The first supported resource type is 'L3', corresponding to the L3 cache in a platform. The MSRs range from 0C90H through 0D0FH (inclusive), enables support for up to 128 L3 CAT Classes of Service.

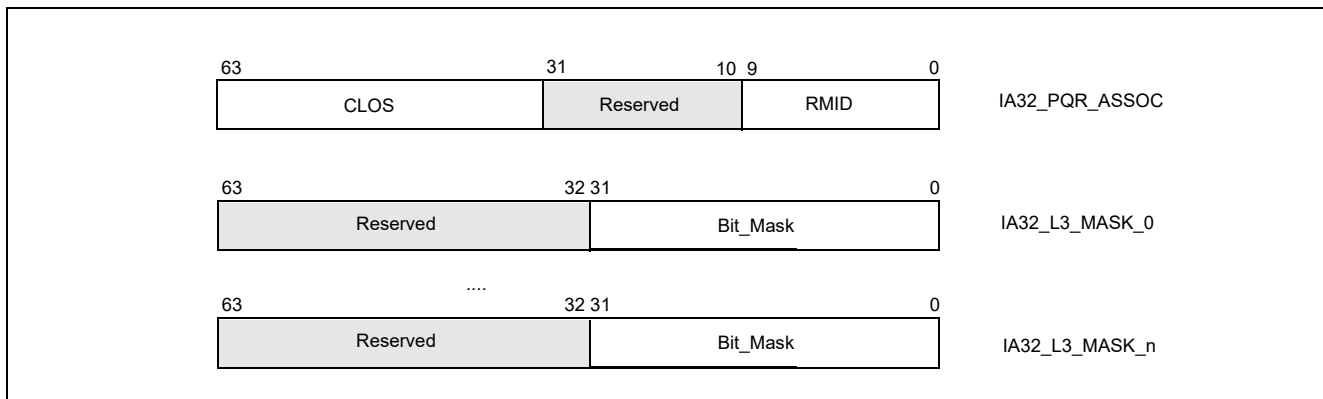


Figure 19-34. IA32_PQR_ASSOC, IA32_L3_MASK_n MSRs

- Within the same CAT range hierarchy, another set of registers is defined for resourceType 'L2', corresponding to the L2 cache in a platform, and MSRs IA32_L2_MASK_n are defined for n=[0,63] at addresses 0D10H through 0D4FH (inclusive).

Figure 19-34 and Figure 19-35 provide an overview of the relevant registers.

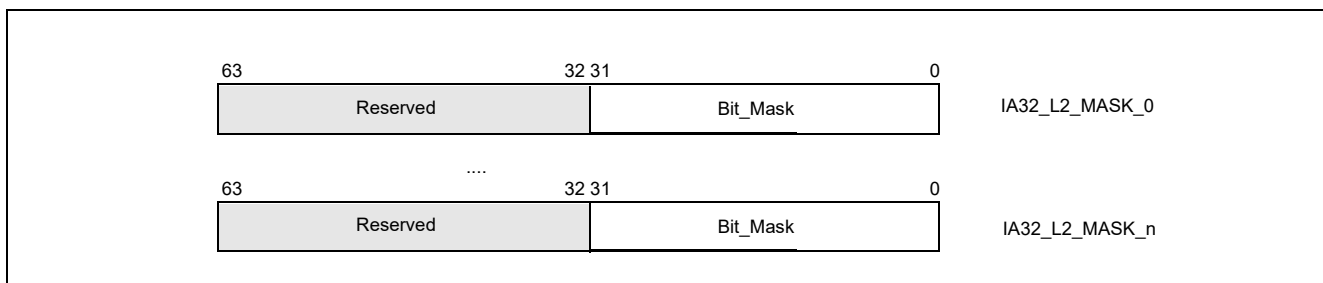


Figure 19-35. IA32_L2_MASK_n MSRs

All CAT configuration registers can be accessed using the standard RDMSR / WRMSR instructions. Note that once L3 or L2 CAT masks are configured, threads can be grouped into Classes of Service (CLOS) using the IA32_PQR_ASSOC MSR as described in Section 19.19.4.4, “Class of Service to Cache Mask Association: Common Across Allocation Features.”

19.19.4.4 Class of Service to Cache Mask Association: Common Across Allocation Features

After configuring the available classes of service with the preferred set of capacity bitmasks, the OS/VMM can set the IA32_PQR_ASSOC.CLOS of a logical processor to the class of service with the desired CBM when a thread context switch occurs. This allows the OS/VMM to indicate which class of service an executing thread/VM belongs within. Each logical processor contains an instance of the IA32_PQR_ASSOC register at MSR location 0C8FH, and Figure 19-34 shows the bit field layout for this register. Bits[63:32] contain the CLOS field for each logical processor.

Note that placing the RMID field within the same PQR register enables both RMID and CLOS to be swapped at context swap time for simultaneous use of monitoring and allocation features with a single register write for efficiency.

When CDP is enabled, Specifying a CLOS value in IA32_PQR_ASSOC.CLOS greater than MAX_CLOS_CDP = (CPUID.10H.01H:EDX[15:0] >> 1) will cause undefined performance impact to code and data fetches. In all cases, code and data masks for L2 and L3 CDP should be programmed with at least one bit set.

Note that if the IA32_PQR_ASSOC.CLOS is never written then the CAT capability defaults to using CLOS 0, which in turn is set to the default mask in IA32_L3_MASK_0 - which is all “1”s (on reset). This essentially disables the enforcement feature by default or for legacy operating systems and software.

See Section 19.19.7, “Introduction to Memory Bandwidth Allocation,” for important CLOS programming considerations including maximum values when using CAT and CDP.

19.19.5 Code and Data Prioritization (CDP): Enumerating and Enabling L3 CDP Technology

L3 CDP is an extension of L3 CAT. The presence of the L3 CDP feature is enumerated via CPUID.10H.01H:ECX.CDP[2] (see Figure 19-32). Most of the CPUID.10H.01H sub-leaf data that applies to CAT also apply to CDP. However, CPUID.10H.01H:EDX.CAT_MAX_CLOS specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.10H.01H:EDX.CAT_MAX_CLOS >> 1).

If CPUID.10H.01H:ECX.CDP[2] = 1, the processor supports CDP and provides a new MSR IA32_L3_QOS_CFG at address 0C81H. The layout of IA32_L3_QOS_CFG is shown in Figure 19-36. The bit field definition of IA32_L3_QOS_CFG are:

- Bit 0: L3 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

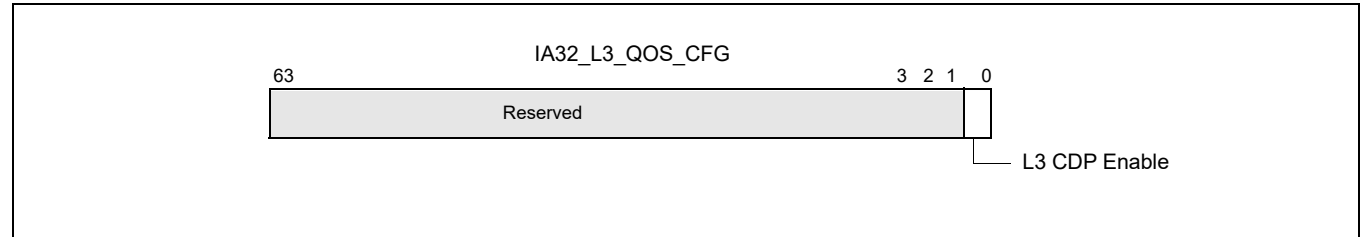


Figure 19-36. Layout of IA32_L3_QOS_CFG

IA32_L3_QOS_CFG default values are all 0s at RESET, the mask MSRs are all 1s. Hence, all logical processors are initialized in CLOS0 allocated with the entire L3 with CDP disabled, until software programs CAT and CDP. The scope of the IA32_L3_QOS_CFG MSR is defined to be the same scope as the L3 cache (e.g., typically per processor socket). Refer to Section 19.19.7 for software considerations while enabling or disabling L3 CDP.

19.19.5.1 Mapping Between L3 CDP Masks and CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per CLOS. The re-mapping is shown in Table 19-19.

Table 19-19. Re-indexing of CLOS Numbers and Mapping to CAT/CDP Mask MSRs

Mask MSR	CAT-only Operation	CDP Operation
IA32_L3_QOS_Mask_0	CLOS0	CLOS0.Data
IA32_L3_QOS_Mask_1	CLOS1	CLOS0.Code
IA32_L3_QOS_Mask_2	CLOS2	CLOS1.Data
IA32_L3_QOS_Mask_3	CLOS3	CLOS1.Code
IA32_L3_QOS_Mask_4	CLOS4	CLOS2.Data
IA32_L3_QOS_Mask_5	CLOS5	CLOS2.Code
....		
IA32_L3_QOS_Mask_‘2n’	CLOS‘2n’	CLOS‘n’.Data
IA32_L3_QOS_Mask_‘2n+1’	CLOS‘2n+1’	CLOS‘n’.Code

One can derive the MSR address for the data mask or code mask for a given CLOS number ‘n’ by:

- $\text{data\_mask\_address}(n) = \text{base} + (n \ll 1)$, where base is the address of IA32_L3_QOS_MASK_0.
- $\text{code\_mask\_address}(n) = \text{base} + (n \ll 1) + 1$.

When CDP is enabled, each CLOS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 19-29).

Mask MSR field definitions remain the same. Capacity masks must be formed of contiguous set bits, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

19.19.6 Code and Data Prioritization (CDP): Enumerating and Enabling L2 CDP Technology

L2 CDP is an extension of the L2 CAT feature. The presence of the L2 CDP feature is enumerated via CPUID.10H.02H:ECX.CDP[2] (see Figure 17-33). Most of the CPUID.10H.02H sub-leaf data that applies to CAT also apply to CDP. However, CPUID.10H.02H:EDX.CAT_MAX_CLOS specifies the maximum CLOS applicable to CAT-only operation. For CDP operations, CLOS_MAX_CDP is equal to (CPUID.10H.02H:EDX.CAT_MAX_CLOS >> 1).

If CPUID.10H.02H:ECX.CDP[2] = 1, the processor supports L2 CDP and provides a new MSR IA32_L2_QOS_CFG at address 0C82H. The layout of IA32_L2_QOS_CFG is shown in Figure 19-37. The bit field definition of IA32_L2_QOS_CFG are:

- Bit 0: L2 CDP Enable. If set, enables CDP, maps CAT mask MSRs into pairs of Data Mask and Code Mask MSRs. The maximum allowed value to write into IA32_PQR_ASSOC.CLOS is CLOS_MAX_CDP.
- Bits 63:1: Reserved. Attempts to write to reserved bits result in a #GP(0).

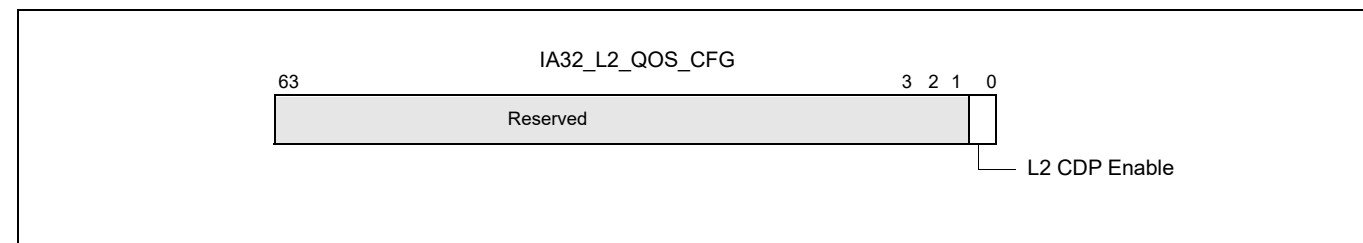


Figure 19-37. Layout of IA32_L2_QOS_CFG

IA32_L2_QOS_CFG default values are all 0s at RESET, and the mask MSRs are all 1s. Hence all logical processors are initialized in CLOS0 allocated with the entire L2 available and with CDP disabled, until software programs CAT and CDP. The IA32_L2_QOS_CFG MSR is defined at the same scope as the L2 cache, typically at the module level for Intel Atom processors for instance. In processors with multiple modules present it is recommended to program the IA32_L2_QOS_CFG MSR consistently across all modules for simplicity.

19.19.6.1 Mapping Between L2 CDP Masks and L2 CAT Masks

When CDP is enabled, the existing CAT mask MSR space is re-mapped to provide a code mask and a data mask per CLOS. This remapping is the same as the remapping shown in Table 19-19 for L3 CDP, but for the L2 MSR block (IA32_L2_QOS_MASK_n) instead of the L3 MSR block (IA32_L3_QOS_MASK_n). The same code / data mask mapping algorithm applies to remapping the MSR block between code and data masks.

As with L3 CDP, when L2 CDP is enabled, each CLOS is mapped 1:2 with mask MSRs, with one mask enabling programmatic control over data fill location and one mask enabling control over code placement. A variety of overlapped and isolated mask configurations are possible (see the example in Figure 19-29).

Mask MSR field definitions for L2 CDP remain the same as for L2 CAT. Capacity masks must be formed of contiguous set bits, unless otherwise non-contiguous capacity bitmask support is specified in CPUID enumeration for the resource type with a length of 1 bit or longer and should not exceed the maximum mask length specified in CPUID. As examples, valid masks on a cache with max bitmask length of 16b (from CPUID) include 0xFFFF, 0xFF00, 0x00FF, 0x00F0, 0x0001, 0x0003, and so on. Maximum valid mask lengths are unchanged whether CDP is enabled or disabled, and writes of invalid mask values may lead to undefined behavior. Writes to reserved bits will generate #GP(0).

19.19.6.2 Common L2 and L3 CDP Programming Considerations

Before enabling or disabling L2 or L3 CDP, software should write all 1's to all of the corresponding CAT/CDP masks to ensure proper behavior (e.g., the IA32_L3_QOS_Mask_n set of MSRs for the L3 CAT feature). When enabling CDP, software should also ensure that only CLOS number which are valid in CDP operation is used, otherwise undefined behavior may result. For instance in a case with 16 CAT CLOS, since CLOS are reduced by half when CDP is enabled, software should ensure that only CLOS 0-7 are in use before enabling CDP (along with writing 1's to all mask bits before enabling or disabling CDP).

Software should also account for the fact that mask interpretations change when CDP is enabled or disabled, meaning for instance that a CAT mask for a given CLOS may become a code mask for a different Class of Service when CDP is enabled. In order to simplify this behavior and prevent unintended remapping software should consider resetting all threads to CLOS[0] before enabling or disabling CDP.

19.19.6.3 Cache Allocation Technology Dynamic Configuration

All Intel Resource Director Technology (Intel RDT) interfaces including the IA32_PQR_ASSOC MSR, CAT/CDP masks, MBA delay values, and CQM/MBM registers are accessible and modifiable at any time during execution using RDMSR/WRMSR unless otherwise noted. When writing to these MSRs a #GP(0) will be generated if any of the following conditions occur:

- A reserved bit is modified,

- Accessing a QOS mask register outside the supported CLOS (the max CLOS number is specified in CPUID.10H.ResID:EDX[15:0]), or
- Writing a CLOS greater than the supported maximum (specified as the maximum value of CPUID.10H.ResID:EDX[15:0] for all valid ResID values) is written to the IA32_PQR_ASSOC.CLOS field.

When CDP is enabled, specifying a CLOS value in IA32_PQR_ASSOC.CLOS outside of the lower half of the CLOS space will cause undefined performance impact to code and data fetches due to MSR space re-indexing into code/data masks when CDP is enabled.

When reading the IA32_PQR_ASSOC register the currently programmed CLOS on the core will be returned.

When reading an IA32_resourceType_MASK_n register the current capacity bit mask for CLOS 'n' will be returned.

As noted previously, software should minimize migrations of CLOS across logical processors (across threads or cores), as a reduction in the accuracy of the Cache Allocation feature may result if CLOS are migrated frequently. This is aligned with the industry standard practice of minimizing unnecessary thread migrations across processor cores in order to avoid excessive time spent warming up processor caches after a migration. In general, for best performance, minimize thread migration and CLOS migration across processor logical threads and processor cores.

19.19.6.4 Cache Allocation Technology Operation With Power Saving Features

Note that the Cache Allocation Technology feature cannot be used to enforce cache coherency, and that some advanced power management features such as C-states which may shrink or power off various caches within the system may interfere with CAT hints - in such cases the CAT bitmasks are ignored and the other features take precedence. If the highest possible level of CAT differentiation or determinism is required, disable any power-saving features which shrink the caches or power off caches. The details of the power management interfaces are typically implementation-specific, but can be found at Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C.

If software requires differentiation between threads but not absolute determinism then in many cases it is possible to leave power-saving cache shrink features enabled, which can provide substantial power savings and increase battery life in mobile platforms. In such cases when the caches are powered off (e.g., package C-states) the entire cache of a portion thereof may be powered off. Upon resuming an active state any new incoming data to the cache will be filled subject to the cache capacity bitmasks. Any data in the cache prior to the cache shrink or power off may have been flushed to memory during the process of entering the idle state, however, and is not guaranteed to remain in the cache. If differentiation between threads is the goal of system software then this model allows substantial power savings while continuing to deliver performance differentiation. If system software needs optimal determinism then power saving modes which flush portions of the caches and power them off should be disabled.

NOTE

IA32_PQR_ASSOC is saved and restored across C6 entry/exit. Similarly, the mask register contents are saved across package C-state entry/exit and are not lost.

19.19.6.5 Cache Allocation Technology Operation with Other Operating Modes

The states in IA32_PQR_ASSOC and mask registers are unmodified across an SMI delivery. Thus, the execution of SMM handler code can interact with the Cache Allocation Technology resource and manifest some degree of non-determinism to the non-SMM software stack. An SMM handler may also perform certain system-level or power management practices that affect CAT operation.

It is possible for an SMM handler to minimize the impact on data determinism in the cache by reserving a CLOS with a dedicated partition in the cache. Such an SMM handler can switch to the dedicated CLOS immediately upon entering SMM, and switching back to the previously running CLOS upon exit.

19.19.6.6 Associating Threads with CAT/CDP Classes of Service

Threads are associated with Classes of Service (CLOS) via the per-logical-processor IA32_PQR_ASSOC MSR. The same CLOS concept applies to both CAT and CDP (for instance, CLOS[5] means the same thing whether CAT or CDP

is in use, and the CLOS has associated resource usage constraint attributes including cache capacity masks). The mapping of CLOS to mask MSRs does change when CDP is enabled, according to the following guidelines:

- In CAT-only Mode - one set of bitmasks in one mask MSR control both code and data.
 - Each CLOS number map 1:1 with a capacity mask on the applicable resource (e.g., L3 cache).
- When CDP is enabled:
 - Two mask sets exist for each CLOS number, one for code, one for data.
 - Masks for code/data are interleaved in the MSR address space (see Table 19-19).

19.19.7 Introduction to Memory Bandwidth Allocation

The Memory Bandwidth Allocation (MBA) feature provides indirect and approximate control over memory bandwidth available per-core. It was introduced in the Intel Xeon Scalable Processor Family. This feature provides a method to control applications that may be over-utilizing bandwidth relative to their priority in environments such as the data-center.

The MBA feature uses existing constructs from the Intel RDT feature set, including Classes of Service (CLOS). A given CLOS used for L3 CAT, for instance, means the same thing as a CLOS used for MBA. Infrastructure, such as the MSR used to associate a thread with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID.10H), are shared. Certain generations include advanced hardware controllers for efficiency. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

The following sections describe CPU interfaces to Memory Bandwidth Allocation, such as CPUID enumeration and configuration interfaces (MSRs).

19.19.7.1 Memory Bandwidth Allocation Enumeration

Similar to other Intel RDT features, enumeration of the presence and details of the MBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration are as follows.

- Support for the MBA feature on the processor, and if MBA is supported, the following details:
 - Number of supported Classes of Service (CLOS) for the processor.
 - The maximum MBA delay value supported (which also implicitly provides a definition of the granularity).
 - An indication of whether the delay values which can be programmed are linearly spaced or not.

The presence of any of the Intel RDT features which enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H, ECX = 0H as input. If CPUID.07H.00H:EBX.RDT_A[15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID.10H to enumerate additional details on the specific controls provided.

Through CPUID.10H software may determine whether MBA is supported on the platform. Specifically, as shown in Figure 19-31, bit 3 of the EBX register indicates whether MBA is supported on the processor, and the bit position (3) constitutes a Resource ID (ResID) which allows enumeration of MBA details. For instance, if bit 3 is supported this implies the presence of CPUID.10H.[ResID=3] as shown in Figure 19-38 which provides the following details.

- CPUID.10H.ResID=3:EAX[11:0] reports the maximum MBA throttling value supported, minus one. For instance, a value of 89 indicates that a maximum throttling value of 90 is supported. Additionally, in cases where a linear interface (see below) is supported then one hundred minus the maximum throttling value indicates the granularity, 10% in this example.
- CPUID.10H.ResID=3:EBX is reserved.
- CPUID.10H.ResID=3:ECX[0] indicates that the MBA throttling values are applied per-logical-processor independently if set.
- CPUID.10H.ResID=3:ECX[2] reports whether the response of the delay values is linear (see text).
- CPUID.10H.ResID=3:EDX[15:0] reports the number of Classes of Service (CLOS) supported for the feature (minus one). For instance, a reported value of 15 implies a maximum of 16 supported MBA CLOS.

The number of CLOS supported for the MBA feature may or may not align with other resources such as L3 CAT. In cases where the Intel RDT features support different numbers of CLOS the lowest numerical CLOS support the common set of features, while higher CLOS may support a subset. For instance, if L3 CAT supports 8 CLOS while MBA supports 4 CLOS, all 8 CLOS would have L3 CAT masks available for cache control, but the upper 4 CLOS would not offer MBA support. In this case the upper 4 CLOS would not be subject to any throttling control. Software can manage supported resources / CLOS in order to either have consistent capabilities across CLOS by using the common subset or enable more flexibility by selectively applying resource control where needed based on careful CLOS and thread mapping. In all cases, CLOS[0] supports all Intel RDT resource control features present on the platform.

Discussion on the interpretation and usage of the MBA delay values is provided in Section 19.19.7.2 on MBA configuration.

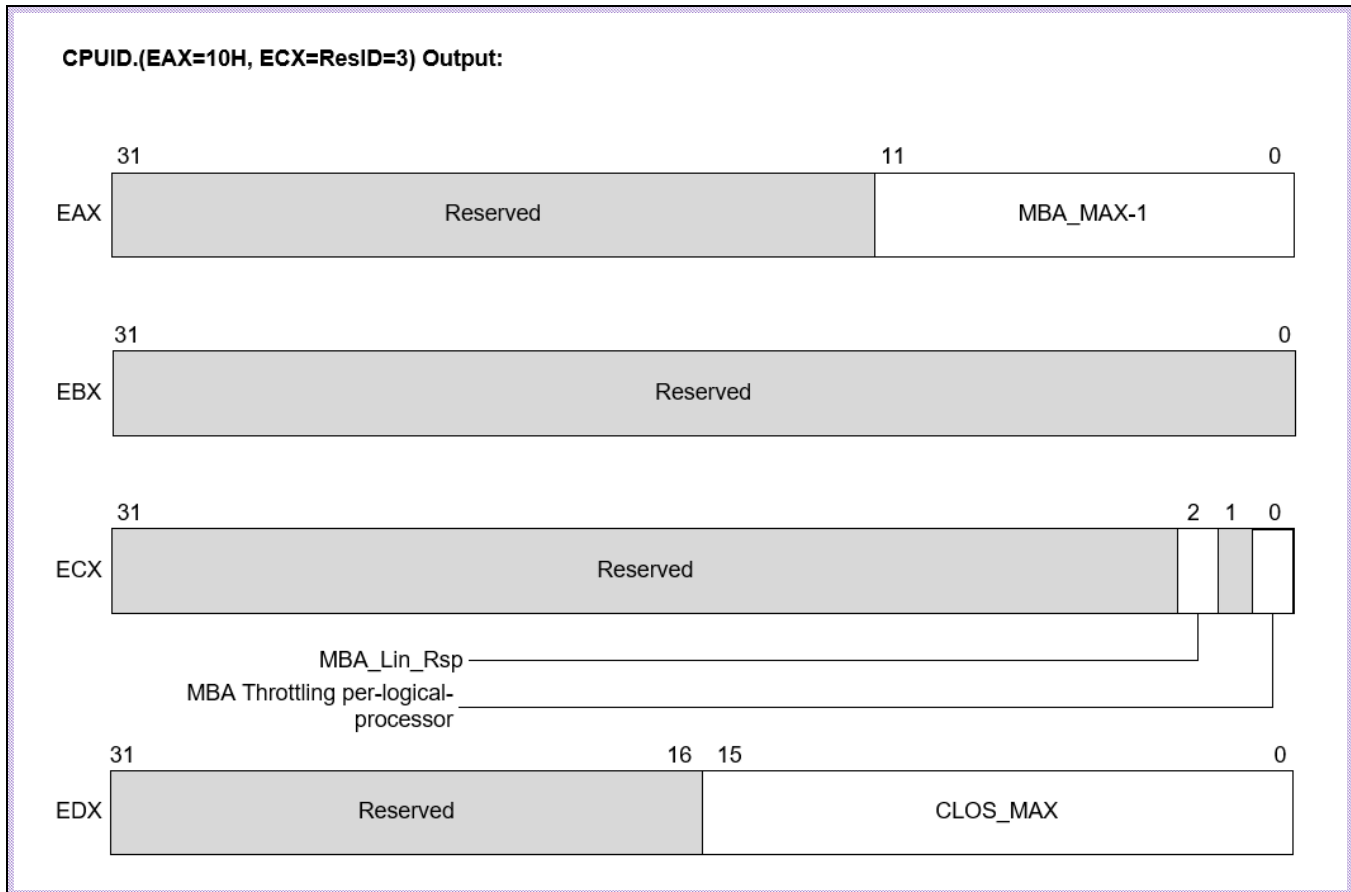


Figure 19-38. CPUID.10H.03H MBA Feature Details Identification

19.19.7.2 Memory Bandwidth Allocation Configuration

The configuration of MBA takes consists of two processes once enumeration is complete.

- Association of threads to Classes of Service (CLOS) - accomplished in a common fashion across Intel RDT features as described in Section 19.19.7.1 via the `IA32_PQR_ASSOC` MSR. As with features such as L3 CAT, software may update the CLOS field of the PQR MSR at context swap time in order to maintain the proper association of software threads to Classes of Service on the hardware. While logical processors may each be associated with independent CLOS, see Section 19.19.7.3 for important usage model considerations (initial versions of the MBA feature select the maximum delay value across threads).
- Configuration of the per-CLOS delay values, accomplished via the `IA32_L2_QoS_Ext_BW_Thrtl_n` MSR set shown in Table 19-20.

The MBA delay values which may be programmed range from zero (implying zero delay, and full bandwidth available) to the maximum (MBA_MAX) specified in CPUID as discussed in Section 19.19.7.1. The throttling values are approximate and do not sum to 100% across CLOS, rather they should be viewed as a maximum bandwidth “cap” per-CLOS.

Software may select an MBA delay value then write the value into one or more of the IA32_L2_QoS_Ext_BW_Thrtl_n MSRs to update the delay values applied for a specific CLOS. As shown in Table 19-20 the base address of the MSRs is at D50H, and the range corresponds to the maximum supported CLOS from CPUID.10H.ResID=1:EDX[15:0] as described in Section 19.19.7.1. For instance, if 16 CLOS are supported then the valid MSR range will extend from D50H through D5F inclusive.

Table 19-20. MBA Delay Value MSRs

Delay Value MSR	Address
IA32_L2_QoS_Ext_BW_Thrtl_0	D50H
IA32_L2_QoS_Ext_BW_Thrtl_1	D51H
IA32_L2_QoS_Ext_BW_Thrtl_2	D52H
....	
IA32_L2_QoS_Ext_BW_Thrtl_‘CLOS_MAX’	D50H + CLOS_MAX from CPUID.10H.03H

The definition for the MBA delay value MSRs is provided in Figure 17.39. The lower 16 bits are used for MBA delay values, and values from zero to the maximum from the CPUID MBA_MAX-1 value are supported. Values outside this range will generate #GP(0).

If linear input throttling values are indicated by CPUID.10H.ResID=3:ECX[2] then values from zero through the MBA_MAX field from CPUID.10H.ResID=3:EAX[11:0] are supported as inputs. In the linear mode the input precision is defined as 100-(MBA_MAX). For instance, if the MBA_MAX value is 90, the input precision is 10%. Values not an even multiple of the precision (e.g., 12%) will be rounded down (e.g., to 10% delay applied).

- If linear values are not supported (CPUID.10H.ResID=3:ECX[2] = 0) then input delay values are powers-of-two from zero to the MBA_MAX value from CPUID. In this case any values not a power of two will be rounded down the next nearest power of two.

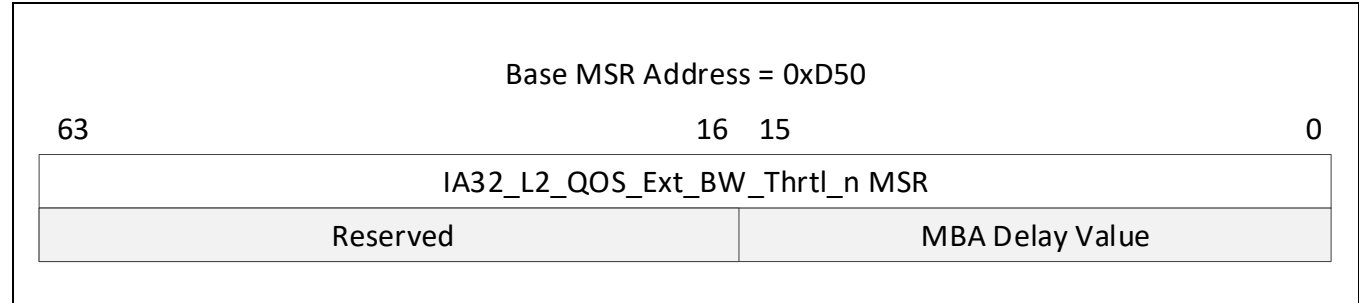


Figure 19-39. IA32_L2_QoS_Ext_BW_Thrtl_n MSR Definition

Note that the throttling values provided to software are calibrated through specific traffic patterns, however as workload characteristics may vary the response precision and linearity of the delay values will vary across products and should be treated as approximate values only.

19.19.7.3 Memory Bandwidth Allocation Usage Considerations

Different versions of Memory Bandwidth Allocation have various usage considerations and improving efficiency over time. See the “Intel® Resource Director Technology Architecture Specification” for additional details.

19.20 INTEL® RESOURCE DIRECTOR TECHNOLOGY (INTEL® RDT) FOR NON-CPU AGENTS

This section describes Intel RDT features for non-CPU agents. CPU agents are threads running on IA cores. Non-CPU agents include PCIe and CXL devices and integrated accelerators, thus broadly encompassing the set of agents that read from and write to either caches or memory, excluding IA cores. The non-CPU agent Intel RDT features enable monitoring of I/O device shared cache and memory bandwidth and cache allocation control. This provides features for I/O devices equivalent to the CPU agent Intel RDT capabilities CMT, MBM, and CAT (discussed in Section 19.18 and Section 19.19). Refer to the “Intel® Resource Director Technology Architecture Specification” regarding design goals, use cases, software architecture, ACPI enumeration, and MMIO register interfaces.

“Non-CPU agent Intel RDT” refers to capabilities that monitor and control non-CPU agents' resource utilization, including PCIe and CXL devices and integrated accelerators. Non-CPU agent Intel RDT may be called I/O RDT in some literature. In this document, the term “non-CPU agent Intel RDT” is used.

19.20.1 Non-CPU Agent Intel® RDT Features Enumeration Details

CPU agent Intel RDT features use the CPUID instruction to enumerate supported features and the level of support. Architectural Model-Specific Registers (MSRs) are interfaces to the monitoring and allocation features, as described in Sections 18.18 and 18.19.

Non-CPU agent Intel RDT builds on CPU agent Intel RDT by extending CPUID to indicate the presence and integration of non-CPU agent Intel RDT and by providing rich enumeration information in vendor-specific extensions to the Advanced Configuration and Power Interface (ACPI), in particular in the I/O RDT (IRDT) table. The ACPI extensions detailed in the “Intel® Resource Director Technology Architecture Specification” provide mechanisms to comprehend the structure of devices attached behind I/O blocks to particular links and what forms of tagging are supported on a per-link basis.

It is recommended that software parse CPUID and ACPI to obtain a detailed understanding of platform support and capabilities before attempting to use non-CPU agent Intel RDT.

19.20.1.1 CPUID-Based Enumeration for Non-CPU Agent Intel® RDT Feature

CPUID-based enumeration provides a method by which all architectural Intel RDT features may be enumerated.

For CPU agent Intel RDT, monitoring details are enumerated in a CPUID sub-leaf denoted as CPUID.0FH.ResID, where ResID corresponds to a resource ID bit index from the CPUID.0FH.00H sub-leaf. Similarly, Intel RDT allocation features are described in CPUID.10H.ResID. (Note that the ResID bit positions are not guaranteed to be symmetric or have the same encodings.)

No CPUID leaves or sub-leaves are created for non-CPU agent Intel RDT. Rather, non-CPU agent Intel RDT extends the existing Intel RDT CPUID sub-leaves with a bit per resource type, indicating whether non-CPU agent Intel RDT monitoring or control is present. CPUID.0FH.ResID=1:EAX[10:9] represents the presence of CMT and MBM features for non-CPU agents. CPUID.10H.ResID=1:ECX[1] represents the presence of the CAT feature for non-CPU agents.

Specifically, for non-CPU Agent Intel RDT Monitoring (see Figure 19-21):

- Bits are added in the CPU Agent Intel RDT CMT/MBM leaf: CPUID.0FH.ResID=1:EAX[10:9].
 - EAX[bit 9]: If set, indicates the presence of non-CPU Agent Cache Occupancy Monitoring (the equivalent of CPU Agent Intel RDT's CMT feature).
 - EAX[bit 10]: If set, indicates the presence of non-CPU Agent memory L3 external BW monitoring (the equivalent of CPU Agent Intel RDT's MBM feature).

For non-CPU Agent Intel RDT Allocation (see Figure 19-32):

- New bit in L3 CAT leaf: CPUID.10H.ResID=1:ECX[1].
 - ECX[bit 1]: If set, indicates the presence of non-CPU Agent Cache Allocation Technology (the equivalent of CPU Agent Intel RDT's L3 CAT feature).
- As before, ECX[bit 2] indicates that L3 CDP is supported if set.

Note that no equivalent bits are defined in CPUID.10H.ResID=2 as there is no ability for devices to fill into core L2 caches.

If any of these non-CPU agent Intel RDT enumeration bits are set, indicating that a monitoring feature or allocation feature is present, it also indicates the presence of the IA32_L3_IO_RDT_CFG architectural MSR. This MSR may be used to enable the non-CPU agent Intel RDT features. See Section 19.20.2 for MSR details.

The presence of Intel RDT is a prerequisite for using the equivalent non-CPU agent Intel RDT feature. If a particular CPU agent Intel RDT feature is absent, any attempt to use non-CPU agent Intel RDT equivalents will result in general protection faults in the MSR interface. Attempts to enable unsupported features in the I/O complex will result in writes to the corresponding MMIO enable or configuration interfaces being ignored.

Software may use the existing CPUID leaves to gather the maximum number of RMID and CLOS tags for each resource level (e.g., L3 cache), and non-CPU agent Intel RDT is also subject to these limits.

Some platforms may support a mix of features, for instance, supporting L3 CAT architectural controls and the non-CPU agent Intel RDT equivalent, but no CMT/MBM monitoring or non-CPU agent monitoring equivalent, and these capabilities should be enumerated on a per-platform basis.

19.20.1.2 ACPI Enumeration

When support for non-CPU agent Intel RDT features is detected using CPUID, ACPI may be consulted for further details on the level of feature support, device structures behind various I/O ports, and the specific MMIO interfaces used to control a given device.

Non-CPU agent Intel RDT enumeration is via the “IRDT” ACPI table. For more information, refer to the “Intel® Resource Director Technology Architecture Specification.”

19.20.2 Non-CPU Agent Intel® RDT Feature Enable MSR

Before configuring non-CPU agent Intel RDT through MMIO, the feature should be enabled using the non-CPU agent Intel RDT Feature Enable MSR, IA32_L3_IO_RDT_CFG (MSR address 0C83H). As described in Section 19.20.1.1, the presence of one or more CPUID bits indicating support for one or more non-CPU agent Intel RDT features also indicates the presence of this MSR. This MSR may be used to enable the non-CPU agent Intel RDT features.

Two bits are defined in this MSR. Bit 0, when set, enables non-CPU agent RDT resource allocation features. Bit 1, when set, enables non-CPU agent Intel RDT monitoring features.

The L3 Non-CPU agent Intel RDT Monitoring Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource monitoring features are present.

The L3 Non-CPU agent Intel RDT Allocation Enable bit is supported if CPUID indicates that one or more non-CPU agent Intel RDT resource allocation features are present.

The default value is 0x0, so both classes of features are disabled by default. All bits not defined are reserved. Writing a non-zero value to any reserved bit will generate a General Protection Fault (#GP(0)).

This MSR is scoped at the L3 cache level and is cleared on system reset. It is expected that the software will configure this MSR consistently across all L3 caches that may be present on that package.

The definition of the IA32_L3_IO_RDT_CFG MSR is shown in Figure 19-40.

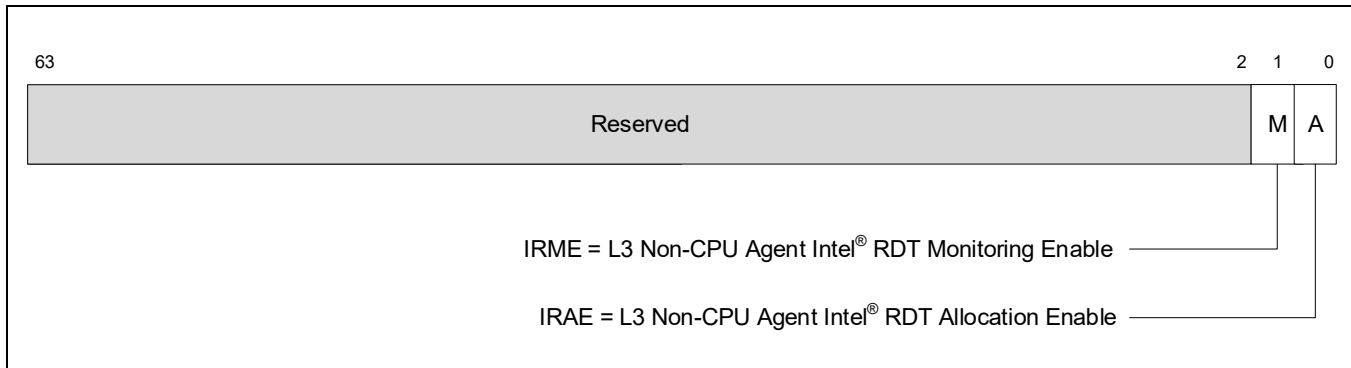


Figure 19-40. Layout of the IA32_L3_IO_RDT_CFG MSR for Enabling Non-CPU Agent Intel® RDT

19.21 CORE TELEMETRY FOR INTEL® PLATFORM MONITORING TECHNOLOGY (PMT)

Core telemetry is part of the Intel® Platform Monitoring Technology (Intel® PMT) feature set. Processors that support core telemetry collect per-core telemetry during core execution (including the execution context of the core — e.g., resource monitoring ID or RMID), aggregate the data internally, and report them to external agents, enabling workload-level tracking. This telemetry can be used by the external agents to monitor processor behavior to improve placement of workloads in data centers, increase resource utilization, monitor energy consumption, etc. The telemetry collected is processor-specific. For information about Intel® PMT refer to Intel® Platform Monitoring Technology (Intel® PMT) External Specification.

To avoid creating a side channel that might leak information from trusted execution environments (TEEs), core telemetry collected during TEE execution is discarded by the processor. Specifically, out-of-band core telemetry is discarded if it was collected while the core was running in any of the following situations:

- During execution of an enclave supported by Intel® Software Guard Extensions (Intel® SGX). This is true even for enclave debug following an opt-in entry.
- During execution of an authenticated code module.
- During execution in SEAM root operation.
- During execution in SEAM non-root operation, unless the “allow SEAM-guest telemetry” VM-entry control is 1.

For information about SEAM (secure arbitration mode), see Chapter 34.

18. Updates to Chapter 24, Volume 3B

Change bars and violet text show changes to Chapter 24 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide, Part 2*.

Changes to this chapter:

- Added UDB to Section 24.15, "Undefined Opcodes".

Intel 64 and IA-32 processors are binary compatible. Compatibility means that, within limited constraints, programs that execute on previous generations of processors will produce identical results when executed on later processors. The compatibility constraints and any implementation differences between the Intel 64 and IA-32 processors are described in this chapter.

Each new processor has enhanced the software visible architecture from that found in earlier Intel 64 and IA-32 processors. Those enhancements have been defined with consideration for compatibility with previous and future processors. This chapter also summarizes the compatibility considerations for those extensions.

24.1 PROCESSOR FAMILIES AND CATEGORIES

IA-32 processors are referred to in several different ways in this chapter, depending on the type of compatibility information being related, as described in the following:

- **IA-32 Processors** — All the Intel processors based on the Intel IA-32 Architecture, which include the 8086/88, Intel 286, Intel386, Intel486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Intel Xeon processors.
- **32-bit Processors** — All the IA-32 processors that use a 32-bit architecture, which include the Intel386, Intel486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Intel Xeon processors.
- **16-bit Processors** — All the IA-32 processors that use a 16-bit architecture, which include the 8086/88 and Intel 286 processors.
- **P6 Family Processors** — All the IA-32 processors that are based on the P6 microarchitecture, which include the Pentium Pro, Pentium II, and Pentium III processors.
- **Pentium® 4 Processors** — A family of IA-32 and Intel 64 processors that are based on the Intel NetBurst® microarchitecture.
- **Intel® Pentium® M Processors** — A family of IA-32 processors that are based on the Intel Pentium M processor microarchitecture.
- **Intel® Core™ Duo and Solo Processors** — Families of IA-32 processors that are based on an improved Intel Pentium M processor microarchitecture.
- **Intel® Xeon® Processors** — A family of IA-32 and Intel 64 processors that are based on the Intel NetBurst microarchitecture. This family includes the Intel Xeon processor and the Intel Xeon processor MP based on the Intel NetBurst microarchitecture. Intel Xeon processors 3000, 3100, 3200, 3300, 3200, 5100, 5200, 5300, 5400, 7200, 7300 series are based on Intel Core microarchitectures and support Intel 64 architecture.
- **Pentium® D Processors** — A family of dual-core Intel 64 processors that provides two processor cores in a physical package. Each core is based on the Intel NetBurst microarchitecture.
- **Pentium® Processor Extreme Editions** — A family of dual-core Intel 64 processors that provides two processor cores in a physical package. Each core is based on the Intel NetBurst microarchitecture and supports Intel Hyper-Threading Technology.
- **Intel® Core™ 2 Processor family** — A family of Intel 64 processors that are based on the Intel Core microarchitecture. Intel Pentium Dual-Core processors are also based on the Intel Core microarchitecture.
- **Intel Atom® Processors** — A family of IA-32 and Intel 64 processors. 45 nm Intel Atom processors are based on the Intel Atom microarchitecture. 32 nm Intel Atom processors are based on newer microarchitectures including the Silvermont microarchitecture and the Airmont microarchitecture. Each generation of Intel Atom processors can be identified by the CPUID's DisplayFamily_DisplayModel signature; see Table 2-1 "CPUID Signature Values of DisplayFamily_DisplayModel" in Chapter 2, "Model-Specific Registers (MSRs)," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4.

24.2 RESERVED BITS

Throughout this manual, certain bits are marked as reserved in many register and memory layout descriptions. When bits are marked as undefined or reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown effect. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers or memory locations that contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing them to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

Software written for existing IA-32 processor that handles reserved bits correctly will port to future IA-32 processors without generating protection exceptions.

24.3 ENABLING NEW FUNCTIONS AND MODES

Most of the new control functions defined for the P6 family and Pentium processors are enabled by new mode flags in the control registers (primarily register CR4). This register is undefined for IA-32 processors earlier than the Pentium processor. Attempting to access this register with an Intel486 or earlier IA-32 processor results in an invalid-opcode exception (#UD). Consequently, programs that execute correctly on the Intel486 or earlier IA-32 processor cannot erroneously enable these functions. Attempting to set a reserved bit in register CR4 to a value other than its original value results in a general-protection exception (#GP). So, programs that execute on the P6 family and Pentium processors cannot erroneously enable functions that may be implemented in future IA-32 processors.

The P6 family and Pentium processors do not check for attempts to set reserved bits in model-specific registers; however these bits may be checked on more recent processors. It is the obligation of the software writer to enforce this discipline. These reserved bits may be used in future Intel processors.

24.4 DETECTING THE PRESENCE OF NEW FEATURES THROUGH SOFTWARE

Software can check for the presence of new architectural features and extensions in either of two ways:

1. Test for the presence of the feature or extension. Software can test for the presence of new flags in the EFLAGS register and control registers. If these flags are reserved (meaning not present in the processor executing the test), an exception is generated. Likewise, software can attempt to execute a new instruction, which results in an invalid-opcode exception (#UD) being generated if it is not supported.
2. Execute the CPUID instruction. The CPUID instruction (added to the IA-32 in the Pentium processor) indicates the presence of new features directly.

See Chapter 21, "Processor Identification and Feature Determination," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for detailed information on detecting new processor features and extensions.

24.5 INTEL MMX TECHNOLOGY

The Pentium processor with MMX technology introduced the MMX technology and a set of MMX instructions to the IA-32. The MMX instructions are described in Chapter 9, "Programming with Intel® MMX™ Technology," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, and in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D. The MMX technology and MMX instructions are also included in the Pentium II, Pentium III, Pentium 4, and Intel Xeon processors.

24.6 STREAMING SIMD EXTENSIONS (SSE)

The Streaming SIMD Extensions (SSE) were introduced in the Pentium III processor. The SSE extensions consist of a new set of instructions and a new set of registers. The new registers include the eight 128-bit XMM registers and the 32-bit MXCSR control and status register. These instructions and registers are designed to allow SIMD computations to be made on single precision floating-point numbers. Several of these new instructions also operate in the MMX registers. SSE instructions and registers are described in Section 10, “Programming with Intel® Streaming SIMD Extensions (Intel® SSE),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, and in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D.

24.7 STREAMING SIMD EXTENSIONS 2 (SSE2)

The Streaming SIMD Extensions 2 (SSE2) were introduced in the Pentium 4 and Intel Xeon processors. They consist of a new set of instructions that operate on the XMM and MXCSR registers and perform SIMD operations on double precision floating-point values and on integer values. Several of these new instructions also operate in the MMX registers. SSE2 instructions and registers are described in Chapter 11, “Programming with Intel® Streaming SIMD Extensions 2 (Intel® SSE2),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, and in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D.

24.8 STREAMING SIMD EXTENSIONS 3 (SSE3)

The Streaming SIMD Extensions 3 (SSE3) were introduced in Pentium 4 processors supporting Intel Hyper-Threading Technology and Intel Xeon processors. SSE3 extensions include 13 instructions. Ten of these 13 instructions support the single instruction multiple data (SIMD) execution model used with SSE/SSE2 extensions. One SSE3 instruction accelerates x87 style programming for conversion to integer. The remaining two instructions (MONITOR and MWAIT) accelerate synchronization of threads. SSE3 instructions are described in Chapter 12, “Programming with Intel® SSE3, SSSE3, Intel® SSE4, and Intel® AES-NI,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, and in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D.

24.9 ADDITIONAL STREAMING SIMD EXTENSIONS

The Supplemental Streaming SIMD Extensions 3 (SSSE3) were introduced in the Intel Core 2 processor and Intel Xeon processor 5100 series. Streaming SIMD Extensions 4 provided 54 new instructions introduced in 45 nm Intel Xeon processors and Intel Core 2 processors. SSSE3, SSE4.1 and SSE4.2 instructions are described in Chapter 12, “Programming with Intel® SSE3, SSSE3, Intel® SSE4, and Intel® AES-NI,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, and in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D.

24.10 INTEL HYPER-THREADING TECHNOLOGY

Intel Hyper-Threading Technology provides two logical processors that can execute two separate code streams (called *threads*) concurrently by using shared resources in a single processor core or in a physical package.

This feature was introduced in the Intel Xeon processor MP and later steppings of the Intel Xeon processor, and Pentium 4 processors supporting Intel Hyper-Threading Technology. The feature is also found in the Pentium processor Extreme Edition. See also: Section 10.7, “Intel® Hyper-Threading Technology Architecture.”

45 nm and 32 nm Intel Atom processors support Intel Hyper-Threading Technology.

Intel Atom processors based on Silvermont and Airmont microarchitectures do not support Intel Hyper-Threading Technology.

24.11 MULTI-CORE TECHNOLOGY

The Pentium D processor and Pentium processor Extreme Edition provide two processor cores in each physical processor package. See also: Section 10.5, “Intel® Hyper-Threading Technology and Intel® Multi-Core Technology,” and Section 10.8, “Multi-Core Architecture.” Intel Core 2 Duo, Intel Pentium Dual-Core processors, Intel Xeon processors 3000, 3100, 5100, 5200 series provide two processor cores in each physical processor package. Intel Core 2 Extreme, Intel Core 2 Quad processors, Intel Xeon processors 3200, 3300, 5300, 5400, 7300 series provide two processor cores in each physical processor package.

24.12 SPECIFIC FEATURES OF DUAL-CORE PROCESSOR

Dual-core processors may have some processor-specific features. Use CPUID feature flags to detect the availability features. Note the following:

- **CPUID Brand String** — On Pentium processor Extreme Edition, the process will report the correct brand string only after the correct microcode updates are loaded.
- **Enhanced Intel SpeedStep Technology** — This feature is supported in Pentium D processor but not in Pentium processor Extreme Edition.

24.13 NEW INSTRUCTIONS IN THE PENTIUM AND LATER IA-32 PROCESSORS

Table 24-1 identifies the instructions introduced into the IA-32 in the Pentium processor and later IA-32 processors.

24.13.1 Instructions Added Prior to the Pentium Processor

The following instructions were added in the Intel486 processor:

- BSWAP (byte swap) instruction.
- XADD (exchange and add) instruction.
- CMPXCHG (compare and exchange) instruction.
- INVD (invalidate cache) instruction.
- WBINVD (write-back and invalidate cache) instruction.
- INVLPG (invalidate TLB entry) instruction.

Table 24-1. New Instruction in the Pentium Processor and Later IA-32 Processors

Instruction	CPUID Identification Bits	Introduced In
CMOVcc (conditional move)	EDX, Bit 15	Pentium Pro processor
FCMOVcc (floating-point conditional move)	EDX, Bits 0 and 15	
FCOMI (floating-point compare and set EFLAGS)	EDX, Bits 0 and 15	
RDPMC (read performance monitoring counters)	EAX, Bits 8-11, set to 6H; see Note 1	
UD2 (undefined)	EAX, Bits 8-11, set to 6H	

Table 24-1. New Instruction in the Pentium Processor and Later IA-32 Processors (Contd.)

Instruction	CPUID Identification Bits	Introduced In
CMPXCHG8B (compare and exchange 8 bytes)	EDX, Bit 8	Pentium processor
CPUID (CPU identification)	None; see Note 2	
RDTSC (read time-stamp counter)	EDX, Bit 4	
RDMSR (read model-specific register)	EDX, Bit 5	
WRMSR (write model-specific register)	EDX, Bit 5	
MMX Instructions	EDX, Bit 23	

NOTES:

1. The RDPMS instruction was introduced in the P6 family of processors and added to later model Pentium processors. This instruction is model specific in nature and not architectural.
2. The CPUID instruction is available in all Pentium and P6 family processors and in later models of the Intel486 processors. The ability to set and clear the ID flag (bit 21) in the EFLAGS register indicates the availability of the CPUID instruction.

The following instructions were added in the Intel386 processor:

- LSS, LFS, and LGS (load SS, FS, and GS registers).
- Long-displacement conditional jumps.
- Single-bit instructions.
- Bit scan instructions.
- Double-shift instructions.
- Byte set on condition instruction.
- Move with sign/zero extension.
- Generalized multiply instruction.
- MOV to and from control registers.
- MOV to and from test registers (now obsolete).
- MOV to and from debug registers.
- RSM (resume from SMM). This instruction was introduced in the Intel386 SL and Intel486 SL processors.

The following instructions were added in the Intel 387 math coprocessor:

- FPREM1.
- FUCOM, FUCOMP, and FUCOMPP.

24.14 OBSOLETE INSTRUCTIONS

The MOV to and from test registers instructions were removed from the Pentium processor and future IA-32 processors. Execution of these instructions generates an invalid-opcode exception (#UD).

24.15 UNDEFINED OPCODES

All new instructions defined for Intel 64 and IA-32 processors use binary encodings that were reserved on earlier-generation processors. Generally, attempting to execute a reserved opcode results in an invalid-opcode (#UD) exception being generated. Consequently, programs that execute correctly on earlier-generation processors cannot erroneously execute these instructions and thereby produce unexpected results when executed on later Intel 64 processors.

For compatibility with prior generations, there are a few reserved opcodes which do not result in a #UD but rather result in the same behavior as certain defined instructions. In the interest of standardization, it is recommended

that software not use the opcodes given below but instead use those defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

The following items enumerate those reserved opcodes (referring in some cases to opcode groups as defined in Appendix A, "Opcode Map," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D).

- **Immediate Group 1** - When not in 64-bit mode, instructions encoded with opcode 82H result in the behavior of the corresponding instructions encoded with opcode 80H. Depending on the Op/Reg field of the ModR/M Byte, these opcodes are the byte forms of ADD, OR, ADC, SBB, AND, SUB, XOR, CMP. (In 64-bit mode, these opcodes cause a #UD.)
- **Shift Group 2 / 6** - Instructions encoded with opcodes C0H, C1H, D0H, D1H, D2H, and D3H with value 110B in the Op/Reg field (/6) of the ModR/M Byte result in the behavior of the corresponding instructions with value 100B in the Op/Reg field (/4). These are various forms of the SAL/SHL instruction.
- **Unary Group 3 / 1** - Instructions encoded with opcodes F6H and F7H with value 001B in the Op/Reg field (/01) of the ModR/M Byte result in the behavior of the corresponding instructions with value 000B in the Op/Reg field (/0). These are various forms of the TEST instruction.
- **Reserved NOP** - Instructions encoded with the opcode 0F0DH or with the opcodes 0F18H through 0F1FH result in the behavior of the NOP (No Operation) instruction, except for those opcodes defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D. The opcodes not so defined are considered "Reserved NOP" and may be used for future instructions which have no defined impact on existing architectural state. These reserved NOP opcodes are decoded with a ModR/M byte and typical instruction prefix options but still result in the behavior of the NOP instruction.
- **x87 Opcodes** - There are several groups of x87 opcodes which provide the same behavior as other x87 instructions. See Section 24.18.9 for the complete list.

There are a few reserved opcodes that provide unique behavior but do not provide capabilities that are not already available in the main instructions defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

- **D6H** - Definition depends on the mode:
 - When not in 64-bit mode, this opcode is SALC, "Set AL to Carry flag": IF (CF=1) THEN AL:=FF; ELSE AL:=0; FI.
 - In 64-bit mode, this opcode is UDB and generates an invalid-opcode exception (#UD).
- **x87 Opcodes** - There are a few x87 opcodes with subtly different behavior from existing x87 instructions. See Section 24.18.9 for details.

24.16 NEW FLAGS IN THE EFLAGS REGISTER

The section titled "EFLAGS Register" in Chapter 3, "Basic Execution Environment," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, shows the configuration of flags in the EFLAGS register for the P6 family processors. No new flags have been added to this register in the P6 family processors. The flags added to this register in the Pentium and Intel486 processors are described in the following sections.

The following flags were added to the EFLAGS register in the Pentium processor:

- VIF (virtual interrupt flag), bit 19.
- VIP (virtual interrupt pending), bit 20.
- ID (identification flag), bit 21.

The AC flag (bit 18) was added to the EFLAGS register in the Intel486 processor.

24.16.1 Using EFLAGS Flags to Distinguish Between 32-Bit IA-32 Processors

The following bits in the EFLAGS register that can be used to differentiate between the 32-bit IA-32 processors:

- Bit 18 (the AC flag) can be used to distinguish an Intel386 processor from the P6 family, Pentium, and Intel486 processors. Since it is not implemented on the Intel386 processor, it will always be clear.

- Bit 21 (the ID flag) indicates whether an application can execute the CUID instruction. The ability to set and clear this bit indicates that the processor is a P6 family or Pentium processor. The CUID instruction can then be used to determine which processor.
- Bits 19 (the VIF flag) and 20 (the VIP flag) will always be zero on processors that do not support virtual mode extensions, which includes all 32-bit processors prior to the Pentium processor.

See Chapter 21, “Processor Identification and Feature Determination,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, for more information on identifying processors.

24.17 STACK OPERATIONS AND USER SOFTWARE

This section identifies the differences in stack implementation between the various IA-32 processors.

24.17.1 PUSH SP

The P6 family, Pentium, Intel486, Intel386, and Intel 286 processors push a different value on the stack for a PUSH SP instruction than the 8086 processor. The 32-bit processors push the value of the SP register before it is decremented as part of the push operation; the 8086 processor pushes the value of the SP register after it is decremented. If the value pushed is important, replace PUSH SP instructions with the following three instructions:

```
PUSH BP
MOV BP, SP
XCHG BP, [BP]
```

This code functions as the 8086 processor PUSH SP instruction on the P6 family, Pentium, Intel486, Intel386, and Intel 286 processors.

24.17.2 EFLAGS Pushed on the Stack

The setting of the stored values of bits 12 through 15 (which includes the IOPL field and the NT flag) in the EFLAGS register by the PUSHF instruction, by interrupts, and by exceptions is different with the 32-bit IA-32 processors than with the 8086 and Intel 286 processors. The differences are as follows:

- 8086 processor—bits 12 through 15 are always set.
- Intel 286 processor—bits 12 through 15 are always cleared in real-address mode.
- 32-bit processors in real-address mode—bit 15 (reserved) is always cleared, and bits 12 through 14 have the last value loaded into them.

24.18 X87 FPU

This section addresses the issues that must be faced when porting floating-point software designed to run on earlier IA-32 processors and math coprocessors to a Pentium 4, Intel Xeon, P6 family, or Pentium processor with integrated x87 FPU. To software, a Pentium 4, Intel Xeon, or P6 family processor looks very much like a Pentium processor. Floating-point software which runs on a Pentium or Intel486 DX processor, or on an Intel486 SX processor/Intel 487 SX math coprocessor system or an Intel386 processor/Intel 387 math coprocessor system, will run with at most minor modifications on a Pentium 4, Intel Xeon, or P6 family processor. To port code directly from an Intel 286 processor/Intel 287 math coprocessor system or an Intel 8086 processor/8087 math coprocessor system to a Pentium 4, Intel Xeon, P6 family, or Pentium processor, certain additional issues must be addressed.

In the following sections, the term “32-bit x87 FPUs” refers to the P6 family, Pentium, and Intel486 DX processors, and to the Intel 487 SX and Intel 387 math coprocessors; the term “16-bit IA-32 math coprocessors” refers to the Intel 287 and 8087 math coprocessors.

24.18.1 Control Register CR0 Flags

The ET, NE, and MP flags in control register CR0 control the interface between the integer unit of an IA-32 processor and either its internal x87 FPU or an external math coprocessor. The effect of these flags in the various IA-32 processors are described in the following paragraphs.

The ET (extension type) flag (bit 4 of the CR0 register) is used in the Intel386 processor to indicate whether the math coprocessor in the system is an Intel 287 math coprocessor (flag is clear) or an Intel 387 DX math coprocessor (flag is set). This bit is hardwired to 1 in the P6 family, Pentium, and Intel486 processors.

The NE (Numeric Exception) flag (bit 5 of the CR0 register) is used in the P6 family, Pentium, and Intel486 processors to determine whether unmasked floating-point exceptions are reported internally through interrupt vector 16 (flag is set) or externally through an external interrupt (flag is clear). On a hardware reset, the NE flag is initialized to 0, so software using the automatic internal error-reporting mechanism must set this flag to 1. This flag is nonexistent on the Intel386 processor.

As on the Intel 286 and Intel386 processors, the MP (monitor coprocessor) flag (bit 1 of register CR0) determines whether the WAIT/FWAIT instructions or waiting-type floating-point instructions trap when the context of the x87 FPU is different from that of the currently-executing task. If the MP and TS flag are set, then a WAIT/FWAIT instruction and waiting instructions will cause a device-not-available exception (interrupt vector 7). The MP flag is used on the Intel 286 and Intel386 processors to support the use of a WAIT/FWAIT instruction to wait on a device other than a math coprocessor. The device reports its status through the BUSY# pin. Since the P6 family, Pentium, and Intel486 processors do not have such a pin, the MP flag has no relevant use and should be set to 1 for normal operation.

24.18.2 x87 FPU Status Word

This section identifies differences to the x87 FPU status word for the different IA-32 processors and math coprocessors, the reason for the differences, and their impact on software.

24.18.2.1 Condition Code Flags (C0 through C3)

The following information pertains to differences in the use of the condition code flags (C0 through C3) located in bits 8, 9, 10, and 14 of the x87 FPU status word.

After execution of an FINIT instruction or a hardware reset on a 32-bit x87 FPU, the condition code flags are set to 0. The same operations on a 16-bit IA-32 math coprocessor leave these flags intact (they contain their prior value). This difference in operation has no impact on software and provides a consistent state after reset.

Transcendental instruction results in the core range of the P6 family and Pentium processors may differ from the Intel486 DX processor and Intel 487 SX math coprocessor by 2 to 3 units in the last place (ulps)—(see “Transcendental Instruction Accuracy” in Chapter 8, “Programming with the x87 FPU,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). As a result, the value saved in the C1 flag may also differ.

After an incomplete FPREM/FPREM1 instruction, the C0, C1, and C3 flags are set to 0 on the 32-bit x87 FPU. After the same operation on a 16-bit IA-32 math coprocessor, these flags are left intact.

On the 32-bit x87 FPU, the C2 flag serves as an incomplete flag for the FTAN instruction. On the 16-bit IA-32 math coprocessors, the C2 flag is undefined for the FPTAN instruction. This difference has no impact on software, because Intel 287 or 8087 programs do not check C2 after an FPTAN instruction. The use of this flag on later processors allows fast checking of operand range.

24.18.2.2 Stack Fault Flag

When unmasked stack overflow or underflow occurs on a 32-bit x87 FPU, the IE flag (bit 0) and the SF flag (bit 6) of the x87 FPU status word are set to indicate a stack fault and condition code flag C1 is set or cleared to indicate overflow or underflow, respectively. When unmasked stack overflow or underflow occurs on a 16-bit IA-32 math coprocessor, only the IE flag is set. Bit 6 is reserved on these processors. The addition of the SF flag on a 32-bit x87 FPU has no impact on software. Existing exception handlers need not change, but may be upgraded to take advantage of the additional information.

24.18.3 x87 FPU Control Word

Only affine closure is supported for infinity control on a 32-bit x87 FPU. The infinity control flag (bit 12 of the x87 FPU control word) remains programmable on these processors, but has no effect. This change was made to conform to the IEEE Standard 754 for Floating-Point Arithmetic. On a 16-bit IA-32 math coprocessor, both affine and projective closures are supported, as determined by the setting of bit 12. After a hardware reset, the default value of bit 12 is projective. Software that requires projective infinity arithmetic may give different results.

24.18.4 x87 FPU Tag Word

When loading the tag word of a 32-bit x87 FPU, using an `FLDENV`, `FRSTOR`, or `FXRSTOR` (Pentium III processor only) instruction, the processor examines the incoming tag and classifies the location only as empty or non-empty. Thus, tag values of 00, 01, and 10 are interpreted by the processor to indicate a non-empty location. The tag value of 11 is interpreted by the processor to indicate an empty location. Subsequent operations on a non-empty register always examine the value in the register, not the value in its tag. The `FSTENV`, `FSAVE`, and `FXSAVE` (Pentium III processor only) instructions examine the non-empty registers and put the correct values in the tags before storing the tag word.

The corresponding tag for a 16-bit IA-32 math coprocessor is checked before each register access to determine the class of operand in the register; the tag is updated after every change to a register so that the tag always reflects the most recent status of the register. Software can load a tag with a value that disagrees with the contents of a register (for example, the register contains a valid value, but the tag says special). Here, the 16-bit IA-32 math coprocessors honor the tag and do not examine the register.

Software written to run on a 16-bit IA-32 math coprocessor may not operate correctly on a 16-bit x87 FPU, if it uses the `FLDENV`, `FRSTOR`, or `FXRSTOR` instructions to change tags to values (other than to empty) that are different from actual register contents.

The encoding in the tag word for the 32-bit x87 FPUs for unsupported data formats (including pseudo-zero and unnormal) is special (10B), to comply with IEEE Standard 754. The encoding in the 16-bit IA-32 math coprocessors for pseudo-zero and unnormal is valid (00B) and the encoding for other unsupported data formats is special (10B). Code that recognizes the pseudo-zero or unnormal format as valid must therefore be changed if it is ported to a 32-bit x87 FPU.

24.18.5 Data Types

This section discusses the differences of data types for the various x87 FPUs and math coprocessors.

24.18.5.1 NaNs

The 32-bit x87 FPUs distinguish between signaling NaNs (SNaNs) and quiet NaNs (QNaNs). These x87 FPUs only generate QNaNs and normally do not generate an exception upon encountering a QNaN. An invalid-operation exception (#1) is generated only upon encountering a SNaN, except for the `FCOM`, `FIST`, and `FBSTP` instructions, which also generates an invalid-operation exceptions for a QNaNs. This behavior matches IEEE Standard 754.

The 16-bit IA-32 math coprocessors only generate one kind of NaN (the equivalent of a QNaN), but the raise an invalid-operation exception upon encountering any kind of NaN.

When porting software written to run on a 16-bit IA-32 math coprocessor to a 32-bit x87 FPU, uninitialized memory locations that contain QNaNs should be changed to SNaNs to cause the x87 FPU or math coprocessor to fault when uninitialized memory locations are referenced.

24.18.5.2 Pseudo-zero, Pseudo-NaN, Pseudo-infinity, and Unnormal Formats

The 32-bit x87 FPUs neither generate nor support the pseudo-zero, pseudo-NaN, pseudo-infinity, and unnormal formats. Whenever they encounter them in an arithmetic operation, they raise an invalid-operation exception. The 16-bit IA-32 math coprocessors define and support special handling for these formats. Support for these formats was dropped to conform with IEEE Standard 754 for Floating-Point Arithmetic.

This change should not impact software ported from 16-bit IA-32 math coprocessors to 32-bit x87 FPU. The 32-bit x87 FPU does not generate these formats, and therefore will not encounter them unless software explicitly loads them in the data registers. The only affect may be in how software handles the tags in the tag word (see also: Section 24.18.4, "x87 FPU Tag Word").

24.18.6 Floating-Point Exceptions

This section identifies the implementation differences in exception handling for floating-point instructions in the various x87 FPU and math coprocessors.

24.18.6.1 Denormal Operand Exception (#D)

When the denormal operand exception is masked, the 32-bit x87 FPU automatically normalize denormalized numbers when possible; whereas, the 16-bit IA-32 math coprocessors return a denormal result. A program written to run on a 16-bit IA-32 math coprocessor that uses the denormal exception solely to normalize denormalized operands is redundant when run on the 32-bit x87 FPU. If such a program is run on 32-bit x87 FPU, performance can be improved by masking the denormal exception. Floating-point programs run faster when the FPU performs normalization of denormalized operands.

The denormal operand exception is not raised for transcendental instructions and the FEXTRACT instruction on the 16-bit IA-32 math coprocessors. This exception is raised for these instructions on the 32-bit x87 FPU. The exception handlers ported to these latter processors need to be changed only if the handlers give special treatment to different opcodes.

24.18.6.2 Numeric Overflow Exception (#O)

On the 32-bit x87 FPU, when the numeric overflow exception is masked and the rounding mode is set to chop (toward 0), the result is the largest positive or smallest negative number. The 16-bit IA-32 math coprocessors do not signal the overflow exception when the masked response is not ∞ ; that is, they signal overflow only when the rounding control is not set to round to 0. If rounding is set to chop (toward 0), the result is positive or negative ∞ . Under the most common rounding modes, this difference has no impact on existing software.

If rounding is toward 0 (chop), a program on a 32-bit x87 FPU produces, under overflow conditions, a result that is different in the least significant bit of the significand, compared to the result on a 16-bit IA-32 math coprocessor. The reason for this difference is IEEE Standard 754 compatibility.

When the overflow exception is not masked, the precision exception is flagged on the 32-bit x87 FPU. When the result is stored in the stack, the significand is rounded according to the precision control (PC) field of the FPU control word or according to the opcode. On the 16-bit IA-32 math coprocessors, the precision exception is not flagged and the significand is not rounded. The impact on existing software is that if the result is stored on the stack, a program running on a 32-bit x87 FPU produces a different result under overflow conditions than on a 16-bit IA-32 math coprocessor. The difference is apparent only to the exception handler. This difference is for IEEE Standard 754 compatibility.

24.18.6.3 Numeric Underflow Exception (#U)

When the underflow exception is masked on the 32-bit x87 FPU, the underflow exception is signaled when the result is tiny and inexact (see Section 4.9.1.5, "Numeric Underflow Exception (#U)," in Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1). When the underflow exception is unmasked and the instruction is supposed to store the result on the stack, the significand is rounded to the appropriate precision (according to the PC flag in the FPU control word, for those instructions controlled by PC, otherwise to extended precision), after adjusting the exponent.

24.18.6.4 Exception Precedence

There is no difference in the precedence of the denormal-operand exception on the 32-bit x87 FPU, whether it be masked or not. When the denormal-operand exception is not masked on the 16-bit IA-32 math coprocessors, it takes precedence over all other exceptions. This difference causes no impact on existing software, but some

unnneeded normalization of denormalized operands is prevented on the Intel486 processor and Intel 387 math coprocessor.

24.18.6.5 CS and EIP For FPU Exceptions

On the Intel 32-bit x87 FPU, the values from the CS and EIP registers saved for floating-point exceptions point to any prefixes that come before the floating-point instruction. On the 8087 math coprocessor, the saved CS and IP registers points to the floating-point instruction.

24.18.6.6 FPU Error Signals

The floating-point error signals to the P6 family, Pentium, and Intel486 processors do not pass through an interrupt controller; an INT# signal from an Intel 387, Intel 287 or 8087 math coprocessors does. If an 8086 processor uses another exception for the 8087 interrupt, both exception vectors should call the floating-point-error exception handler. Some instructions in a floating-point-error exception handler may need to be deleted if they use the interrupt controller. The P6 family, Pentium, and Intel486 processors have signals that, with the addition of external logic, support reporting for emulation of the interrupt mechanism used in many personal computers.

On the P6 family, Pentium, and Intel486 processors, an undefined floating-point opcode will cause an invalid-opcode exception (#UD, interrupt vector 6). Undefined floating-point opcodes, like legal floating-point opcodes, cause a device not available exception (#NM, interrupt vector 7) when either the TS or EM flag in control register CR0 is set. The P6 family, Pentium, and Intel486 processors do not check for floating-point error conditions on encountering an undefined floating-point opcode.

24.18.6.7 Assertion of the FERR# Pin

When using the MS-DOS compatibility mode for handling floating-point exceptions, the FERR# pin must be connected to an input to an external interrupt controller. An external interrupt is then generated when the FERR# output drives the input to the interrupt controller and the interrupt controller in turn drives the INTR pin on the processor.

For the P6 family and Intel386 processors, an unmasked floating-point exception always causes the FERR# pin to be asserted upon completion of the instruction that caused the exception. For the Pentium and Intel486 processors, an unmasked floating-point exception may cause the FERR# pin to be asserted either at the end of the instruction causing the exception or immediately before execution of the next floating-point instruction. (Note that the next floating-point instruction would not be executed until the pending unmasked exception has been handled.) See Appendix D, "Guidelines for Writing SIMD Floating-Point Exception Handlers," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for a complete description of the required mechanism for handling floating-point exceptions using the MS-DOS compatibility mode.

Using FERR# and IGNNE# to handle floating-point exception is deprecated by modern operating systems; this approach also limits newer processors to operate with one logical processor active.

24.18.6.8 Invalid Operation Exception On Denormals

An invalid-operation exception is not generated on the 32-bit x87 FPU upon encountering a denormal value when executing a FSQRT, FDIV, or FPREM instruction or upon conversion to BCD or to integer. The operation proceeds by first normalizing the value. On the 16-bit IA-32 math coprocessors, upon encountering this situation, the invalid-operation exception is generated. This difference has no impact on existing software. Software running on the 32-bit x87 FPU continues to execute in cases where the 16-bit IA-32 math coprocessors trap. The reason for this change was to eliminate an exception from being raised.

24.18.6.9 Alignment Check Exceptions (#AC)

If alignment checking is enabled, a misaligned data operand on the P6 family, Pentium, and Intel486 processors causes an alignment check exception (#AC) when a program or procedure is running at privilege-level 3, except for the stack portion of the FSAVE/FNSAVE, FXSAVE, FRSTOR, and FXRSTOR instructions.

24.18.6.10 Segment Not Present Exception During FLDENV

On the Intel486 processor, when a segment not present exception (#NP) occurs in the middle of an FLDENV instruction, it can happen that part of the environment is loaded and part not. In such cases, the FPU control word is left with a value of 007FH. The P6 family and Pentium processors ensure the internal state is correct at all times by attempting to read the first and last bytes of the environment before updating the internal state.

24.18.6.11 Device Not Available Exception (#NM)

The device-not-available exception (#NM, interrupt 7) will occur in the P6 family, Pentium, and Intel486 processors as described in Section 2.5, "Control Registers," Table 2-2, and Chapter 7, "Interrupt 7—Device Not Available Exception (#NM)."

24.18.6.12 Coprocessor Segment Overrun Exception

The coprocessor segment overrun exception (interrupt 9) does not occur in the P6 family, Pentium, and Intel486 processors. In situations where the Intel 387 math coprocessor would cause an interrupt 9, the P6 family, Pentium, and Intel486 processors simply abort the instruction. To avoid undetected segment overruns, it is recommended that the floating-point save area be placed in the same page as the TSS. This placement will prevent the FPU environment from being lost if a page fault occurs during the execution of an FLDENV, FRSTOR, or FXRSTOR instruction while the operating system is performing a task switch.

24.18.6.13 General Protection Exception (#GP)

A general-protection exception (#GP, interrupt 13) occurs if the starting address of a floating-point operand falls outside a segment's size. An exception handler should be included to report these programming errors.

24.18.6.14 Floating-Point Error Exception (#MF)

In real mode and protected mode (not including virtual-8086 mode), interrupt vector 16 must point to the floating-point exception handler. In virtual-8086 mode, the virtual-8086 monitor can be programmed to accommodate a different location of the interrupt vector for floating-point exceptions.

24.18.7 Changes to Floating-Point Instructions

This section identifies the differences in floating-point instructions for the various Intel FPU and math coprocessor architectures, the reason for the differences, and their impact on software.

24.18.7.1 FDIV, FPREM, and FSQRT Instructions

The 32-bit x87 FPUs support operations on denormalized operands and, when detected, an underflow exception can occur, for compatibility with the IEEE Standard 754. The 16-bit IA-32 math coprocessors do not operate on denormalized operands or return underflow results. Instead, they generate an invalid-operation exception when they detect an underflow condition. An existing underflow exception handler will require change only if it gives different treatment to different opcodes. Also, it is possible that fewer invalid-operation exceptions will occur.

24.18.7.2 FSCALE Instruction

With the 32-bit x87 FPUs, the range of the scaling operand is not restricted. If $(0 < |ST(1)| < 1)$, the scaling factor is 0; therefore, $ST(0)$ remains unchanged. If the rounded result is not exact or if there was a loss of accuracy (masked underflow), the precision exception is signaled. With the 16-bit IA-32 math coprocessors, the range of the scaling operand is restricted. If $(0 < |ST(1)| < 1)$, the result is undefined and no exception is signaled. The impact of this difference on exiting software is that different results are delivered on the 32-bit and 16-bit FPUs and math coprocessors when $(0 < |ST(1)| < 1)$.

24.18.7.3 FPREM1 Instruction

The 32-bit x87 FPU's compute a partial remainder according to IEEE Standard 754. This instruction does not exist on the 16-bit IA-32 math coprocessors. The availability of the FPREM1 instruction has no impact on existing software.

24.18.7.4 FPREM Instruction

On the 32-bit x87 FPU's, the condition code flags C0, C3, C1 in the status word correctly reflect the three low-order bits of the quotient following execution of the FPREM instruction. On the 16-bit IA-32 math coprocessors, the quotient bits are incorrect when performing a reduction of $(64^N + M)$ when $(N \geq 1)$ and M is 1 or 2. This difference does not affect existing software; software that works around the bug should not be affected.

24.18.7.5 FUCOM, FUCOMP, and FUCOMPP Instructions

When executing the FUCOM, FUCOMP, and FUCOMPP instructions, the 32-bit x87 FPU's perform unordered compare according to IEEE Standard 754. These instructions do not exist on the 16-bit IA-32 math coprocessors. The availability of these new instructions has no impact on existing software.

24.18.7.6 FPTAN Instruction

On the 32-bit x87 FPU's, the range of the operand for the FPTAN instruction is much less restricted ($|ST(0)| < 2^{63}$) than on earlier math coprocessors. The instruction reduces the operand internally using an internal $\pi/4$ constant that is more accurate. The range of the operand is restricted to $(|ST(0)| < \pi/4)$ on the 16-bit IA-32 math coprocessors; the operand must be reduced to this range using FPREM. This change has no impact on existing software. See also sections 8.3.8 and section 8.3.10 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for more information on the accuracy of the FPTAN instruction.

24.18.7.7 Stack Overflow

On the 32-bit x87 FPU's, if an FPU stack overflow occurs when the invalid-operation exception is masked, the FPU returns the real, integer, or BCD-integer indefinite value to the destination operand, depending on the instruction being executed. On the 16-bit IA-32 math coprocessors, the original operand remains unchanged following a stack overflow, but it is loaded into register ST(1). This difference has no impact on existing software.

24.18.7.8 FSIN, FCOS, and FSINCOS Instructions

On the 32-bit x87 FPU's, these instructions perform three common trigonometric functions. These instructions do not exist on the 16-bit IA-32 math coprocessors. The availability of these instructions has no impact on existing software, but using them provides a performance upgrade. See also sections 8.3.8 and section 8.3.10 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, for more information on the accuracy of the FSIN, FCOS, and FSINCOS instructions.

24.18.7.9 FPATAN Instruction

On the 32-bit x87 FPU's, the range of operands for the FPATAN instruction is unrestricted. On the 16-bit IA-32 math coprocessors, the absolute value of the operand in register ST(0) must be smaller than the absolute value of the operand in register ST(1). This difference has impact on existing software.

24.18.7.10 F2XM1 Instruction

The 32-bit x87 FPU's support a wider range of operands $(-1 < ST(0) < +1)$ for the F2XM1 instruction. The supported operand range for the 16-bit IA-32 math coprocessors is $(0 \leq ST(0) \leq 0.5)$. This difference has no impact on existing software.

24.18.7.11 FLD Instruction

On the 32-bit x87 FPU, when using the FLD instruction to load an extended-real value, a denormal-operand exception is not generated because the instruction is not arithmetic. The 16-bit IA-32 math coprocessors do report a denormal-operand exception in this situation. This difference does not affect existing software.

On the 32-bit x87 FPU, loading a denormal value that is in single- or double-real format causes the value to be converted to extended-real format. Loading a denormal value on the 16-bit IA-32 math coprocessors causes the value to be converted to an unnormal. If the next instruction is FXTRACT or FXAM, the 32-bit x87 FPU will give a different result than the 16-bit IA-32 math coprocessors. This change was made for IEEE Standard 754 compatibility.

On the 32-bit x87 FPU, loading an SNaN that is in single- or double-real format causes the FPU to generate an invalid-operation exception. The 16-bit IA-32 math coprocessors do not raise an exception when loading a signaling NaN. The invalid-operation exception handler for 16-bit math coprocessor software needs to be updated to handle this condition when porting software to 32-bit FPU. This change was made for IEEE Standard 754 compatibility.

24.18.7.12 FXTRACT Instruction

On the 32-bit x87 FPU, if the operand is 0 for the FXTRACT instruction, the divide-by-zero exception is reported and $-\infty$ is delivered to register ST(1). If the operand is $+\infty$, no exception is reported. If the operand is 0 on the 16-bit IA-32 math coprocessors, 0 is delivered to register ST(1) and no exception is reported. If the operand is $+\infty$, the invalid-operation exception is reported. These differences have no impact on existing software. Software usually bypasses 0 and ∞ . This change is due to the IEEE Standard 754 recommendation to fully support the “logb” function.

24.18.7.13 Load Constant Instructions

On 32-bit x87 FPU, rounding control is in effect for the load constant instructions. Rounding control is not in effect for the 16-bit IA-32 math coprocessors. Results for the FLDPI, FLDLN2, FLDLG2, and FLDL2E instructions are the same as for the 16-bit IA-32 math coprocessors when rounding control is set to round to nearest or round to $+\infty$. They are the same for the FLDL2T instruction when rounding control is set to round to nearest, round to $-\infty$, or round to zero. Results are different from the 16-bit IA-32 math coprocessors in the least significant bit of the mantissa if rounding control is set to round to $-\infty$ or round to 0 for the FLDPI, FLDLN2, FLDLG2, and FLDL2E instructions; they are different for the FLDL2T instruction if round to $+\infty$ is specified. These changes were implemented for compatibility with IEEE Standard 754 for Floating-Point Arithmetic recommendations.

24.18.7.14 FXAM Instruction

With the 32-bit x87 FPU, if the FPU encounters an empty register when executing the FXAM instruction, it not generate combinations of C0 through C3 equal to 1101 or 1111. The 16-bit IA-32 math coprocessors may generate these combinations, among others. This difference has no impact on existing software; it provides a performance upgrade to provide repeatable results.

24.18.7.15 FSAVE and FSTENV Instructions

With the 32-bit x87 FPU, the address of a memory operand pointer stored by FSAVE or FSTENV is undefined if the previous floating-point instruction did not refer to memory

24.18.8 Transcendental Instructions

The floating-point results of the P6 family and Pentium processors for transcendental instructions in the core range may differ from the Intel486 processors by about 2 or 3 ulps (see “Transcendental Instruction Accuracy” in Chapter 8, “Programming with the x87 FPU,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). Condition code flag C1 of the status word may differ as a result. The exact threshold for underflow and overflow will vary by a few ulps. The P6 family and Pentium processors’ results will have a worst case error of less than 1 ulp when rounding to the nearest-even and less than 1.5 ulps when rounding in other modes. The transcendental

instructions are guaranteed to be monotonic, with respect to the input operands, throughout the domain supported by the instruction.

Transcendental instructions may generate different results in the round-up flag (C1) on the 32-bit x87 FPU. The round-up flag is undefined for these instructions on the 16-bit IA-32 math coprocessors. This difference has no impact on existing software.

24.18.9 Obsolete Instructions and Undefined Opcodes

The 8087 math coprocessor instructions FENI and FDISI, and the Intel 287 math coprocessor instruction FSETPM are treated as integer NOP instructions in the 32-bit x87 FPU. If these opcodes are detected in the instruction stream, no specific operation is performed and no internal states are affected. FSETPM informed the Intel 287 math coprocessor that the processor was in protected mode. The 32-bit x87 FPU handles all addressing and exception-pointer information, whether in protected mode or not.

For compatibility with prior generations there are a few reserved x87 opcodes which do not result in an invalid-opcode (#UD) exception, but rather result in the same behavior as existing defined x87 instructions. In the interest of standardization, it is recommended that the opcodes defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D, be used for these operations for standardization.

- DCD0H through DCD7H - Behaves the same as FCOM, D8D0H through D8D7H.
- DCD8H through DCDFH - Behaves the same as FCOMP, D8D8H through D8DFH.
- DDC8H through DDCFH - Behaves the same as FXCH, D9C8H through D9CFH.
- DED0H through DED7H - Behaves the same as FCOMP, D8D8H through D8DFH.
- DFD0H through DFD7H - Behaves the same as FSTP, DDD8H through DDDFH.
- DFC8H through DFCFH - Behaves the same as FXCH, D9C8H through D9CFH.
- DFD8H through DFDFH - Behaves the same as FSTP, DDD8H through DDDFH.

There are a few reserved x87 opcodes which provide unique behavior but do not provide capabilities which are not already available in the main instructions defined in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volumes 2A, 2B, 2C, & 2D.

- D9D8H through D9DFH - Behaves the same as FSTP (DDD8H through DDDFH) but won't cause a stack underflow exception.
- DFC0H through DFC7H - Behaves the same as FFREE (DDC0H through DDD7H) with the addition of an x87 stack POP.

24.18.10 WAIT/FWAIT Prefix Differences

On the Intel486 processor, when a WAIT/FWAIT instruction precedes a floating-point instruction (one which itself automatically synchronizes with the previous floating-point instruction), the WAIT/FWAIT instruction is treated as a no-op. Pending floating-point exceptions from a previous floating-point instruction are processed not on the WAIT/FWAIT instruction but on the floating-point instruction following the WAIT/FWAIT instruction. In such a case, the report of a floating-point exception may appear one instruction later on the Intel486 processor than on a P6 family or Pentium FPU, or on Intel 387 math coprocessor.

24.18.11 Operands Split Across Segments and/or Pages

On the P6 family, Pentium, and Intel486 processor FPUs, when the first half of an operand to be written is inside a page or segment and the second half is outside, a memory fault can cause the first half to be stored but not the second half. In this situation, the Intel 387 math coprocessor stores nothing.

24.18.12 FPU Instruction Synchronization

On the 32-bit x87 FPU, all floating-point instructions are automatically synchronized; that is, the processor automatically waits until the previous floating-point instruction has completed before completing the next floating-point

instruction. No explicit WAIT/FWAIT instructions are required to assure this synchronization. For the 8087 math coprocessors, explicit waits are required before each floating-point instruction to ensure synchronization. Although 8087 programs having explicit WAIT instructions execute perfectly on the 32-bit IA-32 processors without reassembly, these WAIT instructions are unnecessary.

24.19 SERIALIZING INSTRUCTIONS

Certain instructions have been defined to serialize instruction execution to ensure that modifications to flags, registers, and memory are completed before the next instruction is executed (or in P6 family processor terminology “committed to machine state”). Because the P6 family processors use branch-prediction and out-of-order execution techniques to improve performance, instruction execution is not generally serialized until the results of an executed instruction are committed to machine state (see Chapter 2, “Intel® 64 and IA-32 Architectures,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).

As a result, at places in a program or task where it is critical to have execution completed for all previous instructions before executing the next instruction (for example, at a branch, at the end of a procedure, or in multiprocessor dependent code), it is useful to add a serializing instruction. See Section 10.3, “Serializing Instructions,” for more information on serializing instructions.

24.20 FPU AND MATH COPROCESSOR INITIALIZATION

Table 11-1 shows the states of the FPU in the P6 family, Pentium, Intel486 processors and of the Intel 387 math coprocessor and Intel 287 coprocessor following a power-up, reset, or INIT, or following the execution of an FINIT/FNINIT instruction. The following is some additional compatibility information concerning the initialization of x87 FPU and math coprocessors.

24.20.1 Intel® 387 and Intel® 287 Math Coprocessor Initialization

Following an Intel386 processor reset, the processor identifies its coprocessor type (Intel® 287 or Intel® 387 DX math coprocessor) by sampling its ERROR# input some time after the falling edge of RESET# signal and before execution of the first floating-point instruction. The Intel 287 coprocessor keeps its ERROR# output in inactive state after hardware reset; the Intel 387 coprocessor keeps its ERROR# output in active state after hardware reset.

Upon hardware reset or execution of the FINIT/FNINIT instruction, the Intel 387 math coprocessor signals an error condition. The P6 family, Pentium, and Intel486 processors, like the Intel 287 coprocessor, do not.

24.20.2 Intel486 SX Processor and Intel 487 SX Math Coprocessor Initialization

When initializing an Intel486 SX processor and an Intel 487 SX math coprocessor, the initialization routine should check the presence of the math coprocessor and should set the FPU related flags (EM, MP, and NE) in control register CR0 accordingly (see Section 2.5, “Control Registers,” for a complete description of these flags). Table 24-2 gives the recommended settings for these flags when the math coprocessor is present. The FSTCW instruction will give a value of FFFFH for the Intel486 SX microprocessor and 037FH for the Intel 487 SX math coprocessor.

Table 24-2. Recommended Values of the EM, MP, and NE Flags for Intel486 SX Microprocessor/Intel 487 SX Math Coprocessor System

CR0 Flags	Intel486 SX Processor Only	Intel 487 SX Math Coprocessor Present
EM	1	0
MP	0	1
NE	1	0, for MS-DOS* systems 1, for user-defined exception handler

The EM and MP flags in register CR0 are interpreted as shown in Table 24-3.

Table 24-3. EM and MP Flag Interpretation

EM	MP	Interpretation
0	0	Floating-point instructions are passed to FPU; WAIT/FWAIT and other waiting-type instructions ignore TS.
0	1	Floating-point instructions are passed to FPU; WAIT/FWAIT and other waiting-type instructions test TS.
1	0	Floating-point instructions trap to emulator; WAIT/FWAIT and other waiting-type instructions ignore TS.
1	1	Floating-point instructions trap to emulator; WAIT/FWAIT and other waiting-type instructions test TS.

Following is an example code sequence to initialize the system and check for the presence of Intel486 SX processor/Intel 487 SX math coprocessor.

```
fninit
fstcw mem_loc
mov ax, mem_loc
cmp ax, 037fh
jz Intel487_SX_Math_CoProcessor_present    ;ax=037fh
jmp Intel486_SX_microprocessor_present     ;ax=ffffh
```

If the Intel 487 SX math coprocessor is not present, the following code can be run to set the CR0 register for the Intel486 SX processor.

```
mov eax, cr0
and eax, ffffffffh ;make MP=0
or  eax, 0024h     ;make EM=1, NE=1
mov cr0, eax
```

This initialization will cause any floating-point instruction to generate a device not available exception (#NM), interrupt 7. The software emulation will then take control to execute these instructions. This code is not required if an Intel 487 SX math coprocessor is present in the system. In that case, the typical initialization routine for the Intel486 SX microprocessor will be adequate.

Also, when designing an Intel486 SX processor based system with an Intel 487 SX math coprocessor, timing loops should be independent of frequency and clocks per instruction. One way to attain this is to implement these loops in hardware and not in software (for example, BIOS).

24.21 CONTROL REGISTERS

The following sections identify the new control registers and control register flags and fields that were introduced to the 32-bit IA-32 in various processor families. See Figure 2-7 for the location of these flags and fields in the control registers.

The Pentium III processor introduced one new control flag in control register CR4:

- OSXMMEXCPT (bit 10) — The OS will set this bit if it supports unmasked SIMD floating-point exceptions.

The Pentium II processor introduced one new control flag in control register CR4:

- OSFXSR (bit 9) — The OS supports saving and restoring the Pentium III processor state during context switches.

The Pentium Pro processor introduced three new control flags in control register CR4:

- PAE (bit 5) — Physical address extension. Enables paging mechanism to reference extended physical addresses when set; restricts physical addresses to 32 bits when clear (see also: Section 24.22.1.1, “Physical Memory Addressing Extension”).
- PGE (bit 7) — Page global enable. Inhibits flushing of frequently-used or shared pages on CR3 writes (see also: Section 24.22.1.2, “Global Pages”).
- PCE (bit 8) — Performance-monitoring counter enable. Enables execution of the RDPMC instruction at any protection level.

The content of CR4 is 0H following a hardware reset.

Control register CR4 was introduced in the Pentium processor. This register contains flags that enable certain new extensions provided in the Pentium processor:

- VME — Virtual-8086 mode extensions. Enables support for a virtual interrupt flag in virtual-8086 mode (see Section 22.3, “Interrupt and Exception Handling in Virtual-8086 Mode”).
- PVI — Protected-mode virtual interrupts. Enables support for a virtual interrupt flag in protected mode (see Section 22.4, “Protected-Mode Virtual Interrupts”).
- TSD — Time-stamp disable. Restricts the execution of the RDTSC instruction to procedures running at privileged level 0.
- DE — Debugging extensions. Causes an undefined opcode (#UD) exception to be generated when debug registers DR4 and DR5 are references for improved performance (see Section 24.23.3, “Debug Registers DR4 and DR5”).
- PSE — Page size extensions. Enables 4-MByte pages with 32-bit paging when set (see Section 5.3, “32-Bit Paging”).
- MCE — Machine-check enable. Enables the machine-check exception, allowing exception handling for certain hardware error conditions (see Chapter 17, “Machine-Check Architecture”).

The Intel486 processor introduced five new flags in control register CR0:

- NE — Numeric error. Enables the normal mechanism for reporting floating-point numeric errors.
- WP — Write protect. Write-protects read-only pages against supervisor-mode accesses.
- AM — Alignment mask. Controls whether alignment checking is performed. Operates in conjunction with the AC (Alignment Check) flag.
- NW — Not write-through. Enables write-throughs and cache invalidation cycles when clear and disables invalidation cycles and write-throughs that hit in the cache when set.
- CD — Cache disable. Enables the internal cache when clear and disables the cache when set.

The Intel486 processor introduced two new flags in control register CR3:

- PCD — Page-level cache disable. The state of this flag is driven on the PCD# pin during bus cycles that are not paged, such as interrupt acknowledge cycles, when paging is enabled. The PCD# pin is used to control caching in an external cache on a cycle-by-cycle basis.
- PWT — Page-level write-through. The state of this flag is driven on the PWT# pin during bus cycles that are not paged, such as interrupt acknowledge cycles, when paging is enabled. The PWT# pin is used to control write through in an external cache on a cycle-by-cycle basis.

24.22 MEMORY MANAGEMENT FACILITIES

The following sections describe the new memory management facilities available in the various IA-32 processors and some compatibility differences.

24.22.1 New Memory Management Control Flags

The Pentium Pro processor introduced three new memory management features: physical memory addressing extension, the global bit in page-table entries, and general support for larger page sizes. These features are only available when operating in protected mode.

24.22.1.1 Physical Memory Addressing Extension

The new PAE (physical address extension) flag in control register CR4, bit 5, may enable additional address lines on the processor, allowing extended physical addresses. This option can only be used when paging is enabled, using a new page-table mechanism provided to support the larger physical address range (see Section 5.1, “Paging Modes and Control Bits”).

24.22.1.2 Global Pages

The new PGE (page global enable) flag in control register CR4, bit 7, provides a mechanism for preventing frequently used pages from being flushed from the translation lookaside buffer (TLB). When this flag is set, frequently used pages (such as pages containing kernel procedures or common data tables) can be marked global by setting the global flag in a page-directory or page-table entry.

On a task switch or a write to control register CR3 (which normally causes the TLBs to be flushed), the entries in the TLB marked global are not flushed. Marking pages global in this manner prevents unnecessary reloading of the TLB due to TLB misses on frequently used pages. See Section 5.10, “Caching Translation Information,” for a detailed description of this mechanism.

24.22.1.3 Larger Page Sizes

The P6 family processors support large page sizes. For 32-bit paging, this facility is enabled with the PSE (page size extension) flag in control register CR4, bit 4. When this flag is set, the processor supports either 4-KByte or 4-MByte page sizes. PAE paging and 4-level paging¹ support 2-MByte pages regardless of the value of CR4.PSE (see Section 5.4, “PAE Paging,” and Section 5.5, “4-Level Paging and 5-Level Paging”). See Chapter 5, “Paging,” for more information about large page sizes.

24.22.2 CD and NW Cache Control Flags

The CD and NW flags in control register CR0 were introduced in the Intel486 processor. In the P6 family and Pentium processors, these flags are used to implement a writeback strategy for the data cache; in the Intel486 processor, they implement a write-through strategy. See Table 13-5 for a comparison of these bits on the P6 family, Pentium, and Intel486 processors. For complete information on caching, see Chapter 13, “Memory Cache Control.”

24.22.3 Descriptor Types and Contents

Operating-system code that manages space in descriptor tables often contains an invalid value in the access-rights field of descriptor-table entries to identify unused entries. Access rights values of 80H and 00H remain invalid for the P6 family, Pentium, Intel486, Intel386, and Intel 286 processors. Other values that were invalid on the Intel 286 processor may be valid on the 32-bit processors because uses for these bits have been defined.

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

24.22.4 Changes in Segment Descriptor Loads

On the Intel386 processor, loading a segment descriptor always causes a locked read and write to set the accessed bit of the descriptor. On the P6 family, Pentium, and Intel486 processors, the locked read and write occur only if the bit is not already set.

24.23 DEBUG FACILITIES

The P6 family and Pentium processors include extensions to the Intel486 processor debugging support for breakpoints. To use the new breakpoint features, it is necessary to set the DE flag in control register CR4.

24.23.1 Differences in Debug Register DR6

It is not possible to write a 1 to reserved bit 12 in debug status register DR6 on the P6 family and Pentium processors; however, it is possible to write a 1 in this bit on the Intel486 processor. See Table 11-1 for the different setting of this register following a power-up or hardware reset.

24.23.2 Differences in Debug Register DR7

The P6 family and Pentium processors determines the type of breakpoint access by the R/W0 through R/W3 fields in debug control register DR7 as follows:

- 00 Break on instruction execution only.
- 01 Break on data writes only.
- 10 Undefined if the DE flag in control register CR4 is cleared; break on I/O reads or writes but not instruction fetches if the DE flag in control register CR4 is set.
- 11 Break on data reads or writes but not instruction fetches.

On the P6 family and Pentium processors, reserved bits 11, 12, 14, and 15 are hard-wired to 0. On the Intel486 processor, however, bit 12 can be set. See Table 11-1 for the different settings of this register following a power-up or hardware reset.

24.23.3 Debug Registers DR4 and DR5

Although the DR4 and DR5 registers are documented as reserved, previous generations of processors aliased references to these registers to debug registers DR6 and DR7, respectively. When debug extensions are not enabled (the DE flag in control register CR4 is cleared), the P6 family and Pentium processors remain compatible with existing software by allowing these aliased references. When debug extensions are enabled (the DE flag is set), attempts to reference registers DR4 or DR5 will result in an invalid-opcode exception (#UD).

24.24 RECOGNITION OF BREAKPOINTS

For the Pentium processor, it is recommended that debuggers execute the LGDT instruction before returning to the program being debugged to ensure that breakpoints are detected. This operation does not need to be performed on the P6 family, Intel486, or Intel386 processors.

The implementation of test registers on the Intel486 processor used for testing the cache and TLB has been redesigned using MSRs on the P6 family and Pentium processors. (Note that MSRs used for this function are different on the P6 family and Pentium processors.) The MOV to and from test register instructions generate invalid-opcode exceptions (#UD) on the P6 family processors.

24.25 EXCEPTIONS AND/OR EXCEPTION CONDITIONS

This section describes the new exceptions and exception conditions added to the 32-bit IA-32 processors and implementation differences in existing exception handling. See Chapter 7, “Interrupt and Exception Handling,” for a detailed description of the IA-32 exceptions.

The Pentium III processor introduced new state with the XMM registers. Computations involving data in these registers can produce exceptions. A new MXCSR control/status register is used to determine which exception or exceptions have occurred. When an exception associated with the XMM registers occurs, an interrupt is generated.

- SIMD floating-point exception (#XM, interrupt 19) — New exceptions associated with the SIMD floating-point registers and resulting computations.

No new exceptions were added with the Pentium Pro and Pentium II processors. The set of available exceptions is the same as for the Pentium processor. However, the following exception condition was added to the IA-32 with the Pentium Pro processor:

- Machine-check exception (#MC, interrupt 18) — New exception conditions. Many exception conditions have been added to the machine-check exception and a new architecture has been added for handling and reporting on hardware errors. See Chapter 17, “Machine-Check Architecture,” for a detailed description of the new conditions.

The following exceptions and/or exception conditions were added to the IA-32 with the Pentium processor:

- Machine-check exception (#MC, interrupt 18) — New exception. This exception reports parity and other hardware errors. It is a model-specific exception and may not be implemented or implemented differently in future processors. The MCE flag in control register CR4 enables the machine-check exception. When this bit is clear (which it is at reset), the processor inhibits generation of the machine-check exception.
- General-protection exception (#GP, interrupt 13) — New exception condition added. An attempt to write a 1 to a reserved bit position of a special register causes a general-protection exception to be generated.
- Page-fault exception (#PF, interrupt 14) — New exception condition added. When a 1 is detected in any of the reserved bit positions of a page-table entry, page-directory entry, or page-directory pointer during address translation, a page-fault exception is generated.

The following exception was added to the Intel486 processor:

- Alignment-check exception (#AC, interrupt 17) — New exception. Reports unaligned memory references when alignment checking is being performed.

The following exceptions and/or exception conditions were added to the Intel386 processor:

- Divide-error exception (#DE, interrupt 0)
 - Change in exception handling. Divide-error exceptions on the Intel386 processors always leave the saved CS:IP value pointing to the instruction that failed. On the 8086 processor, the CS:IP value points to the next instruction.
 - Change in exception handling. The Intel386 processors can generate the largest negative number as a quotient for the IDIV instruction (80H and 8000H). The 8086 processor generates a divide-error exception instead.
- Invalid-opcode exception (#UD, interrupt 6) — New exception condition added. Improper use of the LOCK instruction prefix can generate an invalid-opcode exception.
- Page-fault exception (#PF, interrupt 14) — New exception condition added. If paging is enabled in a 16-bit program, a page-fault exception can be generated as follows. Paging can be used in a system with 16-bit tasks if all tasks use the same page directory. Because there is no place in a 16-bit TSS to store the PDBR register, switching to a 16-bit task does not change the value of the PDBR register. Tasks ported from the Intel 286 processor should be given 32-bit TSSs so they can make full use of paging.
- General-protection exception (#GP, interrupt 13) — New exception condition added. The Intel386 processor sets a limit of 15 bytes on instruction length. The only way to violate this limit is by putting redundant prefixes before an instruction. A general-protection exception is generated if the limit on instruction length is violated. The 8086 processor has no instruction length limit.

24.25.1 Machine-Check Architecture

The Pentium Pro processor introduced a new architecture to the IA-32 for handling and reporting on machine-check exceptions. This machine-check architecture (described in detail in Chapter 17, “Machine-Check Architecture”) greatly expands the ability of the processor to report on internal hardware errors.

24.25.2 Priority of Exceptions

The priority of exceptions are broken down into several major categories:

1. Traps on the previous instruction
2. External interrupts
3. Faults on fetching the next instruction
4. Faults in decoding the next instruction
5. Faults on executing an instruction

There are no changes in the priority of these major categories between the different processors, however, exceptions within these categories are implementation dependent and may change from processor to processor.

24.25.3 Exception Conditions of Legacy SIMD Instructions Operating on MMX Registers

MMX instructions and a subset of SSE, SSE2, SSSE3 instructions operate on MMX registers. The exception conditions of these instructions are described in the following tables.

Table 24-4. Exception Conditions for Legacy SIMD/MMX Instructions with FP Exception and 16-Byte Alignment

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is ‘0’
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)	X	X	X	X	Legacy SSE: Memory operand is not 16-byte aligned
			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
#XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1
Applicable Instructions	CVTPD2PI, CVTTSD2PI				

Table 24-5. Exception Conditions for Legacy SIMD/MMX Instructions with XMM and FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 0.
	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
SIMD Floating-point Exception, #XM	X	X	X	X	If an unmasked SIMD floating-point exception and CR4.OSXMMEXCPT[bit 10] = 1
Applicable Instructions	CVTPI2PS, CVTPS2PI, CVTTPS2PI				

Table 24-6. Exception Conditions for Legacy SIMD/MMX Instructions with XMM and without FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1. If CR4.OSFXSR[bit 9] = 0.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF ¹	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	CVTPI2PD				

NOTES:

1. Applies to "CVTPI2PD xmm, mm" but not "CVTPI2PD xmm, m64".

Table 24-7. Exception Conditions for SIMD/MMX Instructions with Memory Reference

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
General Protection, #GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
Alignment Check #AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	PABSB, PABSD, PABSW, PACKSSWB, PACKSSDW, PACKUSWB, PADDB, PADDD, PADDQ, PADDW, PADDSB, PADDSW, PADDUSB, PADDUSW, PALIGNR, PAND, PANDN, PAVGB, PAVGW, PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW, PHADDD, PHADDW, PHADDSW, PHSUBD, PHSUBW, PHSUBSW, PINSRW, PMADDUSW, PMADDWD, PMAXSW, PMAXUB, PMINSW, PMINUB, PMULHRW, PMULHUW, PMULHW, PMULLW, PMULUDQ, PSADB, PSHUFB, PSHUFW, PSIGNB, PSIGND, PSIGNW, PSLLW, PSLLD, PSLLQ, PSRAD, PSRAW, PSRLW, PSRLD, PSRLQ, PSUBB, PSUBD, PSUBQ, PSUBW, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ, PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ, PXOR				

Table 24-8. Exception Conditions for Legacy SIMD/MMX Instructions without FP Exception

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1. If ModR/M.mod $\neq$ 11b ¹
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM	X	X	X	X	If CR0.TS[bit 3]=1
Stack, SS(0)			X		For an illegal address in the SS segment
				X	If a memory address referencing the SS segment is in a non-canonical form
#GP(0)			X		For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments. If the destination operand is in a non-writable segment. ² If the DS, ES, FS, or GS register contains a NULL segment selector. ³
				X	If the memory address is in a non-canonical form.
	X	X			If any part of the operand lies outside the effective address space from 0 to FFFFH
#PF(fault-code)		X	X	X	For a page fault
#AC(0)		X	X	X	If alignment checking is enabled and an unaligned memory reference is made while the current privilege level is 3.
Applicable Instructions	MASKMOVQ, MOVNTQ, "MOVQ (mmreg)"				

NOTES:

1. Applies to MASKMOVQ only.

2. Applies to MASKMOVQ and MOVQ (mmreg) only.

3. Applies to MASKMOVQ only.

Table 24-9. Exception Conditions for Legacy SIMD/MMX Instructions without Memory Reference

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X	X	If CR0.EM[bit 2] = 1.
	X	X	X	X	If preceded by a LOCK prefix (F0H)
	X	X	X	X	If any corresponding CPUID feature flag is '0'
#MF	X	X	X	X	If there is a pending X87 FPU exception
#NM			X	X	If CR0.TS[bit 3]=1
Applicable Instructions	PEXTRW, PMOVBMSKB				

24.26 INTERRUPTS

The following differences in handling interrupts are found among the IA-32 processors.

24.26.1 Interrupt Propagation Delay

External hardware interrupts may be recognized on different instruction boundaries on the P6 family, Pentium, Intel486, and Intel386 processors, due to the superscaler designs of the P6 family and Pentium processors. Therefore, the EIP pushed onto the stack when servicing an interrupt may be different for the P6 family, Pentium, Intel486, and Intel386 processors.

24.26.2 NMI Interrupts

After an NMI interrupt is recognized by the P6 family, Pentium, Intel486, Intel386, and Intel 286 processors, the NMI interrupt is masked until the first IRET instruction is executed, unlike the 8086 processor.

24.26.3 IDT Limit

The LIDT instruction can be used to set a limit on the size of the IDT. A double-fault exception (#DF) is generated if an interrupt or exception attempts to read a vector beyond the limit. Shutdown then occurs on the 32-bit IA-32 processors if the double-fault handler vector is beyond the limit. (The 8086 processor does not have a shutdown mode nor a limit.)

24.27 ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (APIC)

The Advanced Programmable Interrupt Controller (APIC), referred to in this book as the **local APIC**, was introduced into the IA-32 processors with the Pentium processor (beginning with the 735/90 and 815/100 models) and is included in the Pentium 4, Intel Xeon, and P6 family processors. The features and functions of the local APIC are derived from the Intel 82489DX external APIC, which was used with the Intel486 and early Pentium processors. Additional refinements of the local APIC architecture were incorporated in the Pentium 4 and Intel Xeon processors.

24.27.1 Software Visible Differences Between the Local APIC and the 82489DX

The following features in the local APIC features differ from those found in the 82489DX external APIC:

- When the local APIC is disabled by clearing the APIC software enable/disable flag in the spurious-interrupt vector MSR, the state of its internal registers are unaffected, except that the mask bits in the LVT are all set to block local interrupts to the processor. Also, the local APIC ceases accepting IPIs except for INIT, SMI, NMI, and start-up IPIs. In the 82489DX, when the local unit is disabled, all the internal registers including the IRR, ISR, and TMR are cleared and the mask bits in the LVT are set. In this state, the 82489DX local unit will accept only the reset deassert message.
- In the local APIC, NMI and INIT (except for INIT deassert) are always treated as edge triggered interrupts, even if programmed otherwise. In the 82489DX, these interrupts are always level triggered.
- In the local APIC, IPIs generated through the ICR are always treated as edge triggered (except INIT Deassert). In the 82489DX, the ICR can be used to generate either edge or level triggered IPIs.
- In the local APIC, the logical destination register supports 8 bits; in the 82489DX, it supports 32 bits.
- In the local APIC, the APIC ID register is 4 bits wide; in the 82489DX, it is 8 bits wide.
- The remote read delivery mode provided in the 82489DX and local APIC for Pentium processors is not supported in the local APIC in the Pentium 4, Intel Xeon, and P6 family processors.
- For the 82489DX, in the lowest priority delivery mode, all the target local APICs specified by the destination field participate in the lowest priority arbitration. For the local APIC, only those local APICs which have free interrupt slots will participate in the lowest priority arbitration.

24.27.2 New Features Incorporated in the Local APIC for the P6 Family and Pentium Processors

The local APIC in the Pentium and P6 family processors have the following new features not found in the 82489DX external APIC.

- Cluster addressing is supported in logical destination mode.
- Focus processor checking can be enabled/disabled.
- Interrupt input signal polarity can be programmed for the LINT0 and LINT1 pins.
- An SMI IPI is supported through the ICR and I/O redirection table.
- An error status register is incorporated into the LVT to log and report APIC errors.

In the P6 family processors, the local APIC incorporates an additional LVT register to handle performance monitoring counter interrupts.

24.27.3 New Features Incorporated in the Local APIC of the Pentium 4 and Intel Xeon Processors

The local APIC in the Pentium 4 and Intel Xeon processors has the following new features not found in the P6 family and Pentium processors and in the 82489DX.

- The local APIC ID is extended to 8 bits.
- An thermal sensor register is incorporated into the LVT to handle thermal sensor interrupts.
- The the ability to deliver lowest-priority interrupts to a focus processor is no longer supported.
- The flat cluster logical destination mode is not supported.

24.28 TASK SWITCHING AND TSS

This section identifies the implementation differences of task switching, additions to the TSS and the handling of TSSs and TSS segment selectors.

24.28.1 P6 Family and Pentium Processor TSS

When the virtual mode extensions are enabled (by setting the VME flag in control register CR4), the TSS in the P6 family and Pentium processors contain an interrupt redirection bit map, which is used in virtual-8086 mode to redirect interrupts back to an 8086 program.

24.28.2 TSS Selector Writes

During task state saves, the Intel486 processor writes 2-byte segment selectors into a 32-bit TSS, leaving the upper 16 bits undefined. For performance reasons, the P6 family and Pentium processors write 4-byte segment selectors into the TSS, with the upper 2 bytes being 0. For compatibility reasons, code should not depend on the value of the upper 16 bits of the selector in the TSS.

24.28.3 Order of Reads/Writes to the TSS

The order of reads and writes into the TSS is processor dependent. The P6 family and Pentium processors may generate different page-fault addresses in control register CR2 in the same TSS area than the Intel486 and Intel386 processors, if a TSS crosses a page boundary (which is not recommended).

24.28.4 Using A 16-Bit TSS with 32-Bit Constructs

Task switches using 16-bit TSSs should be used only for pure 16-bit code. Any new code written using 32-bit constructs (operands, addressing, or the upper word of the EFLAGS register) should use only 32-bit TSSs. This is due to the fact that the 32-bit processors do not save the upper 16 bits of EFLAGS to a 16-bit TSS. A task switch back to a 16-bit task that was executing in virtual mode will never re-enable the virtual mode, as this flag was not saved in the upper half of the EFLAGS value in the TSS. Therefore, it is strongly recommended that any code using 32-bit constructs use a 32-bit TSS to ensure correct behavior in a multitasking environment.

24.28.5 Differences in I/O Map Base Addresses

The Intel486 processor considers the TSS segment to be a 16-bit segment and wraps around the 64K boundary. Any I/O accesses check for permission to access this I/O address at the I/O base address plus the I/O offset. If the I/O map base address exceeds the specified limit of 0DFFFH, an I/O access will wrap around and obtain the permission for the I/O address at an incorrect location within the TSS. A TSS limit violation does not occur in this situation on the Intel486 processor. However, the P6 family and Pentium processors consider the TSS to be a 32-bit segment and a limit violation occurs when the I/O base address plus the I/O offset is greater than the TSS limit. By following the recommended specification for the I/O base address to be less than 0DFFFH, the Intel486 processor will not wrap around and access incorrect locations within the TSS for I/O port validation and the P6 family and Pentium processors will not experience general-protection exceptions (#GP). Figure 24-1 demonstrates the different areas accessed by the Intel486 and the P6 family and Pentium processors.

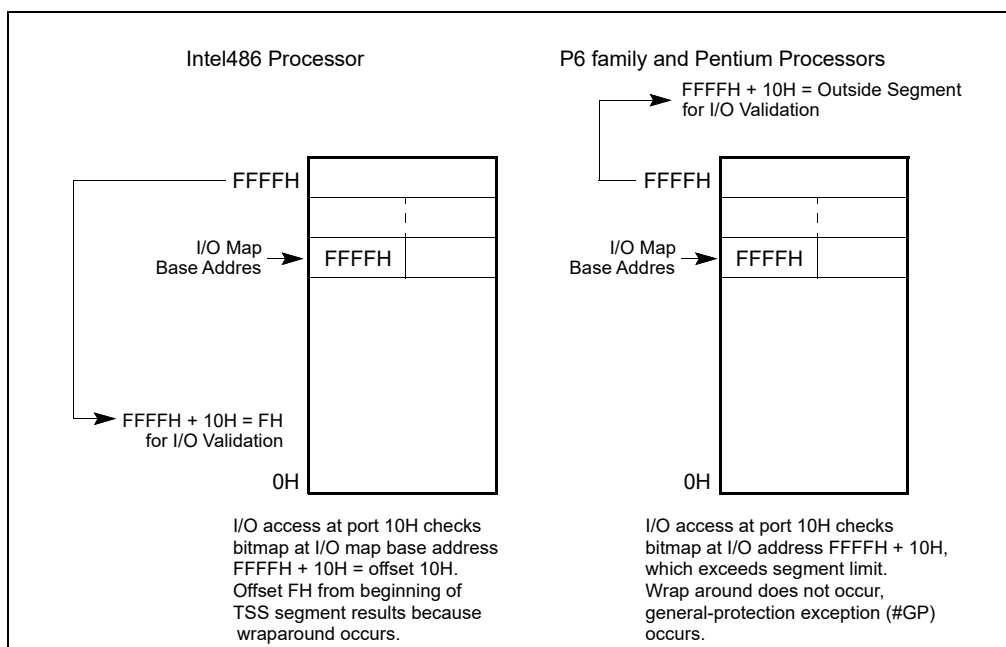


Figure 24-1. I/O Map Base Address Differences

24.29 CACHE MANAGEMENT

The P6 family processors include two levels of internal caches: L1 (level 1) and L2 (level 2). The L1 cache is divided into an instruction cache and a data cache; the L2 cache is a general-purpose cache. See Section 13.1, "Internal Caches, TLBs, and Buffers," for a description of these caches. (Note that although the Pentium II processor L2 cache is physically located on a separate chip in the cassette, it is considered an internal cache.)

The Pentium processor includes separate level 1 instruction and data caches. The data cache supports a writeback (or alternatively write-through, on a line by line basis) policy for memory updates.

The Intel486 processor includes a single level 1 cache for both instructions and data.

The meaning of the CD and NW flags in control register CR0 have been redefined for the P6 family and Pentium processors. For these processors, the recommended value (00B) enables writeback for the data cache of the Pentium processor and for the L1 data cache and L2 cache of the P6 family processors. In the Intel486 processor, setting these flags to (00B) enables write-through for the cache.

External system hardware can force the Pentium processor to disable caching or to use the write-through cache policy should that be required. In the P6 family processors, the MTRRs can be used to override the CD and NW flags (see Table 13-6).

The P6 family and Pentium processors support page-level cache management in the same manner as the Intel486 processor by using the PCD and PWT flags in control register CR3, the page-directory entries, and the page-table entries. The Intel486 processor, however, is not affected by the state of the PWT flag since the internal cache of the Intel486 processor is a write-through cache.

24.29.1 Self-Modifying Code with Cache Enabled

On the Intel486 processor, a write to an instruction in the cache will modify it in both the cache and memory. If the instruction was prefetched before the write, however, the old version of the instruction could be the one executed. To prevent this problem, it is necessary to flush the instruction prefetch unit of the Intel486 processor by coding a jump instruction immediately after any write that modifies an instruction. The P6 family and Pentium processors, however, check whether a write may modify an instruction that has been prefetched for execution. This check is based on the linear address of the instruction. If the linear address of an instruction is found to be present in the

prefetch queue, the P6 family and Pentium processors flush the prefetch queue, eliminating the need to code a jump instruction after any writes that modify an instruction.

Because the linear address of the write is checked against the linear address of the instructions that have been prefetched, special care must be taken for self-modifying code to work correctly when the physical addresses of the instruction and the written data are the same, but the linear addresses differ. In such cases, it is necessary to execute a serializing operation to flush the prefetch queue after the write and before executing the modified instruction. See Section 10.3, “Serializing Instructions,” for more information on serializing instructions.

NOTE

The check on linear addresses described above is not in practice a concern for compatibility. Applications that include self-modifying code use the same linear address for modifying and fetching the instruction. System software, such as a debugger, that might possibly modify an instruction using a different linear address than that used to fetch the instruction must execute a serializing operation, such as IRET, before the modified instruction is executed.

24.29.2 Disabling the L3 Cache

A unified third-level (L3) cache in processors based on Intel NetBurst microarchitecture (see Section 13.1, “Internal Caches, TLBs, and Buffers”) provides the third-level cache disable flag, bit 6 of the IA32_MISC_ENABLE MSR. The third-level cache disable flag allows the L3 cache to be disabled and enabled, independently of the L1 and L2 caches (see Section 13.5.4, “Disabling and Enabling the L3 Cache”). The third-level cache disable flag applies only to processors based on Intel NetBurst microarchitecture. Processors with L3 and based on other microarchitectures do not support the third-level cache disable flag.

24.30 PAGING

This section identifies enhancements made to the paging mechanism and implementation differences in the paging mechanism for various IA-32 processors.

24.30.1 Large Pages

The Pentium processor extended the memory management/paging facilities of the IA-32 to allow large (4 MBytes) pages sizes (see Section 5.3, “32-Bit Paging”). The first P6 family processor (the Pentium Pro processor) added a 2 MByte page size to the IA-32 in conjunction with the physical address extension (PAE) feature (see Section 5.4, “PAE Paging”).

The availability of large pages with 32-bit paging on any IA-32 processor can be determined via feature bit 3 (PSE) of register EDX after the CPUID instruction has been execution with an argument of 1. (Large pages are always available with PAE paging and 4-level paging.) Intel processors that do not support the CPUID instruction support only 32-bit paging and do not support page size enhancements. (See “CPUID—CPU Identification” in Chapter 3, “Instruction Set Reference, A-L,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A, for more information on the CPUID instruction.)

24.30.2 PCD and PWT Flags

The PCD and PWT flags were introduced to the IA-32 in the Intel486 processor to control the caching of pages:

- PCD (page-level cache disable) flag—Controls caching on a page-by-page basis.
- PWT (page-level write-through) flag—Controls the write-through/writeback caching policy on a page-by-page basis. Since the internal cache of the Intel486 processor is a write-through cache, it is not affected by the state of the PWT flag.

24.30.3 Enabling and Disabling Paging

Paging is enabled and disabled by loading a value into control register CR0 that modifies the PG flag. For backward and forward compatibility with all IA-32 processors, Intel recommends that the following operations be performed when enabling or disabling paging:

1. Execute a MOV CR0, REG instruction to either set (enable paging) or clear (disable paging) the PG flag.
2. Execute a near JMP instruction.

The sequence bounded by the MOV and JMP instructions should be identity mapped (that is, the instructions should reside on a page whose linear and physical addresses are identical).

For the P6 family processors, the MOV CR0, REG instruction is serializing, so the jump operation is not required. However, for backwards compatibility, the JMP instruction should still be included.

24.31 STACK OPERATIONS AND SUPERVISOR SOFTWARE

This section identifies the differences in the stack mechanism for the various IA-32 processors.

24.31.1 Selector Pushes and Pops

When pushing a segment selector onto the stack, the Pentium 4, Intel Xeon, P6 family, and Intel486 processors decrement the ESP register by the operand size and then write 2 bytes. If the operand size is 32-bits, the upper two bytes of the write are not modified. The Pentium processor decrements the ESP register by the operand size and determines the size of the write by the operand size. If the operand size is 32-bits, the upper two bytes are written as 0s.

When popping a segment selector from the stack, the Pentium 4, Intel Xeon, P6 family, and Intel486 processors read 2 bytes and increment the ESP register by the operand size of the instruction. The Pentium processor determines the size of the read from the operand size and increments the ESP register by the operand size.

It is possible to align a 32-bit selector push or pop such that the operation generates an exception on a Pentium processor and not on an Pentium 4, Intel Xeon, P6 family, or Intel486 processor. This could occur if the third and/or fourth byte of the operation lies beyond the limit of the segment or if the third and/or fourth byte of the operation is located on a non-present or inaccessible page.

For a POP-to-memory instruction that meets the following conditions:

- The stack segment size is 16-bit.
- Any 32-bit addressing form with the SIB byte specifying ESP as the base register.
- The initial stack pointer is FFFCH (32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0H as a result of the POP operation.

The result of the memory write is implementation-specific. For example, in P6 family processors, the result of the memory write is SS:0H plus any scaled index and displacement. In Pentium processors, the result of the memory write may be either a stack fault (real mode or protected mode with stack segment size of 64 KByte), or write to SS:10000H plus any scaled index and displacement (protected mode and stack segment size exceeds 64 KByte).

24.31.2 Error Code Pushes

The Intel486 processor implements the error code pushed on the stack as a 16-bit value. When pushed onto a 32-bit stack, the Intel486 processor only pushes 2 bytes and updates ESP by 4. The P6 family and Pentium processors' error code is a full 32 bits with the upper 16 bits set to zero. The P6 family and Pentium processors, therefore, push 4 bytes and update ESP by 4. Any code that relies on the state of the upper 16 bits may produce inconsistent results.

24.31.3 Fault Handling Effects on the Stack

During the handling of certain instructions, such as CALL and PUSH, faults may occur in different sequences for the different processors. For example, during far calls, the Intel486 processor pushes the old CS and EIP before a possible branch fault is resolved. A branch fault is a fault from a branch instruction occurring from a segment limit or access rights violation. If a branch fault is taken, the Intel486 and P6 family processors will have corrupted memory below the stack pointer. However, the ESP register is backed up to make the instruction restartable. The P6 family processors issue the branch before the pushes. Therefore, if a branch fault does occur, these processors do not corrupt memory below the stack pointer. This implementation difference, however, does not constitute a compatibility problem, as only values at or above the stack pointer are considered to be valid. Other operations that encounter faults may also corrupt memory below the stack pointer and this behavior may vary on different implementations.

24.31.4 Interlevel RET/IRET From a 16-Bit Interrupt or Call Gate

If a call or interrupt is made from a 32-bit stack environment through a 16-bit gate, only 16 bits of the old ESP can be pushed onto the stack. On the subsequent RET/IRET, the 16-bit ESP is popped but the full 32-bit ESP is updated since control is being resumed in a 32-bit stack environment. The Intel486 processor writes the SS selector into the upper 16 bits of ESP. The P6 family and Pentium processors write zeros into the upper 16 bits.

24.32 MIXING 16- AND 32-BIT SEGMENTS

The features of the 16-bit Intel 286 processor are an object-code compatible subset of those of the 32-bit IA-32 processors. The D (default operation size) flag in segment descriptors indicates whether the processor treats a code or data segment as a 16-bit or 32-bit segment; the B (default stack size) flag in segment descriptors indicates whether the processor treats a stack segment as a 16-bit or 32-bit segment.

The segment descriptors used by the Intel 286 processor are supported by the 32-bit IA-32 processors if the Intel-reserved word (highest word) of the descriptor is clear. On the 32-bit IA-32 processors, this word includes the upper bits of the base address and the segment limit.

The segment descriptors for data segments, code segments, local descriptor tables (there are no descriptors for global descriptor tables), and task gates are the same for the 16- and 32-bit processors. Other 16-bit descriptors (TSS segment, call gate, interrupt gate, and trap gate) are supported by the 32-bit processors.

The 32-bit processors also have descriptors for TSS segments, call gates, interrupt gates, and trap gates that support the 32-bit architecture. Both kinds of descriptors can be used in the same system.

For those segment descriptors common to both 16- and 32-bit processors, clear bits in the reserved word cause the 32-bit processors to interpret these descriptors exactly as an Intel 286 processor does, that is:

- Base Address — The upper 8 bits of the 32-bit base address are clear, which limits base addresses to 24 bits.
- Limit — The upper 4 bits of the limit field are clear, restricting the value of the limit field to 64 KBytes.
- Granularity bit — The G (granularity) flag is clear, indicating the value of the 16-bit limit is interpreted in units of 1 byte.
- Big bit — In a data-segment descriptor, the B flag is clear in the segment descriptor used by the 32-bit processors, indicating the segment is no larger than 64 KBytes.
- Default bit — In a code-segment descriptor, the D flag is clear, indicating 16-bit addressing and operands are the default. In a stack-segment descriptor, the D flag is clear, indicating use of the SP register (instead of the ESP register) and a 64-KByte maximum segment limit.

For information on mixing 16- and 32-bit code in applications, see Chapter 23, "Mixing 16-Bit and 32-Bit Code."

24.33 SEGMENT AND ADDRESS WRAPAROUND

This section discusses differences in segment and address wraparound between the P6 family, Pentium, Intel486, Intel386, Intel 286, and 8086 processors.

24.33.1 Segment Wraparound

On the 8086 processor, an attempt to access a memory operand that crosses offset 65,535 or 0FFFFH or offset 0 (for example, moving a word to offset 65,535 or pushing a word when the stack pointer is set to 1) causes the offset to wrap around modulo 65,536 or 010000H. With the Intel 286 processor, any base and offset combination that addresses beyond 16 MBytes wraps around to the 1 MByte of the address space. The P6 family, Pentium, Intel486, and Intel386 processors in real-address mode generate an exception in these cases:

- A general-protection exception (#GP) if the segment is a data segment (that is, if the CS, DS, ES, FS, or GS register is being used to address the segment).
- A stack-fault exception (#SS) if the segment is a stack segment (that is, if the SS register is being used).

An exception to this behavior occurs when a stack access is data aligned, and the stack pointer is pointing to the last aligned piece of data at the top of the stack (ESP is FFFFFFFCH). When this data is popped, no segment limit violation occurs and the stack pointer will wrap around to 0.

The address space of the P6 family, Pentium, and Intel486 processors may wraparound at 1 MByte in real-address mode. An external A20M# pin forces wraparound if enabled. On Intel 8086 processors, it is possible to specify addresses greater than 1 MByte. For example, with a selector value FFFFH and an offset of FFFFH, the effective address would be 10FFE FH (1 MByte plus 65519 bytes). The 8086 processor, which can form addresses up to 20 bits long, truncates the uppermost bit, which “wraps” this address to FFE FH. However, the P6 family, Pentium, and Intel486 processors do not truncate this bit if A20M# is not enabled.

If a stack operation wraps around the address limit, shutdown occurs. (The 8086 processor does not have a shutdown mode or a limit.)

The behavior when executing near the limit of a 4-GByte selector (limit = FFFFFFFFH) is different between the Pentium Pro and the Pentium 4 family of processors. On the Pentium Pro, instructions which cross the limit -- for example, a two byte instruction such as INC EAX that is encoded as FFH C0H starting exactly at the limit faults for a segment violation (a one byte instruction at FFFFFFFFH does not cause an exception). Using the Pentium 4 micro-processor family, neither of these situations causes a fault.

Segment wraparound and the functionality of A20M# is used primarily by older operating systems and not used by modern operating systems. On newer Intel 64 processors, A20M# may be absent.

24.34 STORE BUFFERS AND MEMORY ORDERING

The Pentium 4, Intel Xeon, and P6 family processors provide a store buffer for temporary storage of writes (stores) to memory (see Section 13.10, “Store Buffer”). Writes stored in the store buffer(s) are always written to memory in program order, with the exception of “fast string” store operations (see Section 10.2.4, “Fast-String Operation and Out-of-Order Stores”).

The Pentium processor has two store buffers, one corresponding to each of the pipelines. Writes in these buffers are always written to memory in the order they were generated by the processor core.

It should be noted that only memory writes are buffered and I/O writes are not. The Pentium 4, Intel Xeon, P6 family, Pentium, and Intel486 processors do not synchronize the completion of memory writes on the bus and instruction execution after a write. An I/O, locked, or serializing instruction needs to be executed to synchronize writes with the next instruction (see Section 10.3, “Serializing Instructions”).

The Pentium 4, Intel Xeon, and P6 family processors use processor ordering to maintain consistency in the order that data is read (loaded) and written (stored) in a program and the order the processor actually carries out the reads and writes. With this type of ordering, reads can be carried out speculatively and in any order, reads can pass buffered writes, and writes to memory are always carried out in program order. (See Section 10.2, “Memory Ordering,” for more information about processor ordering.) The Pentium III processor introduced a new instruction to serialize writes and make them globally visible. Memory ordering issues can arise between a producer and a consumer of data. The SFENCE instruction provides a performance-efficient way of ensuring ordering between routines that produce weakly-ordered results and routines that consume this data.

No re-ordering of reads occurs on the Pentium processor, except under the condition noted in Section 10.2.1, “Memory Ordering in the Intel® Pentium® and Intel486™ Processors,” and in the following paragraph describing the Intel486 processor.

Specifically, the store buffers are flushed before the IN instruction is executed. No reads (as a result of cache miss) are reordered around previously generated writes sitting in the store buffers. The implication of this is that the store buffers will be flushed or emptied before a subsequent bus cycle is run on the external bus.

On both the Intel486 and Pentium processors, under certain conditions, a memory read will go onto the external bus before the pending memory writes in the buffer even though the writes occurred earlier in the program execution. A memory read will only be reordered in front of all writes pending in the buffers if all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions, the Intel486 and Pentium processors will not read from an external memory location that needs to be updated by one of the pending writes.

During a locked bus cycle, the Intel486 processor will always access external memory, it will never look for the location in the on-chip cache. All data pending in the Intel486 processor's store buffers will be written to memory before a locked cycle is allowed to proceed to the external bus. Thus, the locked bus cycle can be used for eliminating the possibility of reordering read cycles on the Intel486 processor. The Pentium processor does check its cache on a read-modify-write access and, if the cache line has been modified, writes the contents back to memory before locking the bus. The P6 family processors write to their cache on a read-modify-write operation (if the access does not split across a cache line) and does not write back to system memory. If the access does split across a cache line, it locks the bus and accesses system memory.

I/O reads are never reordered in front of buffered memory writes on an IA-32 processor. This ensures an update of all memory locations before reading the status from an I/O device.

24.35 BUS LOCKING

The Intel 286 processor performs the bus locking differently than the Intel P6 family, Pentium, Intel486, and Intel386 processors. Programs that use forms of memory locking specific to the Intel 286 processor may not run properly when run on later processors.

A locked instruction is guaranteed to lock only the area of memory defined by the destination operand, but may lock a larger memory area. For example, typical 8086 and Intel 286 configurations lock the entire physical memory space. Programmers should not depend on this.

On the Intel 286 processor, the LOCK prefix is sensitive to IOPL. If the CPL is greater than the IOPL, a general-protection exception (#GP) is generated. On the Intel386 DX, Intel486, and Pentium, and P6 family processors, no check against IOPL is performed.

The Pentium processor automatically asserts the LOCK# signal when acknowledging external interrupts. After signaling an interrupt request, an external interrupt controller may use the data bus to send the interrupt vector to the processor. After receiving the interrupt request signal, the processor asserts LOCK# to ensure that no other data appears on the data bus until the interrupt vector is received. This bus locking does not occur on the P6 family processors.

24.36 BUS HOLD

Unlike the 8086 and Intel 286 processors, but like the Intel386 and Intel486 processors, the P6 family and Pentium processors respond to requests for control of the bus from other potential bus masters, such as DMA controllers, between transfers of parts of an unaligned operand, such as two words which form a doubleword. Unlike the Intel386 processor, the P6 family, Pentium, and Intel486 processors respond to bus hold during reset initialization.

24.37 MODEL-SPECIFIC EXTENSIONS TO THE IA-32

Certain extensions to the IA-32 are specific to a processor or family of IA-32 processors and may not be implemented or implemented in the same way in future processors. The following sections describe these model-specific extensions. The CPUID instruction indicates the availability of some of the model-specific features.

24.37.1 Model-Specific Registers

The Pentium processor introduced a set of model-specific registers (MSRs) for use in controlling hardware functions and performance monitoring. To access these MSRs, two new instructions were added to the IA-32 architecture: read MSR (RDMSR) and write MSR (WRMSR). The MSRs in the Pentium processor are not guaranteed to be duplicated or provided in the next generation IA-32 processors.

The P6 family processors greatly increased the number of MSRs available to software. See Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for a complete list of the available MSRs. The new registers control the debug extensions, the performance counters, the machine-check exception capability, the machine-check architecture, and the MTRRs. These registers are accessible using the RDMSR and WRMSR instructions. Specific information on some of these new MSRs is provided in the following sections. As with the Pentium processor MSR, the P6 family processor MSRs are not guaranteed to be duplicated or provided in the next generation IA-32 processors.

24.37.2 RDMSR and WRMSR Instructions

The RDMSR (read model-specific register) and WRMSR (write model-specific register) instructions recognize a much larger number of model-specific registers in the P6 family processors. (See “RDMSR—Read from Model Specific Register” and “WRMSR—Write to Model Specific Register” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volumes 2A, 2B, 2C, & 2D, for more information.)

24.37.3 Memory Type Range Registers

Memory type range registers (MTRRs) are a new feature introduced into the IA-32 in the Pentium Pro processor. MTRRs allow the processor to optimize memory operations for different types of memory, such as RAM, ROM, frame buffer memory, and memory-mapped I/O.

MTRRs are MSRs that contain an internal map of how physical address ranges are mapped to various types of memory. The processor uses this internal memory map to determine the cacheability of various physical memory locations and the optimal method of accessing memory locations. For example, if a memory location is specified in an MTRR as write-through memory, the processor handles accesses to this location as follows. It reads data from that location in lines and caches the read data or maps all writes to that location to the bus and updates the cache to maintain cache coherency. In mapping the physical address space with MTRRs, the processor recognizes five types of memory: uncacheable (UC), uncacheable, speculatable, write-combining (WC), write-through (WT), write-protected (WP), and writeback (WB).

Earlier IA-32 processors (such as the Intel486 and Pentium processors) used the KEN# (cache enable) pin and external logic to maintain an external memory map and signal cacheable accesses to the processor. The MTRR mechanism simplifies hardware designs by eliminating the KEN# pin and the external logic required to drive it.

See Chapter 11, “Processor Management and Initialization,” and Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, for more information on the MTRRs.

24.37.4 Machine-Check Exception and Architecture

The Pentium processor introduced a new exception called the machine-check exception (#MC, interrupt 18). This exception is used to detect hardware-related errors, such as a parity error on a read cycle.

The P6 family processors extend the types of errors that can be detected and that generate a machine-check exception. It also provides a new machine-check architecture for recording information about a machine-check error and provides extended recovery capability.

The machine-check architecture provides several banks of reporting registers for recording machine-check errors. Each bank of registers is associated with a specific hardware unit in the processor. The primary focus of the machine checks is on bus and interconnect operations; however, checks are also made of translation lookaside buffer (TLB) and cache operations.

The machine-check architecture can correct some errors automatically and allow for reliable restart of instruction execution. It also collects sufficient information for software to use in correcting other machine errors not corrected by hardware.

See Chapter 17, “Machine-Check Architecture,” for more information on the machine-check exception and the machine-check architecture.

24.37.5 Performance-Monitoring Counters

The P6 family and Pentium processors provide two performance-monitoring counters for use in monitoring internal hardware operations. The number of performance monitoring counters and associated programming interfaces may be implementation specific for Pentium 4 processors, Pentium M processors. Later processors may have implemented these as part of an architectural performance monitoring feature. The architectural and non-architectural performance monitoring interfaces for different processor families are described in Chapter 21, “Performance Monitoring.” <https://perfmon-events.intel.com/> lists all the events that can be counted for architectural performance monitoring events and non-architectural events. The counters are set up, started, and stopped using two MSRs and the RDMSR and WRMSR instructions. For the P6 family processors, the current count for a particular counter can be read using the new RDPNC instruction.

The performance-monitoring counters are useful for debugging programs, optimizing code, diagnosing system failures, or refining hardware designs. See Chapter 21, “Performance Monitoring,” for more information on these counters.

24.38 TWO WAYS TO RUN INTEL 286 PROCESSOR TASKS

When porting 16-bit programs to run on 32-bit IA-32 processors, there are two approaches to consider:

- Porting an entire 16-bit software system to a 32-bit processor, complete with the old operating system, loader, and system builder. Here, all tasks will have 16-bit TSSs. The 32-bit processor is being used as if it were a faster version of the 16-bit processor.
- Porting selected 16-bit applications to run in a 32-bit processor environment with a 32-bit operating system, loader, and system builder. Here, the TSSs used to represent 286 tasks should be changed to 32-bit TSSs. It is possible to mix 16 and 32-bit TSSs, but the benefits are small and the problems are great. All tasks in a 32-bit software system should have 32-bit TSSs. It is not necessary to change the 16-bit object modules themselves; TSSs are usually constructed by the operating system, by the loader, or by the system builder. See Chapter 23, “Mixing 16-Bit and 32-Bit Code,” for more detailed information about mixing 16-bit and 32-bit code.

Because the 32-bit processors use the contents of the reserved word of 16-bit segment descriptors, 16-bit programs that place values in this word may not run correctly on the 32-bit processors.

24.39 INITIAL STATE OF PENTIUM, PENTIUM PRO AND PENTIUM 4 PROCESSORS

Table 24-10 shows the state of the flags and other registers following power-up for the Pentium, Pentium Pro and Pentium 4 processors. The state of control register CR0 is 60000010H (see Figure 11-1 “Contents of CR0 Register after Reset” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). This places the processor in real-address mode with paging disabled.

Table 24-10. Processor State Following Power-up/Reset/INIT for Pentium, Pentium Pro and Pentium 4 Processors

Register	Pentium 4 Processor	Pentium Pro Processor	Pentium Processor
EFLAGS ¹	00000002H	00000002H	00000002H
EIP	0000FFF0H	0000FFF0H	0000FFF0H
CR0	60000010H ²	60000010H ²	60000010H ²
CR2, CR3, CR4	00000000H	00000000H	00000000H

Table 24-10. Processor State Following Power-up/Reset/INIT for Pentium, Pentium Pro and Pentium 4 Processors

Register	Pentium 4 Processor	Pentium Pro Processor	Pentium Processor
CS	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = F000H Base = FFFF0000H Limit = FFFFH AR = Present, R/W, Accessed
SS, DS, ES, FS, GS	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W, Accessed
EDX	00000FxxH	000n06xxH ³	000005xxH
EAX	0 ⁴	0 ⁴	0 ⁴
EBX, ECX, ESI, EDI, EBP, ESP	00000000H	00000000H	00000000H
ST0 through ST7 ⁵	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged	Pwr up or Reset: +0.0 FINIT/FNINIT: Unchanged
x87 FPU Control Word ⁵	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH	Pwr up or Reset: 0040H FINIT/FNINIT: 037FH
x87 FPU Status Word ⁵	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H
x87 FPU Tag Word ⁵	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH	Pwr up or Reset: 5555H FINIT/FNINIT: FFFFH
x87 FPU Data Operand and CS Seg. Selectors ⁵	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H	Pwr up or Reset: 0000H FINIT/FNINIT: 0000H
x87 FPU Data Operand and Inst. Pointers ⁵	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H	Pwr up or Reset: 00000000H FINIT/FNINIT: 00000000H
MM0 through MM7 ⁵	Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged	Pentium II and Pentium III Processors Only— Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged	Pentium with MMX Technology Only— Pwr up or Reset: 0000000000000000H INIT or FINIT/FNINIT: Unchanged
XMM0 through XMM7	Pwr up or Reset: 0H INIT: Unchanged	If CPUID.01H:SSE is 1 — Pwr up or Reset: 0H INIT: Unchanged	NA
MXCSR	Pwr up or Reset: 1F80H INIT: Unchanged	Pentium III processor only— Pwr up or Reset: 1F80H INIT: Unchanged	NA
GDTR, IDTR	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W	Base = 00000000H Limit = FFFFH AR = Present, R/W
LDTR, Task Register	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W	Selector = 0000H Base = 00000000H Limit = FFFFH AR = Present, R/W
DR0, DR1, DR2, DR3	00000000H	00000000H	00000000H
DR6	FFFF0FF0H	FFFF0FF0H	FFFF0FF0H

Table 24-10. Processor State Following Power-up/Reset/INIT for Pentium, Pentium Pro and Pentium 4 Processors

Register	Pentium 4 Processor	Pentium Pro Processor	Pentium Processor
DR7	00000400H	00000400H	00000400H
Time-Stamp Counter	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged
Perf. Counters and Event Select	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged	Power up or Reset: 0H INIT: Unchanged
All Other MSRs	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged
Data and Code Cache, TLBs	Invalid ⁶	Invalid ⁶	Invalid ⁶
Fixed MTRRs	Pwr up or Reset: Disabled INIT: Unchanged	Pwr up or Reset: Disabled INIT: Unchanged	Not Implemented
Variable MTRRs	Pwr up or Reset: Disabled INIT: Unchanged	Pwr up or Reset: Disabled INIT: Unchanged	Not Implemented
Machine-Check Architecture	Pwr up or Reset: Undefined INIT: Unchanged	Pwr up or Reset: Undefined INIT: Unchanged	Not Implemented
APIC	Pwr up or Reset: Enabled INIT: Unchanged	Pwr up or Reset: Enabled INIT: Unchanged	Pwr up or Reset: Enabled INIT: Unchanged
R8-R15 ⁷	0000000000000000H	0000000000000000H	N.A.
XMM8-XMM15 ⁷	Pwr up or Reset: 0H INIT: Unchanged	Pwr up or Reset: 0H INIT: Unchanged	N.A.

NOTES:

1. The 10 most-significant bits of the EFLAGS register are undefined following a reset. Software should not depend on the states of any of these bits.
2. The CD and NW flags are unchanged, bit 4 is set to 1, all other bits are cleared.
3. Where “n” is the Extended Model Value for the respective processor.
4. If Built-In Self-Test (BIST) is invoked on power up or reset, EAX is 0 only if all tests passed. (BIST cannot be invoked during an INIT.)
5. The state of the x87 FPU and MMX registers is not changed by the execution of an INIT.
6. Internal caches are invalid after power-up and RESET, but left unchanged with an INIT.
7. If the processor supports IA-32e mode.

19. Updates to Chapter 26, Volume 3C

Change bars and violet text show changes to Chapter 26 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- In Section 26.1, "Overview", updated footnote 2 on page 26-1.
- In Section 26.6.2, "Processor-Based VM-Execution Controls", added Bit Position 5 for SEAM guest-physical address width to Table 26-8.
- In Section 26.6.11, "Extended-Page-Table Pointer (EPTP)", updated extended-page-table pointer description and Table 26-9 for MAXPHYADDR.
- In Section 26.6.21, "Sub-Page-Permission-Table Pointer (SPPTP)", updated Table 26-11 for MAXPHYADDR.
- In Section 26.6.22, "Fields Related to Hypervisor-Managed Linear-Address Translation" updated Table 26-12 for MAXPHYADDR.
- Added Section 26.6.26, "Fields Controlling SEAM Non-Root Operation".
- In Section 26.8.1, "VM-Entry Controls", added bit 25 for SEAM-guest telemetry to Table 26-17.
- Updated Section 26.9.2, "Information for VM Exits Due to Vectored Events" to include UDB.
- Updated footnote 1 for the VMXON pointer.

26.1 OVERVIEW

A logical processor uses **virtual-machine control data structures (VMCSs)** while it is in VMX operation. These manage transitions into and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. This structure is manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

A VMM can use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM can use a different VMCS for each virtual processor.

A logical processor associates a region in memory with each VMCS. This region is called the **VMCS region**.¹ Software references a specific VMCS using the 64-bit physical address of the region (a **VMCS pointer**). VMCS pointers must be aligned on a 4-KByte boundary (bits 11:0 must be zero). These pointers must not set bits beyond the processor's physical-address width.^{2,3}

A logical processor may maintain a number of VMCSs that are **active**. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. At any given time, at most one of the active VMCSs is the **current** VMCS. (This document frequently uses the term "the VMCS" to refer to the current VMCS.) The VMLAUNCH, VMREAD, VMRESUME, and VMWRITE instructions operate only on the current VMCS.

The following items describe how a logical processor determines which VMCSs are active and which is current:

- The memory operand of the VMPTRLD instruction is the address of a VMCS. After execution of the instruction, that VMCS is both active and current on the logical processor. Any other VMCS that had been active remains so, but no other VMCS is current.
- The VMCS link pointer field in the current VMCS (see Section 26.4.2) is itself the address of a VMCS. If VM entry is performed successfully with the 1-setting of the "VMCS shadowing" VM-execution control, the VMCS referenced by the VMCS link pointer field becomes active on the logical processor. The identity of the current VMCS does not change.
- The memory operand of the VMCLEAR instruction is also the address of a VMCS. After execution of the instruction, that VMCS is neither active nor current on the logical processor. If the VMCS had been current on the logical processor, the logical processor no longer has a current VMCS.

The VMPTRST instruction stores the address of the logical processor's current VMCS into a specified memory location (it stores the value FFFFFFFF_FFFFFFFFH if there is no current VMCS).

The **launch state** of a VMCS determines which VM-entry instruction should be used with that VMCS: the VMLAUNCH instruction requires a VMCS whose launch state is "clear"; the VMRESUME instruction requires a VMCS whose launch state is "launched". A logical processor maintains a VMCS's launch state in the corresponding VMCS region. The following items describe how a logical processor manages the launch state of a VMCS:

- If the launch state of the current VMCS is "clear", successful execution of the VMLAUNCH instruction changes the launch state to "launched".

1. The amount of memory required for a VMCS region is at most 4 KBytes. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).
2. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. This width is used to determine the value of MAXPHYADDR, which constrains VMCS pointers. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM). IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.
3. If IA32_VMX_BASIC[48] is read as 1, these pointers must not set any bits in the range 63:32; see Appendix A.1.

- The memory operand of the VMCLEAR instruction is the address of a VMCS. After execution of the instruction, the launch state of that VMCS is “clear”.
- There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to discover it (it cannot be read using VMREAD).

Figure 26-1 illustrates the different states of a VMCS. It uses “X” to refer to the VMCS and “Y” to refer to any other VMCS. Thus: “VMPTRLD X” always makes X current and active; “VMPTRLD Y” always makes X not current (because it makes Y current); VMLAUNCH makes the launch state of X “launched” if X was current and its launch state was “clear”; and VMCLEAR X always makes X inactive and not current and makes its launch state “clear”.

The figure does not illustrate operations that do not modify the VMCS state relative to these parameters (e.g., execution of VMPTRLD X when X is already current). Note that VMCLEAR X makes X “inactive, not current, and clear,” even if X’s current state is not defined (e.g., even if X has not yet been initialized). See Section 26.11.3.

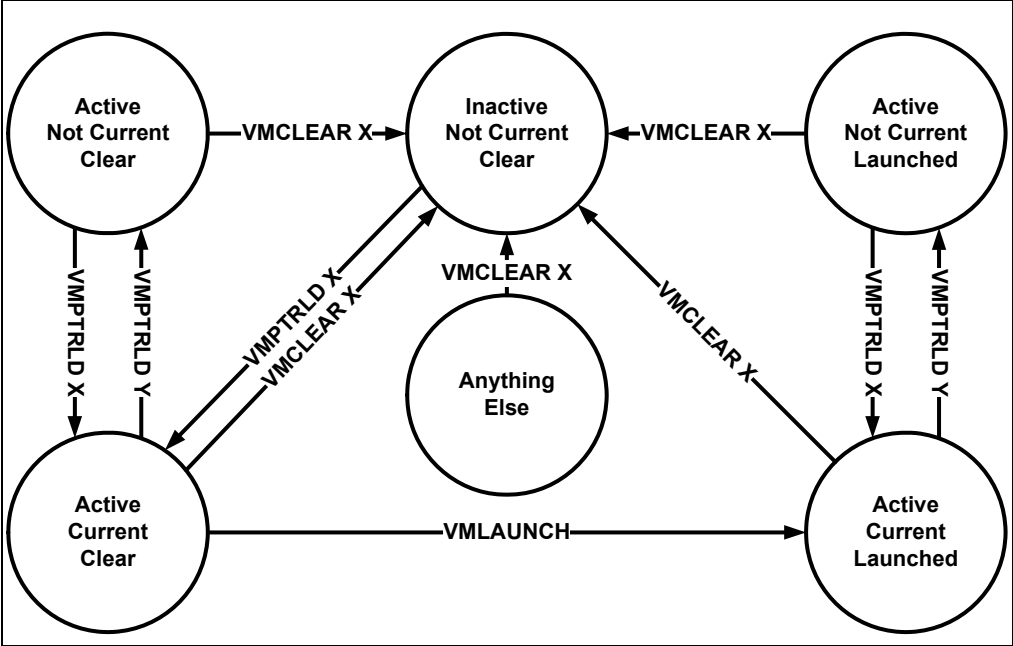


Figure 26-1. States of VMCS X

Because a shadow VMCS (see Section 26.10) cannot be used for VM entry, the launch state of a shadow VMCS is not meaningful. Figure 26-1 does not illustrate all the ways in which a shadow VMCS may be made active.

26.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.¹ The format of a VMCS region is given in Table 26-1.

Table 26-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 26.10)
4	VMX-abort indicator

1. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

Table 26-1. Format of the VMCS Region (Contd.)

Byte Offset	Contents
8	VMCS data (implementation-specific format)

The first 4 bytes of the VMCS region contain the **VMCS revision identifier** at bits 30:0.¹ Processors that maintain VMCS data in different formats (see below) use different VMCS revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.² Bit 31 of this 4-byte region indicates whether the VMCS is a shadow VMCS (see Section 26.10).

Software should write the VMCS revision identifier to the VMCS region before using that region for a VMCS. The VMCS revision identifier is never written by the processor; VMPTRLD fails if its operand references a VMCS region whose VMCS revision identifier differs from that used by the processor. (VMPTRLD also fails if the shadow-VMCS indicator is 1 and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control; see Section 26.6.2.) Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Software should clear or set the shadow-VMCS indicator depending on whether the VMCS is to be an ordinary VMCS or a shadow VMCS (see Section 26.10). VMPTRLD fails if the shadow-VMCS indicator is set and the processor does not support the 1-setting of the “VMCS shadowing” VM-execution control. Software can discover support for this setting by reading the VMX capability MSR IA32_VMX_PROCBASED_CTL2 (see Appendix A.3.3).

The next 4 bytes of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 29.7). Software may also write into this field.

The remainder of the VMCS region is used for **VMCS data** (those parts of the VMCS that control VMX non-root operation and the VMX transitions). The format of these data is implementation-specific. VMCS data are discussed in Section 26.3 through Section 26.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 26.11.4) in writeback cacheable memory. Future implementations may allow or require a different memory type³. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

26.3 ORGANIZATION OF VMCS DATA

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. On some processors, these fields are read-only.⁴

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

-
1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.
 2. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions.
 3. Alternatively, software may map any of these regions or structures with the UC memory type. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.
 4. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

26.4 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. VM entries load processor state from these fields and VM exits store processor state into these fields. See Section 28.3.2 and Section 29.3 for details.

26.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).
- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).¹
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
 - Selector (16 bits).
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
 - Segment limit (32 bits). The limit field is always a measure in bytes.
 - Access rights (32 bits). The format of this field is given in Table 26-2 and detailed as follows:
 - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
 - Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.²
 - Bits 31:17 are reserved.

Table 26-2. Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see “Interrupt 10—Invalid TSS Exception (#TS)” in Section 7.14, “Exception and Interrupt Handling in 64-bit Mode,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 12-1 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 26-2. Format of Access Rights (Contd.)

Bit Position(s)	Field
13	Reserved (except for CS) L — 64-bit mode active (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the “hidden” part (or “descriptor cache”) of each segment register. These data are included in the VMCS because it is possible for a segment register’s descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register’s selector.

The value of the DPL field for SS is always equal to the logical processor’s current privilege level (CPL).¹

On some processors, executions of VMWRITE ignore attempts to write non-zero values to any of bits 11:8 or bits 31:17. On such processors, VMREAD always returns 0 for those bits, and VM entry treats those bits as if they were all 0 (see Section 28.3.1.2).

- The following fields for each of the registers GDTR and IDTR:
 - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The following MSRs:
 - IA32_DEBUGCTL (64 bits)
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture)
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-entry control.
 - IA32_PAT (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_PAT” VM-entry control or that of the “save IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_EFER” VM-entry control or that of the “save IA32_EFER” VM-exit control.
 - IA32_BNDCFGS (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control.
 - IA32_RTIT_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control.
 - IA32_LBR_CTL (64 bits). This field is supported only on processors that support either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
 - IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

1. In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.

- IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-entry control.
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.
- The register SMBASE (32 bits). This register contains the base address of the logical processor’s SMRAM image.

26.4.2 Guest Non-Register State

In addition to the register state described in Section 26.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor’s activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

The following activity states are defined:¹

- 0: **Active**. The logical processor is executing instructions normally.
- 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
- 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**² or some other serious error.
- 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 26-3.

Table 26-3. Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the “STI—Set Interrupt Flag” section in Chapter 4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B. Execution of STI with RFLAGS.IF = 0 blocks maskable interrupts on the instruction boundary following its execution. ¹ Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See Section 7.8.3, “Masking Exceptions and Interrupts When Switching Stacks,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. Execution of a MOV to SS or a POP to SS blocks or suppresses certain debug exceptions as well as interrupts (maskable and nonmaskable) on the instruction boundary following its execution. Setting this bit indicates that this blocking is in effect. ² This document uses the term “blocking by MOV SS,” but it applies equally to POP SS.
2	Blocking by SMI	See Section 33.2, “System Management Interrupt (SMI).” System-management interrupts (SMIs) are disabled while the processor is in system-management mode (SMM). Setting this bit indicates that blocking of SMIs is in effect.

1. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 29.1.

2. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 26-3. Format of Interruptibility State (Contd.)

Bit Position(s)	Bit Name	Notes
3	Blocking by NMI	See Section 7.7.1, “Handling Multiple NMIs,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A and Section 33.8, “NMI Handling While in SMM.” Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 27.3 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the “virtual NMIs” VM-execution control (see Section 26.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to “virtual-NMI blocking” (the fact that guest software is not ready for an NMI).
4	Enclave interruption	Set to 1 if the VM exit occurred while the logical processor was in enclave mode. Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode. A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field sets this bit; or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
31:5	Reserved	VM entry will fail if these bits are not 0. See Section 28.3.1.5.

NOTES:

1. Nonmaskable interrupts and system-management interrupts may also be inhibited on the instruction boundary following such an execution of STI.
 2. System-management interrupts may also be inhibited on the instruction boundary following such an execution of MOV or POP.
- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.¹ This field contains information about such exceptions. This field is described in Table 26-4.

Table 26-4. Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 – B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
10:4	Reserved	VM entry fails if these bits are not 0. See Section 28.3.1.5.
11	BLD	When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 19.3.1.6, “OS Bus-Lock Detection”). ¹
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7; the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled; or a bus lock was asserted while CPL > 0 and OS bus-lock detection had been enabled.

1. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 7.8.3 of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 7-2 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Table 26-4. Format of Pending-Debug-Exceptions (Contd.)

Bit Position(s)	Bit Name	Notes
13	Reserved	VM entry fails if this bit is not 0. See Section 28.3.1.5.
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
15	Reserved	VM entry fails if this bit is not 0. See Section 28.3.1.5.
16	RTM	When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Reserved	VM entry fails if these bits are not 0. See Section 28.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- **VMCS link pointer** (64 bits). If the “VMCS shadowing” VM-execution control is 1, the VMREAD and VMWRITE instructions access the VMCS referenced by this pointer (see Section 26.10). Otherwise, software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 28.3.1.5).
- **VMX-preemption timer value** (32 bits). This field is supported only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 27.5.1 and Section 28.7.4.
- **Page-directory-pointer-table entries** (PDPTEs; 64 bits each). These four (4) fields (PDPTE0, PDPTE1, PDPTE2, and PDPTE3) are supported only on processors that support the 1-setting of the “enable EPT” VM-execution control. They correspond to the PDPTEs referenced by CR3 when PAE paging is in use (see Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). They are used only if the “enable EPT” VM-execution control is 1.
- **Guest interrupt status** (16 bits). This field is supported only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control. It characterizes part of the guest’s virtual-APIC state and does not correspond to any processor or APIC registers. It comprises two 8-bit subfields:
 - **Requesting virtual interrupt (RVI)**. This is the low byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is requesting service. (The value 0 implies that there is no such interrupt.)
 - **Servicing virtual interrupt (SVI)**. This is the high byte of the guest interrupt status. The processor treats this value as the vector of the highest priority virtual interrupt that is in service. (The value 0 implies that there is no such interrupt.)

See Chapter 31 for more information on the use of this field.

- **PML index** (16 bits). This field is supported only on processors that support the 1-setting of the “enable PML” VM-execution control. It contains the logical index of the next entry in the page-modification log. Because the page-modification log comprises 512 entries, the PML index is typically a value in the range 0–511. Details of the page-modification log and use of the PML index are given in Section 30.3.6.

26.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 29.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- The following MSRs:
 - IA32_SYSENTER_CS (32 bits)
 - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
 - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PERF_GLOBAL_CTRL” VM-exit control.
 - IA32_PAT (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_PAT” VM-exit control.
 - IA32_EFER (64 bits). This field is supported only on processors that support the 1-setting of the “load IA32_EFER” VM-exit control.
 - IA32_S_CET (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_INTERRUPT_SSP_TABLE_ADDR (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.
 - IA32_PKRS (64 bits). This field is supported only on processors that support the 1-setting of the “load PKRS” VM-exit control.
- The shadow-stack pointer register SSP (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 29.5 for details of how state is loaded on VM exits.

26.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 26.6.1 through Section 26.6.8.

26.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).¹ Table 26-5 lists the controls. See Chapter 28 for how these controls affect processor behavior in VMX non-root operation.

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 27.2).

Table 26-5. Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally through the guest interrupt-descriptor table (IDT). If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using descriptor 2 of the IDT. This control also determines interactions between IRET and blocking by NMI (see Section 27.3).
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 26-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 26.6.2).
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 27.5.1. A VM exit occurs when the timer counts down to zero; see Section 27.2.
7	Process posted interrupts	If this control is 1, the processor treats interrupts with the posted-interrupt notification vector (see Section 26.6.8) specially, updating the virtual-APIC page with posted-interrupt requests (see Section 31.6).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs `IA32_VMX_PINBASED_CTL`s and `IA32_VMX_TRUE_PINBASED_CTL`s (see Appendix A.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 28.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR `IA32_VMX_PINBASED_CTL`s will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR `IA32_VMX_TRUE_PINBASED_CTL`s MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

26.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute three vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.¹ These are the **primary processor-based VM-execution controls** (32 bits), the **secondary processor-based VM-execution controls** (32 bits), and the tertiary **VM-execution controls** (64 bits).

Table 26-6 lists the primary processor-based VM-execution controls. See Chapter 26 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 26-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if <code>RFLAGS.IF</code> = 1 and there are no other blocking of interrupts (see Section 26.4.2).
3	Use TSC offsetting	This control determines whether executions of <code>RDTSC</code> , executions of <code>RDTSCP</code> , and executions of <code>RDMSR</code> that read from the <code>IA32_TIME_STAMP_COUNTER</code> MSR return a value modified by the TSC offset field (see Section 26.6.5 and Section 27.3).
7	HLT exiting	This control determines whether executions of <code>HLT</code> cause VM exits.
9	INVLPG exiting	This determines whether executions of <code>INVLPG</code> and <code>INVPID</code> cause VM exits.
10	MWAIT exiting	This control determines whether executions of <code>MWAIT</code> cause VM exits.
11	RDPNC exiting	This control determines whether executions of <code>RDPNC</code> cause VM exits.

1. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 27.1.2), as do task switches (see Section 27.2).

Table 26-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 26.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 27.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
17	Activate tertiary controls	This control determines whether the tertiary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the tertiary processor-based VM-execution controls were also 0.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits.
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits.
21	Use TPR shadow	Setting this control to 1 enables TPR virtualization and other APIC-virtualization features. See Chapter 31.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 26.4.2).
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 26.6.4 and Section 27.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 27.5.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 26.6.9 and Section 27.1.3). For this control, “0” means “do not use MSR bitmaps” and “1” means “use MSR bitmaps.” If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs `IA32_VMX_PROCBASED_CTLS` and `IA32_VMX_TRUE_PROCBASED_CTLS` (see Appendix A.3.2) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 28.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR `IA32_VMX_PROCBASED_CTLS` will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR `IA32_VMX_TRUE_PROCBASED_CTLS` MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of

the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 26-7 lists the secondary processor-based VM-execution controls. See Chapter 26 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 26-7. Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, the logical processor treats specially accesses to the page with the APIC-access address. See Section 31.4.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 30.3.
2	Descriptor-table exiting	This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits.
3	Enable RDTSCP	If this control is 0, any execution of RDTSCP causes an invalid-opcode exception (#UD).
4	Virtualize x2APIC mode	If this control is 1, the logical processor treats specially RDMSR and WRMSR to APIC MSRs (in the range 800H-8FFH). See Section 31.5.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 30.1.
6	WBINVD exiting	This control determines whether executions of WBINVD and WBNOINVD cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpaged protected mode or in real-address mode.
8	APIC-register virtualization	If this control is 1, the logical processor virtualizes certain APIC accesses. See Section 31.4 and Section 31.5.
9	Virtual-interrupt delivery	This controls enables the evaluation and delivery of pending virtual interrupts as well as the emulation of writes to the APIC registers that control interrupt prioritization.
10	PAUSE-loop exiting	This control determines whether a series of executions of PAUSE can cause a VM exit (see Section 26.6.13 and Section 27.1.3).
11	RDRAND exiting	This control determines whether executions of RDRAND cause VM exits.
12	Enable INVPCID	If this control is 0, any execution of INVPCID causes a #UD.
13	Enable VM functions	Setting this control to 1 enables use of the VMFUNC instruction in VMX non-root operation. See Section 27.5.6.
14	VMCS shadowing	If this control is 1, executions of VMREAD and VMWRITE in VMX non-root operation may access a shadow VMCS (instead of causing VM exits). See Section 26.10 and Section 32.3.
15	Enable ENCLS exiting	If this control is 1, executions of ENCLS consult the ENCLS-exiting bitmap to determine whether the instruction causes a VM exit. See Section 26.6.16 and Section 27.1.3.
16	RDSEED exiting	This control determines whether executions of RDSEED cause VM exits.
17	Enable PML	If this control is 1, an access to a guest-physical address that sets an EPT dirty bit first adds an entry to the page-modification log. See Section 30.3.6.
18	EPT-violation #VE	If this control is 1, EPT violations may cause virtualization exceptions (#VE) instead of VM exits. See Section 27.5.7.
19	Conceal VMX from PT	If this control is 1, Intel Processor Trace suppresses from PIPs an indication that the processor was in VMX non-root operation and omits a VMCS packet from any PSB+ produced in VMX non-root operation (see Chapter 35).
20	Enable XSAVES/XRSTORS	If this control is 0, any execution of XSAVES or XRSTORS causes a #UD.
21	PASID translation	If this control is 1, PASID translation is performed for executions of ENQCMD and ENQCMLS. See Section 27.5.8.
22	Mode-based execute control for EPT	If this control is 1, EPT execute permissions are based on whether the linear address being accessed is supervisor mode or user mode. See Chapter 30.

Table 26-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
23	Sub-page write permissions for EPT	If this control is 1, EPT write permissions may be specified at the granularity of 128 bytes. See Section 30.3.4.
24	Intel PT uses guest physical addresses	If this control is 1, all output addresses used by Intel Processor Trace are treated as guest-physical addresses and translated using EPT. See Section 27.5.4.
25	Use TSC scaling	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC multiplier field (see Section 26.6.5 and Section 27.3).
26	Enable user wait and pause	If this control is 0, any execution of TPAUSE, UMONITOR, or UMWAIT causes a #UD.
27	Enable PCONFIG	If this control is 0, any execution of PCONFIG causes a #UD.
30	VMM bus-lock detection	This control determines whether assertion of a bus lock causes a VM exit. See Section 27.2.
31	Instruction timeout	If this control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within a specified amount of time. See Section 26.6.24 and Section 27.2.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROC-BASED_CTL2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 28.2.1.1).

Bit 17 of the primary processor-based VM-execution controls determines whether the tertiary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the tertiary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 17 of the primary processor-based VM-execution controls do not support the tertiary processor-based VM-execution controls.

Table 26-8 lists the tertiary processor-based VM-execution controls. See Chapter 26 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 26-8. Definitions of Tertiary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	LOADIWKEY exiting	This control determines whether executions of LOADIWKEY cause VM exits.
1	Enable HLAT	This control enables hypervisor-managed linear-address translation. See Section 5.5.1.
2	EPT paging-write control	If this control is 1, EPT permissions can be specified to allow writes only for paging-related updates. See Section 30.3.3.2.
3	Guest-paging verification	If this control is 1, EPT permissions can be specified to prevent accesses using linear addresses whose translation has certain properties. See Section 30.3.3.2.
4	IPI virtualization	If this control is 1, virtualization of interprocessor interrupts (IPIs) is enabled. See Section 31.1.6.
5	SEAM guest-physical address width	This control determines the operation of EPT in SEAM non-root operation. See Section 34.3.2.
6	Enable MSR-list instructions	If this control is 0, any execution of RDMSRLIST or WRMSRLIST causes a #UD.
7	Virtualize IA32_SPEC_CTRL	If this control is 1, the operation of the RDMSR and WRMSR instructions is changed when accessing the IA32_SPEC_CTRL MSR. See Section 26.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROC-BASED_CTL3 (see Appendix A.3.4) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 28.2.1.1).

26.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally through the IDT, using the descriptor corresponding to the exception's vector.

Whether a page fault (exception with vector 14) causes a VM exit is determined by bit 14 in the exception bitmap as well as the error code produced by the page fault and two 32-bit fields in the VMCS (the **page-fault error-code mask** and **page-fault error-code match**). See Section 27.2 for details.

26.6.4 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of **I/O bitmaps** A and B (each of which are 4 KBytes in size). I/O bitmap A contains one bit for each I/O port in the range 0000H through 7FFFH; I/O bitmap B contains bits for ports in the range 8000H through FFFFH.

A logical processor uses these bitmaps if and only if the "use I/O bitmaps" control is 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is 1. See Section 27.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

26.6.5 Time-Stamp Counter Offset and Multiplier

The VM-execution control fields include a 64-bit **TSC-offset** field. If the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1, this field controls executions of the RDTSC and RDTSCP instructions. It also controls executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. For all of these, the value of the TSC offset is added to the value of the time-stamp counter, and the sum is returned to guest software in EDX:EAX.

Processors that support the 1-setting of the "use TSC scaling" control also support a 64-bit **TSC-multiplier** field. If this control is 1 (and the "RDTSC exiting" control is 0 and the "use TSC offsetting" control is 1), this field also affects the executions of the RDTSC, RDTSCP, and RDMSR instructions identified above. Specifically, the contents of the time-stamp counter is first multiplied by the TSC multiplier before adding the TSC offset.

See Chapter 26 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

26.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include **guest/host masks** and **read shadows** for the CR0 and CR4 registers. These fields control executions of instructions that access those registers (including CLTS, LMSW, MOV CR, and SMSW). They are 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

In general, bits set to 1 in a guest/host mask correspond to bits "owned" by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits.
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits "owned" by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Chapter 28 for details regarding how these fields affect VMX non-root operation.

26.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 **CR3-target values** and a **CR3-target count**. The CR3-target values each have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not. The CR3-target count has 32 bits on all processors.

An execution of MOV to CR3 in VMX non-root operation does not cause a VM exit if its source operand matches one of these values. If the CR3-target count is n , only the first n CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit.

There are no limitations on the values that can be written for the CR3-target values. VM entry fails (see Section 28.2) if the CR3-target count is greater than 4.

Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine the number of values supported.

26.6.8 Controls for APIC Virtualization

There are three mechanisms by which software accesses registers of the logical processor's local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 12.4.4, "Local APIC Status and Location," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, and the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).¹
- If the local APIC is in x2APIC mode, it can access the local APIC's registers using the RDMSR and WRMSR instructions (see the Intel® 64 Architecture Processor Topology Enumeration Technical Paper).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

Several processor-based VM-execution controls (see Section 26.6.2) control such accesses. These are "use TPR shadow", "virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", "APIC-register virtualization", and "IPI virtualization". These controls interact with the following fields:

- **APIC-access address** (64 bits). This field contains the physical address of the 4-KByte **APIC-access page**. If the "virtualize APIC accesses" VM-execution control is 1, access to this page may cause VM exits or be virtualized by the processor. See Section 31.4.

The APIC-access address exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.

- **Virtual-APIC address** (64 bits). This field contains the physical address of the 4-KByte **virtual-APIC page**. The processor uses the virtual-APIC page to virtualize certain accesses to APIC registers and to manage virtual interrupts; see Chapter 31.

Depending on the setting of the controls indicated earlier, the virtual-APIC page may be accessed by the following operations:

- The MOV CR8 instructions (see Section 31.3).
- Accesses to the APIC-access page if, in addition, the "virtualize APIC accesses" VM-execution control is 1 (see Section 31.4).
- The RDMSR and WRMSR instructions if, in addition, the value of ECX is in the range 800H–8FFH (indicating an APIC MSR) and the "virtualize x2APIC mode" VM-execution control is 1 (see Section 31.5).

If the "use TPR shadow" VM-execution control is 1, VM entry ensures that the virtual-APIC address is 4-KByte aligned. The virtual-APIC address exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which bits 7:4 of VTPR (see Section 31.1.1) cannot fall. If the "virtual-interrupt delivery" VM-execution control is 0, a VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the value of those bits below the TPR threshold. See Section 31.1.2.

The TPR threshold exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

- **EOI-exit bitmap** (4 fields; 64 bits each). These fields are supported only on processors that support the 1-setting of the "virtual-interrupt delivery" VM-execution control. They are used to determine which virtualized writes to the APIC's EOI register cause VM exits:

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

- EOI_EXIT0 contains bits for vectors from 0 (bit 0) to 63 (bit 63).
- EOI_EXIT1 contains bits for vectors from 64 (bit 0) to 127 (bit 63).
- EOI_EXIT2 contains bits for vectors from 128 (bit 0) to 191 (bit 63).
- EOI_EXIT3 contains bits for vectors from 192 (bit 0) to 255 (bit 63).

See Section 31.1.4 for more information on the use of this field.

- **Posted-interrupt notification vector** (16 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. Its low 8 bits contain the interrupt vector that is used to notify a logical processor that virtual interrupts have been posted. See Section 31.6 for more information on the use of this field.
- **Posted-interrupt descriptor address** (64 bits). This field is supported only on processors that support the 1-setting of the “process posted interrupts” VM-execution control. It is the physical address of a 64-byte aligned posted interrupt descriptor. See Section 31.6 for more information on the use of this field.
- **PID-pointer table address** (64 bits). This field contains the physical address of the **PID-pointer table**. If the “IPI virtualization” VM-execution control is 1, the logical processor uses entries in this table to virtualize IPIs. See Section 31.1.6.
- **Last PID-pointer index** (16 bits). This field contains the index of the last entry in the PID-pointer table.

26.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the “use MSR bitmaps” VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048). This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.
- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072). This contains one bit for each MSR address in the range C0000000H to C0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the “use MSR bitmaps” control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 27.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

26.6.10 Executive-VMCS Pointer

The executive-VMCS pointer is a 64-bit field used in the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). SMM VM exits save this field as described in Section 33.15.2. VM entries that return from SMM use this field as described in Section 33.15.4.

26.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of **EPT paging structure that is the root of the hierarchy of structures that control the translation of guest-physical addresses** (see Section 30.3.2), as well as other EPT configuration information. The format of this field is shown in Table 26-9.

Table 26-9. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 30.3.7): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved. ¹
5:3	This value is 1 less than the EPT page-walk length (see Section 30.3.2)
6	Setting this control to 1 enables accessed and dirty flags for EPT (see Section 30.3.5) ²
7	Setting this control to 1 enables enforcement of access rights for supervisor shadow-stack pages (see Section 30.3.3.2) ³
11:8	Reserved
M-1:12	Bits M-1:12 of the physical address of the 4-KByte aligned EPT paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT) ⁴
63:M	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT paging-structure memory types are supported.
2. Not all processors support accessed and dirty flags for EPT. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
3. Not all processors enforce access rights for shadow-stack pages. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.
4. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 26-1.

The EPTP exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.

26.6.12 Virtual-Processor Identifier (VPID)

The **virtual-processor identifier** (VPID) is a 16-bit field. It exists only on processors that support the 1-setting of the “enable VPID” VM-execution control. See Section 30.1 for details regarding the use of this field.

26.6.13 Controls for PAUSE-Loop Exiting

On processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control, the VM-execution control fields include the following 32-bit fields:

- **PLE_Gap.** Software can configure this field as an upper bound on the amount of time between two successive executions of PAUSE in a loop.
- **PLE_Window.** Software can configure this field as an upper bound on the amount of time a guest is allowed to execute in a PAUSE loop.

These fields measure time based on a counter that runs at the same rate as the timestamp counter (TSC). See Section 27.1.3 for more details regarding PAUSE-loop exiting.

26.6.14 VM-Function Controls

The **VM-function controls** constitute a 64-bit vector that governs use of the VMFUNC instruction in VMX non-root operation. This field is supported only on processors that support the 1-settings of both the “activate secondary controls” primary processor-based VM-execution control and the “enable VM functions” secondary processor-based VM-execution control.

Table 26-10 lists the VM-function controls. See Section 27.5.6 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 26-10. Definitions of VM-Function Controls

Bit Position(s)	Name	Description
0	EPTP switching	The EPTP-switching VM function changes the EPT pointer to a value chosen from the EPTP list. See Section 27.5.6.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_VMFUNC (see Appendix A.11) to determine which bits are reserved. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 28.2.1.1).

Processors that support the 1-setting of the “EPTP switching” VM-function control also support a 64-bit field called the **EPTP-list address**. This field contains the physical address of the 4-KByte **EPTP list**. The EPTP list comprises 512 8-Byte entries (each an EPTP value) and is used by the EPTP-switching VM function (see Section 27.5.6.3).

26.6.15 VMCS Shadowing Bitmap Addresses

On processors that support the 1-setting of the “VMCS shadowing” VM-execution control, the VM-execution control fields include the 64-bit physical addresses of the **VMREAD bitmap** and the **VMWRITE bitmap**. Each bitmap is 4 KBytes in size and thus contains 32 KBits. The addresses are the **VMREAD-bitmap address** and the **VMWRITE-bitmap address**.

If the “VMCS shadowing” VM-execution control is 1, executions of VMREAD and VMWRITE may consult these bitmaps (see Section 26.10 and Section 32.3).

26.6.16 ENCLS-Exiting Bitmap

The **ENCLS-exiting bitmap** is a 64-bit field. If the “enable ENCLS exiting” VM-execution control is 1, execution of ENCLS causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the bit is 0, the instruction executes normally. See Section 27.1.3 for more information.

26.6.17 PCONFIG-Exiting Bitmap

The **PCONFIG-exiting bitmap** is a 64-bit field. If the “enable PCONFIG” VM-execution control is 1, execution of PCONFIG causes a VM exit if the bit in this field corresponding to the value of EAX is 1. If the control is 0, any execution of PCONFIG causes a #UD. See Section 27.1.3 for more information.

26.6.18 Control Field for Page-Modification Logging

The **PML address** is a 64-bit field. It is the 4-KByte aligned address of the **page-modification log**. The page-modification log consists of 512 64-bit entries. It is used for the page-modification logging feature. Details of the page-modification logging are given in Section 30.3.6.

If the “enable PML” VM-execution control is 1, VM entry ensures that the PML address is 4-KByte aligned. The PML address exists only on processors that support the 1-setting of the “enable PML” VM-execution control.

26.6.19 Controls for Virtualization Exceptions

- On processors that support the 1-setting of the “EPT-violation #VE” VM-execution control, the VM-execution control fields include the following:
- **Virtualization-exception information address** (64 bits). This field contains the physical address of the **virtualization-exception information area**. When a logical processor encounters a virtualization exception, it saves virtualization-exception information at the virtualization-exception information address; see Section 27.5.7.2.
 - **EPTP index** (16 bits). When an EPT violation causes a virtualization exception, the processor writes the value of this field to the virtualization-exception information area. The EPTP-switching VM function updates this field (see Section 27.5.6.3).

26.6.20 XSS-Exiting Bitmap

On processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control, the VM-execution control fields include a 64-bit **XSS-exiting bitmap**. If the “enable XSAVES/XRSTORS” VM-execution control is 1, executions of XSAVES and XRSTORS may consult this bitmap (see Section 27.1.3 and Section 27.3).

26.6.21 Sub-Page-Permission-Table Pointer (SPPTP)

If the sub-page write-permission feature of EPT is enabled, EPT write permissions may be determined at a 128-byte granularity (see Section 30.3.4). These permissions are determined using a hierarchy of sub-page-permission structures in memory.

The root of this hierarchy is referenced by a VM-execution control field called the **sub-page-permission-table pointer** (SPPTP). The SPPTP contains the address of the base of the root SPP table (see Section 30.3.4.2). The format of this field is shown in Table 26-9.

Table 26-11. Format of Sub-Page-Permission-Table Pointer

Bit Position(s)	Field
11:0	Reserved
M-1:12	Bits N-1:12 of the physical address of the 4-KByte aligned root SPP table ¹
63:M	Reserved

NOTES:

1. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 26-1.

The SPPTP exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.

26.6.22 Fields Related to Hypervisor-Managed Linear-Address Translation

Two fields are used when the “enable HLAT” VM-execution control is 1, enabling HLAT paging:

- The **hypervisor-managed linear-address translation pointer** (HLAT pointer or HLATP) is used by HLAT paging to locate and access the first paging structure used for linear-address translation (see Section 5.5). The format of this field is shown in Table 26-12.

Table 26-12. Format of Hypervisor-Managed Linear-Address Translation Pointer

Bit Position(s)	Field
2:0	Reserved
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the first HLAT paging structure during linear-address translation.
11:5	Reserved
M-1:12	Guest-physical address (4KB-aligned) of the first HLAT paging structure during linear-address translation. ¹
63:M	Reserved

NOTES:

1. M is an abbreviation for MAXPHYADDR. See footnote 2 on page 26-1.

- The HLAT prefix size. The value of this field determines which linear address are subject to HLAT paging. See Section 5.5.1.

These fields exist only on processors that support the 1-setting of the “enable HLAT” VM-execution control.

26.6.23 Fields Related to PASID Translation

Two 64-bit VM-execution control fields are used when the “PASID translation” VM-execution control is 1, enabling translation of PASIDs for executions of ENQCMD and ENQCMDS: the **low PASID directory address** and the **high PASID directory address**. These are the physical addresses of the low PASID directory and the high PASID directory, respectively. These fields exist only on processors that support the 1-setting of the “PASID translation” VM-execution control.

See Section 27.5.8 for information on the PASID-translation process for ENQCMD and ENQCMDS.

26.6.24 Instruction-Timeout Control

On processors that support the 1-setting of the “instruction timeout” VM-execution control, the VM-execution control fields include a 32-bit **instruction-timeout control**. The processor interprets the value of this field as an amount of time as measured in units of crystal clock cycles.¹ If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within this amount of time.

26.6.25 Fields Controlling Virtualization of the IA32_SPEC_CTRL MSR

On processors that support the 1-setting of the “virtualize IA32_SPEC_CTRL” VM-execution control, the VM-execution control fields include the following 64-bit fields:

- IA32_SPEC_CTRL mask.** Setting a bit in this field prevents guest software from modifying the corresponding bit in the IA32_SPEC_CTRL MSR.

1. CPUID.15H:ECX enumerates the nominal frequency of the core crystal clock in Hz.

- **IA32_SPEC_CTRL shadow.** This field contains the value that guest software expects to be in the IA32_SPEC_CTRL MSR.

Section 27.3 discusses how these fields are used in VMX non-root operation.

26.6.26 Fields Controlling SEAM Non-Root Operation

On processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control, the VM-execution control fields include the following fields:

- SEAM-guest KeyID. This 32-bit field contains the key identifier associated with the key used to encrypt SEAM private memory for the current SEAM guest.
- SEAM shared EPT pointer. This 64-bit field contains the address of the EPT paging structure that is the root of the hierarchy of structures that control the translation of shared guest-physical addresses. The format of this field is shown in Table 26-13.

Table 26-13. Format of SEAM Shared EPT Pointer

Bit Position(s)	Field
11:0	Reserved
M-1:12	Bits M-1:12 of the physical address of the 4-KByte aligned EPT paging-structure (an EPT PML4 table with 4-level EPT and an EPT PML5 table with 5-level EPT) ¹
63:M	Reserved

NOTES:

1. M is the value enumerated by CPUID.80000008H:EAX[7:0].

These fields control SEAM non-root operation. They are ignored outside of SEAM.

26.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 26.7.1 and Section 26.7.2.

26.7.1 VM-Exit Controls

The VM-exit controls constitute two vectors that govern the basic operation of VM exits. These are the **primary VM-exit controls** (32 bits) and the **secondary VM-exits controls** (64 bits).

Table 26-14 lists the primary VM-exit controls. See Chapter 28 for complete details of how these controls affect VM exits.

Table 26-14. Definitions of Primary VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit. ¹ This control must be 0 on processors that do not support Intel 64 architecture.

Table 26-14. Definitions of Primary VM-Exit Controls (Contd.)

Bit Position(s)	Name	Description
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is stored in the VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.
23	Clear IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is cleared on VM exit.
24	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM exit or a VMCS packet on an SMM VM exit (see Chapter 35).
25	Clear IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is cleared on VM exit.
26	Clear IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is cleared on VM exit.
27	Clear UINV	This control determines whether UINV is cleared on VM exit.
28	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM exit.
29	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM exit.
30	Save IA32_PERF_GLOBAL_CTL	This control determines whether the IA32_PERF_GLOBAL_CTL MSR is saved on VM exit.
31	Activate secondary controls	This control determines whether the secondary VM-exit controls are used. If this control is 0, the logical processor operates as if all the secondary VM-exit controls were also 0.

NOTES:

1. Since the Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX root operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX root operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix A.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 28.2.1.2).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_EXIT_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-exit controls determines whether the secondary VM-exit controls are used. If that bit is 0, VM entries and VM exits function as if all the secondary VM-exit controls were 0. Processors that support only the 0-setting of bit 31 of the primary VM-exit controls do not support the secondary VM-exit controls.

Table 26-15 lists the secondary VM-exit controls. See Chapter 28 for more details of how these controls affect VM exits.

Table 26-15. Definitions of Secondary VM-Exit Controls

Bit Position(s)	Name	Description
3	Prematurely busy shadow stack	If this control is 1, VM exits that cause a shadow stack to become prematurely busy (see Section 18.2.3, “Supervisor Shadow Stack Token,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1) indicate this fact and save additional information into the VMCS.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_EXIT_CTLD2 (see Appendix A.4.2) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 28.2.1.2).

26.7.2 VM-Exit Controls for MSRs

A VMM may specify lists of MSRs to be stored and loaded on VM exits. The following VM-exit control fields determine how MSRs are stored on VM exits:

- **VM-exit MSR-store count** (32 bits). This field specifies the number of MSRs to be stored on VM exit. It is recommended that this count not exceed 512.¹ Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.
- **VM-exit MSR-store address** (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 26-16. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 26-16. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 29.4 for how this area is used on VM exits.

The following VM-exit control fields determine how MSRs are loaded on VM exits:

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.²
- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 26-16). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 29.6 for how this area is used on VM exits.

1. Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

2. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

26.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 26.8.1 through 26.8.3.

26.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 26-17 lists the controls supported. See Chapter 26 for how these controls affect VM entries.

Table 26-17. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM entry. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA as part of VM entry. ¹ This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMI and SMM is in effect after the VM entry (see Section 33.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.
16	Load IA32_BNDCFGS	This control determines whether the IA32_BNDCFGS MSR is loaded on VM entry.
17	Conceal VMX from PT	If this control is 1, Intel Processor Trace does not produce a paging information packet (PIP) on a VM entry or a VMCS packet on a VM entry that returns from SMM (see Chapter 35).
18	Load IA32_RTIT_CTL	This control determines whether the IA32_RTIT_CTL MSR is loaded on VM entry.
19	Load UINV	This control determines whether UINV is loaded on VM entry.
20	Load CET state	This control determines whether CET-related MSRs and SSP are loaded on VM entry.
21	Load guest IA32_LBR_CTL	This control determines whether the IA32_LBR_CTL MSR is loaded on VM entry.
22	Load PKRS	This control determines whether the IA32_PKRS MSR is loaded on VM entry.
25	Allow SEAM-guest telemetry	If set to 1, core out-of-band telemetry can be enabled upon VM entry to SEAM non-root operation. See Section 19.21.

NOTES:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control. If it is read as 1, every VM exit stores the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control (see Section 29.2).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_ENTRY_CTLS and IA32_VMX_TRUE_ENTRY_CTLS (see Appendix A.5) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 28.2.1.3).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8 and 12. The VMX capability MSR `IA32_VMX_ENTRY_CTLS` always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR `IA32_VMX_TRUE_ENTRY_CTLS` MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

26.8.2 VM-Entry Controls for MSRs

A VMM may specify a list of MSRs to be loaded on VM entries. The following VM-entry control fields manage this functionality:

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.¹
- **VM-entry MSR-load address** (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 26-16. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 28.4 for details of how this area is used on VM entries.

26.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the IDT (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **VM-entry interruption-information field** (32 bits). This field provides details about the event to be injected. Table 26-18 describes the field.

Table 26-18. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception (e.g., #PF) 4: Software interrupt (INT <i>n</i>) 5: Privileged software exception (INT1) 6: Software exception (INT3 or INTO) 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

- The **vector** (bits 7:0) determines which entry in the IDT is used or which other event is injected.
- The **interruption type** (bits 10:8) determines details of how the injection is performed. In general, a VMM should use the type hardware exception for all exceptions **other than** the following:
 - breakpoint exceptions (#BP; a VMM should use the type software exception);
 - overflow exceptions (#OF a VMM should use the use type software exception); and

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR `IA32_VMX_MISC` to determine the number supported (see Appendix A.6).

- those debug exceptions (#DB) that are generated by INT1 (a VMM should use the use type privileged software exception).¹

The type **other event** is used for injection of events that are not delivered through the IDT.²

- For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.
- VM entry injects an event if and only if the **valid bit** (bit 31) is 1. The valid bit in this field is cleared on every VM exit (see Section 29.2).
- VM-entry exception error code** (32 bits). This field is used if and only if the valid bit (bit 31) and the deliver-error-code bit (bit 11) are both set in the VM-entry interruption-information field.
- VM-entry instruction length** (32 bits). For injection of events whose type is software interrupt, software exception, or privileged software exception, this field is used to determine the value of RIP that is pushed on the stack.

See Section 28.6 for details regarding the mechanics of event injection, including the use of the interruption type and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the VM-entry interruption-information field.

26.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of fields that contain information about the most recent VM exit.

On some processors, attempts to write to these fields with VMWRITE fail (see “VMWRITE—Write Field to Virtual-Machine Control Structure” in Chapter 31).³

26.9.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- Exit reason** (32 bits). This field encodes the reason for the VM exit and has the structure given in Table 26-19.

Table 26-19. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason.
16	Always cleared to 0.
24:17	Not currently defined.
25	A VM exit saves this bit as 1 to indicate that the VM exit caused a shadow stack to become prematurely busy.
26	A VM exit saves this bit as 1 to indicate that the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1.
27	A VM exit saves this bit as 1 to indicate that the VM exit was incident to enclave mode.
28	Pending MTF VM exit.
29	VM exit from VMX root operation.
30	Not currently defined.

1. The type hardware exception should be used for all other debug exceptions.

2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with values 1 or 3 for *n*.

3. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

Table 26-19. Format of Exit Reason (Contd.)

Bit Position(s)	Contents
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

- Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix C enumerates the basic exit reasons.
- Bit 16 is always cleared to 0.
- Bit 25 is set to 1 if the “prematurely busy shadow stack” VM-exit control is 1 and the VM exit caused a shadow stack to become prematurely busy (see Section 27.4.3). Otherwise, the bit is cleared.
- Bit 26 is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
- Bit 27 is set to 1 if the VM exit occurred while the logical processor was in enclave mode. See Section 29.2.1 for details.
- Bit 28 is set only by an SMM VM exit (see Section 33.15.2) that took priority over an MTF VM exit (see Section 27.5.2) that would have occurred had the SMM VM exit not occurred. See Section 33.15.2.3.
- Bit 29 is set if and only if the processor was in VMX root operation at the time the VM exit occurred. This can happen only for SMM VM exits. See Section 33.15.2.
- Because some VM-entry failures load processor state from the host-state area (see Section 28.8), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.
- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); task switches; INVEPT; INVLPG; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; and MWAIT. The format of the field depends on the cause of the VM exit. See Section 29.2.1 for details.
- **Guest-linear address** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is used in the following cases:
 - VM exits due to attempts to execute LMSW with a memory operand.
 - VM exits due to attempts to execute INS or OUTS.
 - VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions.
 - Certain VM exits due to EPT violations
 See Section 29.2.1 and Section 33.15.2.3 for details of when and how this field is used.
- **Guest-physical address** (64 bits). This field is used by VM exits due to EPT violations and EPT misconfigurations. See Section 29.2.1 for details of when and how this field is used.

26.9.2 Information for VM Exits Due to Vectored Events

Event-specific information is provided for VM exits due to the following vectored events: exceptions (including those generated by the instructions INT3, INTO, INT1, BOUND, UD0, UD1, UD2, and UDB); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs). This information is provided in the following fields:

- **VM-exit interruption information** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 26-20 describes this field.

Table 26-20. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Not used 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
30:13	Not currently defined
31	Valid

- **VM-exit interruption error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

Section 29.2.2 provides details of how these fields are saved on VM exits.

26.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.¹ This information is provided in the following fields:

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 26-21 describes this field.

Table 26-21. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
30:12	Not currently defined
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits that occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

See Section 29.2.4 provides details of how these fields are saved on VM exits.

1. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 28.6.1.2.

26.9.4 Information for VM Exits Due to Instruction Execution

The following fields are used for VM exits caused by attempts to execute certain instructions in VMX non-root operation:

- **VM-exit instruction length** (32 bits). For VM exits resulting from instruction execution, this field receives the length in bytes of the instruction whose execution led to the VM exit.¹ See Section 29.2.5 for details of when and how this field is used.
- **VM-exit instruction information** (32 bits). This field is used for VM exits due to attempts to execute INS, INVEPT, INVVPID, LIDT, LGDT, LLDT, LTR, OUTS, SIDT, SGDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON.² The format of the field depends on the cause of the VM exit. See Section 29.2.5 for details.

The following fields (64 bits each; 32 bits on processors that do not support Intel 64 architecture) are used only for VM exits due to SMIs that arrive immediately after retirement of I/O instructions. They provide information about that I/O instruction:

- **I/O RCX**. The value of RCX before the I/O instruction started.
- **I/O RSI**. The value of RSI before the I/O instruction started.
- **I/O RDI**. The value of RDI before the I/O instruction started.
- **I/O RIP**. The value of RIP before the I/O instruction started (the RIP that addressed the I/O instruction).

An execution of WRMSRLIST causes a VM exit if it writes to an MSR that cannot be written due to the contents of the MSR bitmaps (see Section 27.1.3). Such VM exits save the data that would have been written to the MSR in a 64-bit field called **MSR data**. This field is supported only on processors that support the 1-setting of the “enable MSR-list instructions” VM-execution control.

26.9.5 VM-Instruction Error Field

The 32-bit **VM-instruction error field** does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

26.10 VMCS TYPES: ORDINARY AND SHADOW

Every VMCS is either an **ordinary VMCS** or a **shadow VMCS**. A VMCS’s type is determined by the shadow-VMCS indicator in the VMCS region (this is the value of bit 31 of the first 4 bytes of the VMCS region; see Table 26-1): 0 indicates an ordinary VMCS, while 1 indicates a shadow VMCS. Shadow VMCSs are supported only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control (see Section 26.6.2).

A shadow VMCS differs from an ordinary VMCS in two ways:

- An ordinary VMCS can be used for VM entry but a shadow VMCS cannot. Attempts to perform VM entry when the current VMCS is a shadow VMCS fail (see Section 28.1).
- The VMREAD and VMWRITE instructions can be used in VMX non-root operation to access a shadow VMCS but not an ordinary VMCS. This fact results from the following:
 - If the “VMCS shadowing” VM-execution control is 0, execution of the VMREAD and VMWRITE instructions in VMX non-root operation always cause VM exits (see Section 27.1.3).
 - If the “VMCS shadowing” VM-execution control is 1, execution of the VMREAD and VMWRITE instructions in VMX non-root operation can access the VMCS referenced by the VMCS link pointer (see Section 32.3).
 - If the “VMCS shadowing” VM-execution control is 1, VM entry ensures that any VMCS referenced by the VMCS link pointer is a shadow VMCS (see Section 28.3.1.5).

1. This field is also used for VM exits that occur during the delivery of a software interrupt or software exception.

2. Whether the processor provides this information on VM exits due to attempts to execute INS or OUTS can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

In VMX root operation, both types of VMCSs can be accessed with the VMREAD and VMWRITE instructions.

Software should not modify the shadow-VMCS indicator in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted (see Section 26.11.1). Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

26.11 SOFTWARE USE OF THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when using a VMCS and related structures. It also provides descriptions of consequences for failing to follow guidelines.

26.11.1 Software Use of Virtual-Machine Control Structures

To ensure proper processor behavior, software should observe certain guidelines when using an active VMCS.

No VMCS should ever be active on more than one logical processor. If a VMCS is to be “migrated” from one logical processor to another, the first logical processor should execute VMCLEAR for the VMCS (to make it inactive on that logical processor and to ensure that all VMCS data are in memory) before the other logical processor executes VMPTRLD for the VMCS (to make it active on the second logical processor).¹ A VMCS that is made active on more than one logical processor may become **corrupted** (see below).

Software should not modify the shadow-VMCS indicator (see Table 26-1) in the VMCS region of a VMCS that is active. Doing so may cause the VMCS to become corrupted. Before modifying the shadow-VMCS indicator, software should execute VMCLEAR for the VMCS to ensure that it is not active.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 26.11.2). Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of accessing VMCS data using ordinary memory operations:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may cause the VMCS to become corrupted (see below).

(Software can avoid these hazards by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.)

If a logical processor leaves VMX operation, any VMCSs active on that logical processor may be corrupted (see below). To prevent such corruption of a VMCS that may be used either after a return to VMX operation or on another logical processor, software should execute VMCLEAR for that VMCS before executing the VMXOFF instruction or removing power from the processor (e.g., as part of a transition to the S3 and S4 power states).

This section has identified operations that may cause a VMCS to become corrupted. These operations may cause the VMCS’s data to become undefined. Behavior may be unpredictable if that VMCS used subsequently on any logical processor. The following items detail some hazards of VMCS corruption:

- VM entries may fail for unexplained reasons or may load undesired processor state.
- The processor may not correctly support VMX non-root operation as documented in Chapter 26 and may generate unexpected VM exits.
- VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

1. As noted in Section 26.1, execution of the VMPTRLD instruction makes a VMCS is active. In addition, VM entry makes active any shadow VMCS referenced by the VMCS link pointer in the current VMCS. If a shadow VMCS is made active by VM entry, it is necessary to execute VMCLEAR for that VMCS before allowing that VMCS to become active on another logical processor.

26.11.2 VMREAD, VMWRITE, and Encodings of VMCS Fields

Every field of the VMCS is associated with a 32-bit value that is its **encoding**. The encoding is provided in an operand to VMREAD and VMWRITE when software wishes to read or write that field. These instructions fail if given, in 64-bit mode, an operand that sets an encoding bit beyond bit 32. See Chapter 31 for a description of these instructions.

The structure of the 32-bit encodings of the VMCS components is determined principally by the width of the fields and their function in the VMCS. See Table 26-22.

Table 26-22. Structure of VMCS Component Encoding

Bit Position(s)	Contents
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields
9:1	Index
11:10	Type: 0: control 1: VM-exit information 2: guest state 3: host state
12	Reserved (must be 0)
14:13	Width: 0: 16-bit 1: 64-bit 2: 32-bit 3: natural-width
31:15	Reserved (must be 0)

The following items detail the meaning of the bits in each encoding:

- **Field width.** Bits 14:13 encode the width of the field.
 - A value of 0 indicates a 16-bit field.
 - A value of 1 indicates a 64-bit field.
 - A value of 2 indicates a 32-bit field.
 - A value of 3 indicates a **natural-width** field. Such fields have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

Fields whose encodings use value 1 are specially treated to allow 32-bit software access to all 64 bits of the field. Such access is allowed by defining, for each such field, an encoding that allows direct access to the high 32 bits of the field. See below.
- **Field type.** Bits 11:10 encode the type of VMCS field: control, guest-state, host-state, or VM-exit information. (The last category also includes the VM-instruction error field.)
- **Index.** Bits 9:1 distinguish components with the same field width and type.
- **Access type.** Bit 0 must be 0 for all fields except for 64-bit fields (those with field-width 1; see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a 64-bit field with field-width 1, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix B gives the encodings of all fields in the VMCS.

The following describes the operation of VMREAD and VMWRITE based on processor mode, VMCS-field width, and access type:

- 16-bit fields:
 - A VMREAD returns the value of the field in bits 15:0 of the destination operand; other bits of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 15:0 of the source operand into the VMCS field; other bits of the source operand are not used.
- 32-bit fields:
 - A VMREAD returns the value of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand into the VMCS field; in 64-bit mode, bits 63:32 of the source operand are not used.
- 64-bit fields and natural-width fields using the full access type outside IA-32e mode.
 - A VMREAD returns the value of bits 31:0 of the field in its destination operand; bits 63:32 of the field are ignored.
 - A VMWRITE writes the value of its source operand to bits 31:0 of the field and clears bits 63:32 of the field.
- 64-bit fields and natural-width fields using the full access type in 64-bit mode (only on processors that support Intel 64 architecture).
 - A VMREAD returns the value of the field in bits 63:0 of the destination operand
 - A VMWRITE writes the value of bits 63:0 of the source operand into the VMCS field.
- 64-bit fields using the high access type.
 - A VMREAD returns the value of bits 63:32 of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
 - A VMWRITE writes the value of bits 31:0 of the source operand to bits 63:32 of the field; in 64-bit mode, bits 63:32 of the source operand are not used.

Software seeking to read a 64-bit field outside IA-32e mode can use VMREAD with the full access type (reading bits 31:0 of the field) and VMREAD with the high access type (reading bits 63:32 of the field); the order of the two VMREAD executions is not important. Software seeking to modify a 64-bit field outside IA-32e mode should first

use VMWRITE with the full access type (establishing bits 31:0 of the field while clearing bits 63:32) and then use VMWRITE with the high access type (establishing bits 63:32 of the field).

26.11.3 Initializing a VMCS

Software should initialize fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

It is not necessary to initialize fields that the logical processor will not use. (For example, it is not necessary to initialize the MSR-bitmap address if the “use MSR bitmaps” VM-execution control is 0.)

A processor maintains some VMCS information that cannot be modified with the VMWRITE instruction; this includes a VMCS’s launch state (see Section 26.1). Such information may be stored in the VMCS data portion of a VMCS region. Because the format of this information is implementation-specific, there is no way for software to know, when it first allocates a region of memory for use as a VMCS region, how the processor will determine this information from the contents of the memory region.

In addition to its other functions, the VMCLEAR instruction initializes any implementation-specific information in the VMCS region referenced by its operand. To avoid the uncertainties of implementation-specific behavior, software should execute VMCLEAR on a VMCS region before making the corresponding VMCS active with VMPTRLD for the first time. (Figure 26-1 illustrates how execution of VMCLEAR puts a VMCS into a well-defined state.)

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry for the first time.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since “migrating” a VMCS from one logical processor to another requires use of VMCLEAR (see Section 26.11.1), which sets the launch state of the VMCS to “clear”, such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

26.11.4 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). While the pointers to these data structures are parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 26.11.1). Exceptions are made for the following data structures (subject to detailed discussion in the sections indicated): EPT paging structures and the data structures used to locate SPP vectors (Section 30.4.3); the virtual-APIC page (Section 31.1); the posted interrupt descriptor (Section 31.6); and the virtualization-exception information area (Section 27.5.7.2).

26.11.5 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)¹ that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

1. The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- The VMXON pointer must not set any bits beyond the processor's physical-address width.^{1,2}

Before executing VMXON, software should write the VMCS revision identifier (see Section 26.2) to the VMXON region. (Specifically, it should write the 31-bit VMCS revision identifier to bits 30:0 of the first 4 bytes of the VMXON region; bit 31 should be cleared to 0.) It need not initialize the VMXON region in any other way. Software should use a separate region for each logical processor and should not access or modify the VMXON region of a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 26.11.1).

-
1. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. It is used to determine the value of and MAXPHYADDR, which constrains the VMXON pointer. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36]. (This is different in secure arbitration mode — SEAM — but VMXON cannot be used in SEAM.) IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode SEAM-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for KeyIDs generally.
 2. If IA32_VMX_BASIC[48] is read as 1, the VMXON pointer must not set any bits in the range 63:32; see Appendix A.1.

20. Updates to Chapter 27, Volume 3C

Change bars and violet text show changes to Chapter 27 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- In Section Section 27.1.2, "Instructions That Cause VM Exits Unconditionally", added SEAMCALL and TDCALL.
- In Section 27.2, "Other Causes of VM Exits", added UDB to Exceptions.
- In Section 27.3, "Changes to Instruction Behavior in VMX Non-Root Operation", provided additional detail for MOV to CR3.
- In Section 27.5.2, "Monitor Trap Flag", added UDB to the footnote for invalid opcode exceptions.
- Updated Section 27.5.8, "PASID Translation" with SEAM relevant information.

In a virtualized environment using VMX, the guest software stack typically runs on a logical processor in VMX non-root operation. This mode of operation is similar to that of ordinary processor operation outside of the virtualized environment. This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits (which bring a logical processor from VMX non-root operation to root operation). The differences between VMX non-root operation and ordinary processor operation are described in the following sections:

- Section 27.1, “Instructions That Cause VM Exits.”
- Section 27.2, “Other Causes of VM Exits.”
- Section 27.3, “Changes to Instruction Behavior in VMX Non-Root Operation.”
- Section 27.4, “Other Changes in VMX Non-Root Operation.”
- Section 27.5, “Features Specific to VMX Non-Root Operation.”
- Section 27.6, “Unrestricted Guests.”

Chapter 27, “VMX Non-Root Operation,” describes the data control structures that govern VMX non-root operation. Chapter 27, “VMX Non-Root Operation,” describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 27, “VMX Non-Root Operation,” describes the operation of VM exits by which the processor transitions from VMX non-root operation to VMX root operation.

Chapter 30, “VMX Support for Address Translation,” describes two features that support address translation in VMX non-root operation. Chapter 31, “APIC Virtualization and Virtual Interrupts,” describes features that support virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC) in VMX non-root operation.

27.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, such VM exits are “fault-like,” meaning that the instruction causing the VM exit does not execute and no processor state is updated by the instruction. Section 29.1 details architectural state in the context of a VM exit.

Section 27.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 27.1.2 identifies instructions that cause VM exits whenever they are executed in VMX non-root operation (and thus can never be executed in VMX non-root operation). Section 27.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 26.6).

27.1.1 Relative Priority of Faults and VM Exits

The following principles describe the ordering between existing faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level,¹ and general-protection exceptions that are based on checking I/O permission bits in the task-state segment (TSS). For example, execution of RDMSR with CPL = 3 generates a general-protection exception and not a VM exit.²
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see LMSW in Section 27.1.3).
- VM exits caused by execution of the INS and OUTS instructions (resulting either because the “unconditional I/O exiting” VM-execution control is 1 or because the “use I/O bitmaps control is 1”) have priority over the following faults:

1. These include faults generated by attempts to execute, in virtual-8086 mode, privileged instructions that are not recognized in that mode.
2. MOV DR is an exception to this rule; see Section 27.1.3.

- A general-protection fault due to the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) being unusable
- A general-protection fault due to an offset beyond the limit of the relevant segment
- An alignment-check exception
- Fault-like VM exits have priority over exceptions other than those mentioned above. For example, RDMSR of a non-existent MSR with CPL = 0 generates a VM exit and not a general-protection exception.

When Section 27.1.2 or Section 27.1.3 (below) identify an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that takes priority over a VM exit.

27.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits when they are executed in VMX non-root operation: CPUID, GETSEC,¹ INVD, and XSETBV. This is also true of instructions introduced with VMX: INVEPT, INVVPID, SEAMCALL, TDCALL, VMCALL,² VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMRESUME, VMXOFF, and VMXON.

27.1.3 Instructions That Cause VM Exits Conditionally

Certain instructions cause VM exits in VMX non-root operation depending on the setting of the VM-execution controls. The following instructions can cause “fault-like” VM exits based on the conditions described:³

- **CLTS.** The CLTS instruction causes a VM exit if the bits in position 3 (corresponding to CR0.TS) are set in both the CR0 guest/host mask and the CR0 read shadow.
- **ENCLS.** The ENCLS instruction causes a VM exit if the “enable ENCLS exiting” VM-execution control is 1 and one of the following is true:
 - The value of EAX is less than 63 and the corresponding bit in the ENCLS-exiting bitmap is 1 (see Section 26.6.16).
 - The value of EAX is greater than or equal to 63 and bit 63 in the ENCLS-exiting bitmap is 1.
- **ENQCMD, ENQCMDS.** The behavior of each of these instructions is determined by the setting of the “PASID translation” VM-execution control. If that control is 0, the instruction executes normally. If the control is 1, instruction behavior is modified and may cause a VM exit. See Section 27.5.8.
- **HLT.** The HLT instruction causes a VM exit if the “HLT exiting” VM-execution control is 1.
- **IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD.** The behavior of each of these instructions is determined by the settings of the “unconditional I/O exiting” and “use I/O bitmaps” VM-execution controls:
 - If both controls are 0, the instruction executes normally.
 - If the “unconditional I/O exiting” VM-execution control is 1 and the “use I/O bitmaps” VM-execution control is 0, the instruction causes a VM exit.
 - If the “use I/O bitmaps” VM-execution control is 1, the instruction causes a VM exit if it attempts to access an I/O port corresponding to a bit set to 1 in the appropriate I/O bitmap (see Section 26.6.4). If an I/O operation “wraps around” the 16-bit I/O-port space (accesses ports FFFFH and 0000H), the I/O instruction causes a VM exit (the “unconditional I/O exiting” VM-execution control is ignored if the “use I/O bitmaps” VM-execution control is 1).

1. An execution of GETSEC in VMX non-root operation causes a VM exit if CR4.SMXE[Bit 14] = 1 regardless of the value of CPL or RAX. An execution of GETSEC causes an invalid-opcode exception (#UD) if CR4.SMXE[Bit 14] = 0.

2. Under the dual-monitor treatment of SMIs and SMM, executions of VMCALL cause SMM VM exits in VMX root operation outside SMM. See Section 33.15.2.

3. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 26.6.2.

See Section 27.1.1 for information regarding the priority of VM exits relative to faults that may be caused by the INS and OUTS instructions.

- **INVLPG.** The INVLPG instruction causes a VM exit if the “INVLPG exiting” VM-execution control is 1.
- **INVPCID.** The INVPCID instruction causes a VM exit if the “INVLPG exiting” and “enable INVPCID” VM-execution controls are both 1.
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the “descriptor-table exiting” VM-execution control is 1.
- **LMSW.** In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding bit in the CR0 read shadow. LMSW never clears bit 0 of CR0 (CR0.PE); thus, LMSW causes a VM exit if either of the following are true:
 - The bits in position 0 (corresponding to CR0.PE) are set in both the CR0 guest/host mask and the source operand, and the bit in position 0 is clear in the CR0 read shadow.
 - For any bit position in the range 3:1, the bit in that position is set in the CR0 guest/host mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **LOADIWKEY.** The LOADIWKEY instruction causes a VM exit if the “LOADIWKEY exiting” VM-execution control is 1.
- **MONITOR.** The MONITOR instruction causes a VM exit if the “MONITOR exiting” VM-execution control is 1.
- **MOV from CR3.** The MOV from CR3 instruction causes a VM exit if the “CR3-store exiting” VM-execution control is 1. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
- **MOV from CR8.** The MOV from CR8 instruction causes a VM exit if the “CR8-store exiting” VM-execution control is 1.
- **MOV to CR0.** The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- **MOV to CR3.** The MOV to CR3 instruction causes a VM exit unless the “CR3-load exiting” VM-execution control is 0 or the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Only the first n CR3-target values are considered, where n is the CR3-target count. If the “CR3-load exiting” VM-execution control is 1 and the CR3-target count is 0, MOV to CR3 always causes a VM exit.

The first processors to support the virtual-machine extensions supported only the 1-setting of the “CR3-load exiting” VM-execution control. These processors always consult the CR3-target controls to determine whether an execution of MOV to CR3 causes a VM exit.

- **MOV to CR4.** The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- **MOV to CR8.** The MOV to CR8 instruction causes a VM exit if the “CR8-load exiting” VM-execution control is 1.
- **MOV DR.** The MOV DR instruction causes a VM exit if the “MOV-DR exiting” VM-execution control is 1. Such VM exits represent an exception to the principles identified in Section 27.1.1 in that they take priority over the following: general-protection exceptions based on privilege level; and invalid-opcode exceptions that occur because CR4.DE=1 and the instruction specified access to DR4 or DR5.
- **MWAIT.** The MWAIT instruction causes a VM exit if the “MWAIT exiting” VM-execution control is 1. If this control is 0, the behavior of the MWAIT instruction may be modified (see Section 27.3).
- **PAUSE.** The behavior of each of this instruction depends on CPL and the settings of the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls:
 - CPL = 0.
 - If the “PAUSE exiting” and “PAUSE-loop exiting” VM-execution controls are both 0, the PAUSE instruction executes normally.
 - If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit (the “PAUSE-loop exiting” VM-execution control is ignored if CPL = 0 and the “PAUSE exiting” VM-execution control is 1).

- If the “PAUSE exiting” VM-execution control is 0 and the “PAUSE-loop exiting” VM-execution control is 1, the following treatment applies.

The processor determines the amount of time between this execution of PAUSE and the previous execution of PAUSE at CPL 0. If this amount of time exceeds the value of the VM-execution control field PLE_Gap, the processor considers this execution to be the first execution of PAUSE in a loop. (It also does so for the first execution of PAUSE at CPL 0 after VM entry.)

Otherwise, the processor determines the amount of time since the most recent execution of PAUSE that was considered to be the first in a loop. If this amount of time exceeds the value of the VM-execution control field PLE_Window, a VM exit occurs.

For purposes of these computations, time is measured based on a counter that runs at the same rate as the timestamp counter (TSC).

— CPL > 0.

- If the “PAUSE exiting” VM-execution control is 0, the PAUSE instruction executes normally.
- If the “PAUSE exiting” VM-execution control is 1, the PAUSE instruction causes a VM exit.

The “PAUSE-loop exiting” VM-execution control is ignored if CPL > 0.

- **PCONFIG.** The PCONFIG instruction causes a VM exit if the “enable PCONFIG” VM-execution control is 1 and one of the following is true:

- The value of EAX is less than 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1 (see Section 26.6.17).
- The value of EAX is greater than or equal to 63 and bit 63 in the PCONFIG-exiting bitmap is 1.

If the “enable PCONFIG” VM-execution control is 1 and neither of the previous items hold, the PCONFIG instruction executes normally.

- **RDMSR.** The RDMSR instruction causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in read bitmap for low MSRs is 1, where *n* is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

See Section 26.6.9 for details regarding how these bitmaps are identified.

- **RDMSRLIST.** The RDMSRLIST instruction causes a VM exit if the “enable MSR-list instructions” VM-execution control is 1 and the “use MSR bitmaps” VM-execution control is 0. If both controls are 1, the instruction reads one MSR at a time normally, storing the value read to memory and clearing the corresponding bit in RCX. An attempt to read MSR *X* causes a VM exit if any of the following are true:
 - *X* is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - *X* is in the range 00000000H – 00001FFFH and bit *X* in read bitmap for low MSRs is 1.
 - *X* is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of *X* & 00001FFFH.

If an attempt to read an MSR causes a VM exit, the corresponding bit in RCX is not cleared, the MSR is not read, and no value is stored to memory.

- **RDPMC.** The RDPMC instruction causes a VM exit if the “RDPMC exiting” VM-execution control is 1.
- **RDRAND.** The RDRAND instruction causes a VM exit if the “RDRAND exiting” VM-execution control is 1.
- **RDSEED.** The RDSEED instruction causes a VM exit if the “RDSEED exiting” VM-execution control is 1.
- **RDTSC.** The RDTSC instruction causes a VM exit if the “RDTSC exiting” VM-execution control is 1.
- **RDTSCP.** The RDTSCP instruction causes a VM exit if the “RDTSC exiting” and “enable RDTSCP” VM-execution controls are both 1.
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).¹

- **TPAUSE.** The TPAUSE instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **UMWAIT.** The UMWAIT instruction causes a VM exit if the “RDTSC exiting” and “enable user wait and pause” VM-execution controls are both 1.
- **VMREAD.** The VMREAD instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMREAD bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 26.6.15 for details regarding how the VMREAD bitmap is identified.

If the VMREAD instruction does not cause a VM exit, it reads from the VMCS referenced by the VMCS link pointer. See Chapter 32, “VMREAD—Read Field from Virtual-Machine Control Structure” for details of the operation of the VMREAD instruction.

- **VMWRITE.** The VMWRITE instruction causes a VM exit if any of the following are true:
 - The “VMCS shadowing” VM-execution control is 0.
 - Bits 63:15 (bits 31:15 outside 64-bit mode) of the register source operand are not all 0.
 - Bit n in VMWRITE bitmap is 1, where n is the value of bits 14:0 of the register source operand. See Section 26.6.15 for details regarding how the VMWRITE bitmap is identified.

If the VMWRITE instruction does not cause a VM exit, it writes to the VMCS referenced by the VMCS link pointer. See Chapter 32, “VMWRITE—Write Field to Virtual-Machine Control Structure” for details of the operation of the VMWRITE instruction.

- **WBINVD.** The WBINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WBNOINVD.** The WBNOINVD instruction causes a VM exit if the “WBINVD exiting” VM-execution control is 1.
- **WRMSR, WRMSRNS.** Execution of one of these instructions causes a VM exit if any of the following are true:
 - The “use MSR bitmaps” VM-execution control is 0.
 - The value of ECX is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
 - The value of ECX is in the range 00000000H – 00001FFFH and bit n in write bitmap for low MSRs is 1, where n is the value of ECX.
 - The value of ECX is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of ECX & 00001FFFH.

See Section 26.6.9 for details regarding how these bitmaps are identified.

- **WRMSRLIST.** The WRMSRLIST instruction causes a VM exit if the “enable MSR-list instructions” VM-execution control is 1 and the “use MSR bitmaps” VM-execution control is 0. In this case, the register operands of the instructions are not used, memory is not read, and no MSRs are modified.

If both controls are 1, the instruction writes one MSR at a time normally, using a value read from memory and clearing the corresponding bit in RCX. An attempt to write MSR X causes a VM exit if any of the following are true:

- X is not in the ranges 00000000H – 00001FFFH and C0000000H – C0001FFFH.
- X is in the range 00000000H – 00001FFFH and bit X in write bitmap for low MSRs is 1.
- X is in the range C0000000H – C0001FFFH and bit n in write bitmap for high MSRs is 1, where n is the value of X & 00001FFFH.

Such a VM exit occurs after the data that would be written to the MSR is read from memory, but the corresponding bit in RCX is not cleared and the MSR is not written. The data that would have been written to the MSR is saved into the MSR-data field of the VMCS (see Section 29.2.5).

1. Execution of the RSM instruction outside SMM causes an invalid-opcode exception regardless of whether the processor is in VMX operation. It also does so in VMX root operation in SMM; see Section 33.15.3.

- **XRSTORS.** The XRSTORS instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 26.6.20).
- **XSAVES.** The XSAVES instruction causes a VM exit if the “enable XSAVES/XRSTORS” VM-execution control is 1 and any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap (see Section 26.6.20).

27.2 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:¹

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 26.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT1, INT3, INTO, BOUND, UD0, UD1, and UD2, and UDB.²

Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.

- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the “external-interrupt exiting” VM-execution control is 1 (see Section 31.6 for an exception.) Otherwise, the processor handles the interrupt normally.³ (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The processor does handle the interrupt and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the “NMI exiting” VM-execution control is 1. Otherwise, it is delivered using descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of

1. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 26.6.2.

2. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

3. Normal handling usually means delivery through the IDT, but it could also mean treatment of the interrupt as a user-interrupt notification.

VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)

- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 27.4.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 33.15.2.¹
- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 27.5.1 for details of operation of the VMX-preemption timer.

Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the “NMI-window exiting” VM-execution control and lower priority events.

These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

- **Bus locks.** Assertion of a bus lock (see Section 10.1.2) causes a VM exit if the “VMM bus-lock detection” VM-execution control is 1. Such a VM exit is trap-like because it is generated after execution of an instruction that asserts a bus lock. The VM exit thus does not prevent assertion of the bus lock. These VM exits take priority over system-management interrupts (SMIs), INIT signals, and lower priority events.
- **Instruction timeout.** If the “instruction timeout” VM-execution control is 1, a VM exit occurs if certain operations prevent the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control VM-execution control field (see Section 26.6.24).

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 26-3).

Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.

- If the “NMI-window exiting” VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS and no blocking of events by STI (see Table 26-3).

VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

Conditions necessary for some of the VM exits identified in this section may hold immediately after VM entry. If they do, a corresponding VM exit occurs at that time.

27.3 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:²

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:

1. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 33.14.1.

- If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 25.8), in which case CLTS causes a general-protection exception.
- If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
- If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit.
- **ENQCMD, ENQCMLS.** Each of these instructions performs a 64-byte enqueue store that includes a PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMLS, the PASID is normally read from memory.

The behavior of each of these instructions (and in particular the PASID value used for the enqueue store) is determined by the setting of the “PASID translation” VM-execution control:

- If the “PASID translation” VM-execution control is 0, the instruction operates normally.
- If the “PASID translation” VM-execution control is 1, the PASID value used for the enqueue store is determined by the PASID-translation process described in Section 27.5.8. (Note the PASID translation may result in a VM exit, in which case the enqueue store is not performed.)

An execution of ENQCMD or ENQCMLS performs PASID translation only after checking for conditions that may result in general-protection exception (the check of IA32_PASID.Valid for ENQCMD; the privilege-level check for ENQCMLS), after loading the instruction's source operand from memory, and thus after any faults or VM exits that the loading may cause (e.g., page faults or EPT violations). PASID translation occurs before the actual enqueue store and thus before any faults or VM exits that it may cause.

- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the “enable INVPCID” VM-execution control:
 - If the “enable INVPCID” VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable INVPCID” VM-execution control is 1, treatment is based on the setting of the “INVLPG exiting” VM-execution control:
 - If the “INVLPG exiting” VM-execution control is 0, INVPCID operates normally.
 - If the “INVLPG exiting” VM-execution control is 1, INVPCID causes a VM exit.
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 26-3) is determined by the settings of the “NMI exiting” and “virtual NMIs” VM-execution controls:
 - If the “NMI exiting” VM-execution control is 0, IRET operates normally and unblocks NMIs. (If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 28.2.1.1.)
 - If the “NMI exiting” VM-execution control is 1, IRET does not affect blocking of NMIs. If, in addition, the “virtual NMIs” VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

The unblocking of NMIs or virtual NMIs specified above occurs even if IRET causes a fault.

- **LMSW.** Outside of VMX non-root operation, LMSW loads its source operand into CR0[3:0], but it does not clear CR0.PE if that bit is set. In VMX non-root operation, an execution of LMSW that does not cause a VM exit (see Section 27.1.3) leaves unmodified any bit in CR0[3:0] corresponding to a bit set in the CR0 guest/host mask. An attempt to set any other bit in CR0[3:0] to a value not supported in VMX operation (see Section 25.8) causes a general-protection exception. Attempts to clear CR0.PE are ignored without fault.
- **MOV from CR0.** The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read

-
2. Items in this section may refer to secondary processor-based VM-execution controls and tertiary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the secondary processor-based VM-execution controls were all 0; similarly, if bit 17 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the tertiary processor-based VM-execution controls were all 0. See Section 26.6.2.

shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

Depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **MOV from CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV from CR3 does not cause a VM exit (see Section 27.1.3), the value loaded from CR3 is a guest-physical address; see Section 30.3.1.
- **MOV from CR4.** The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit clear in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow. Thus, if every bit is cleared in the CR4 guest/host mask, MOV from CR4 reads normally from CR4; if every bit is set in the CR4 guest/host mask, MOV from CR4 returns the value of the CR4 read shadow. Depending on the contents of the CR4 guest/host mask and the CR4 read shadow, bits may be set in the destination that would never be set when reading directly from CR4.
- **MOV from CR8.** If the MOV from CR8 instruction does not cause a VM exit (see Section 27.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 31.3.
- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 27.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. Treatment of attempts to modify other bits in CR0 depends on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 to a value not supported in VMX operation (see Section 25.8).
 - If the control is 1, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 other than bit 0 (PE) or bit 31 (PG) to a value not supported in VMX operation. It remains the case, however, that MOV to CR0 causes a general-protection exception if it would result in CR0.PE = 0 and CR0.PG = 1 or if it would result in CR0.PG = 1, CR4.PAE = 0, and IA32_EFER.LME = 1.
- **MOV to CR3.** If the “enable EPT” VM-execution control is 1 and an execution of MOV to CR3 does not cause a VM exit (see Section 27.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 30.3.1. The following details apply:
 - An attempt to set a reserved bit in CR3[63:M] results in #GP(0), where M is the value enumerated in CPUID.80000008H:EAX[7:0]. (Unlike other cases, the value M is not reduced outside SEAM.)
 - If PAE paging is not being used, the instruction does not use the guest-physical address to access memory and it does not cause it to be translated through EPT.¹
 - If PAE paging is being used, the instruction translates the guest-physical address through EPT and uses the result to load the four (4) page-directory-pointer-table entries (PDPTes). The instruction does not use the guest-physical addresses the PDPTes to access memory and it does not cause them to be translated through EPT. An attempt to set a PDPTE bit in positions 63:M results in #GP(0), where M is defined above (M is not reduced outside SEAM).
- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 27.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. Such an execution causes a general-protection exception if it attempts to set any bit in CR4 (not corresponding to a bit set in the CR4 guest/host mask) to a value not supported in VMX operation (see Section 25.8).
- **MOV to CR8.** If the MOV to CR8 instruction does not cause a VM exit (see Section 27.1.3), its behavior is modified if the “use TPR shadow” VM-execution control is 1; see Section 31.3.
- **MWAIT.** Behavior of the MWAIT instruction (which always causes an invalid-opcode exception—#UD—if CPL > 0) is determined by the setting of the “MWAIT exiting” VM-execution control:
 - If the “MWAIT exiting” VM-execution control is 1, MWAIT causes a VM exit.
 - If the “MWAIT exiting” VM-execution control is 0, MWAIT operates normally if one of the following are true: (1) ECX[0] is 0; (2) RFLAGS.IF = 1; or both of the following are true: (a) the “interrupt-window exiting”

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

VM-execution control is 0; and (b) the logical processor has not recognized a pending virtual interrupt (see Section 29.2.1).

- If the “MWAIT exiting” VM-execution control is 0, ECX[0] = 1, and RFLAGS.IF = 0, MWAIT does not cause the processor to enter an implementation-dependent optimized state if either the “interrupt-window exiting” VM-execution control is 1 or the logical processor has recognized a pending virtual interrupt; instead, control passes to the instruction following the MWAIT instruction.
- **PCONFIG.** Behavior of the PCONFIG instruction is determined by the setting of the “enable PCONFIG” VM-execution control:
 - If the “enable PCONFIG” VM-execution control is 0, PCONFIG causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable PCONFIG” VM-execution control is 1, PCONFIG may cause a VM exit as specified in Section 27.1.3; if it does not cause such a VM exit, it operates normally.
- **RDMSR.** Section 27.1.3 identifies when executions of the RDMSR instruction cause VM exits. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
 - If ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR), the value returned by the instruction is determined by the setting of the “use TSC offsetting” VM-execution control:
 - If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
 - If the control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDMSR loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDMSR first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

The 1-setting of the “use TSC-offsetting” VM-execution control does not affect executions of RDMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions return the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.

 - If ECX contains 48H (indicating the IA32_SPEC_CTRL MSR), the value returned by the instruction is determined by the setting of the “virtualize IA32_SPEC_CTRL” VM-execution control:
 - If the control is 0, RDMSR operates normally, loading EAX:EDX with the value of the IA32_SPEC_CTRL MSR.
 - If the control is 1, the value returned is that of the IA32_SPEC_CTRL shadow field in the VMCS.
 - If ECX is in the range 800H–8FFH (indicating an APIC MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 31.5.- **RDMSRLIST.** Behavior of the RDMSRLIST instruction is determined first by the setting of the “enable MSR-list instructions” VM-execution control:
 - If the “enable MSR-list instructions” VM-execution control is 0, RDMSRLIST causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable MSR-list instructions” VM-execution control is 1, the instruction causes a general-protection exception (#GP) normally if CPL > 0. Otherwise, its operation depends on the setting of the “use MSR bitmaps” VM-execution control:
 - If the control is 0, the instruction causes a VM exit.
 - If the control is 1, the instruction commences normally, reading one MSR at a time. Reads of certain MSRs are treated specially as described above for RDMSR. In addition, attempts to access specific MSRs may cause VM exits; see Section 27.1.3 for details.- **RDPID.** Behavior of the RDPID instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:

- If the “enable RDTSCP” VM-execution control is 0, RDPID causes an invalid-opcode exception (#UD).
 - If the “enable RDTSCP” VM-execution control is 1, RDPID operates normally.
 - **RDTSC.** Behavior of the RDTSC instruction is determined by the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSC operates normally.
 - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTSC loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTSC first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.
 - If the “RDTSC exiting” VM-execution control is 1, RDTSC causes a VM exit.
 - **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the “enable RDTSCP” VM-execution control:
 - If the “enable RDTSCP” VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable RDTSCP” VM-execution control is 1, treatment is based on the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
 - If both controls are 0, RDTSCP operates normally.
 - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, the value returned is determined by the setting of the “use TSC scaling” VM-execution control:
 - If the control is 0, RDTSCP loads EAX:EDX with the sum of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset.
 - If the control is 1, RDTSCP first computes the product of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC multiplier. It then shifts the value of the product right 48 bits and loads EAX:EDX with the sum of that shifted value and the value of the TSC offset.

In either case, RDTSCP also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.

 - If the “RDTSC exiting” VM-execution control is 1, RDTSCP causes a VM exit. - **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, SMSW reads normally from CR0; if every bit is set in the CR0 guest/host mask, SMSW returns the value of the CR0 read shadow.
- Note the following: (1) for any memory destination or for a 16-bit register destination, only the low 16 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:16 of a register destination are left unchanged); (2) for a 32-bit register destination, only the low 32 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:32 of the destination are cleared); and (3) depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.
- **TPAUSE.** Behavior of the TPAUSE instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:
 - If the “enable user wait and pause” VM-execution control is 0, TPAUSE causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
 - If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:

- If the “RDTSC exiting” VM-execution control is 0, the instruction delays for an amount of time called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).

If IA32_UMWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UMWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UMWAIT_CONTROL,FFFFFFFFCH).

The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:

- If either control is 0, the physical delay is the virtual delay.
- If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.

- If the “RDTSC exiting” VM-execution control is 1, TPAUSE causes a VM exit.

- **UMONITOR.** Behavior of the UMONITOR instruction is determined by the setting of the “enable user wait and pause” VM-execution control:

- If the “enable user wait and pause” VM-execution control is 0, UMONITOR causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
- If the “enable user wait and pause” VM-execution control is 1, UMONITOR operates normally.

- **UMWAIT.** Behavior of the UMWAIT instruction is determined first by the setting of the “enable user wait and pause” VM-execution control:

- If the “enable user wait and pause” VM-execution control is 0, UMWAIT causes an invalid-opcode exception (#UD). This exception takes priority over any exception the instruction may incur.
- If the “enable user wait and pause” VM-execution control is 1, treatment is based on the setting of the “RDTSC exiting” VM-execution control:

- If the “RDTSC exiting” VM-execution control is 0, and if the instruction causes a delay, the amount of time delayed is called here the **physical delay**. The physical delay is first computed by determining the **virtual delay** (the time to delay relative to the guest’s timestamp counter).

If IA32_UMWAIT_CONTROL[31:2] is zero, the virtual delay is the value in EDX:EAX minus the value that RDTSC would return (see above); if IA32_UMWAIT_CONTROL[31:2] is not zero, the virtual delay is the minimum of that difference and AND(IA32_UMWAIT_CONTROL,FFFFFFFFCH).

The physical delay depends upon the settings of the “use TSC offsetting” and “use TSC scaling” VM-execution controls:

- If either control is 0, the physical delay is the virtual delay.
- If both controls are 1, the virtual delay is multiplied by 2^{48} (using a shift) to produce a 128-bit integer. That product is then divided by the TSC multiplier to produce a 64-bit integer. The physical delay is that quotient.

- If the “RDTSC exiting” VM-execution control is 1, UMWAIT causes a VM exit.

- **WRMSR, WRMSRNS.** Section 27.1.3 identifies when an execution of WRMSR or WRMSRNS would cause a VM exit. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:

- If ECX contains 48H (indicating the IA32_SPEC_CTRL MSR), instruction behavior depends on the setting of the “virtualize IA32_SPEC_CTRL” VM-execution control:
 - If the control is 0, WRMSR and WRMSRNS operate normally, loading the IA32_SPEC_CTRL MSR with the value in EAX:EDX.
 - If the control is 1, the instruction will attempt to write the IA32_SPEC_CTRL MSR using the instruction’s source operand, but it will attempt to modify only those bits in positions corresponding to bits cleared in the IA32_SPEC_CTRL mask field in the VMCS.

Specifically, the instruction attempts to write the MSR with the following value:

(MSR_VAL & ISC_MASK) OR (SRC & NOT ISC_MASK),

where MSR_VAL is the original value of the MSR, ISC_MASK is the IA32_SPEC_CTRL mask, and SRC is the instruction's source operand.

Any fault that would result from writing that value to the MSR (e.g., due to a reserved-bit violation) occurs normally. Otherwise, the value is written to the MSR.

Such a write to the MSR will have any side effects that would occur normally had the MSR been written with the value indicated above (including any side effects that may result from writing unchanged values to the masked bits).

If the write completes without a fault, the unmodified value of the source operand is written to the IA32_SPEC_CTRL shadow field in the VMCS.

- If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
- On processors that support Intel PT but which do not allow it to be used in VMX operation, if ECX contains 570H (indicating the IA32_RTIT_CTL MSR), the instruction causes a general-protection exception.¹
- If ECX contains 808H (indicating the TPR MSR), 80BH (the EOI MSR), 830H (the ICR MSR), or 83FH (the self-IPI MSR), instruction behavior may be modified if the “virtualize x2APIC mode” VM-execution control is 1; see Section 31.5.
- **WRMSRLIST.** Behavior of the WRMSRLIST instruction is determined first by the setting of the “enable MSR-list instructions” VM-execution control:
 - If the “enable MSR-list instructions” VM-execution control is 0, WRMSRLIST causes an invalid-opcode exception (#UD). This exception takes priority over any other exception the instruction may incur.
 - If the “enable MSR-list instructions” VM-execution control is 1, the instruction causes a general-protection exception (#GP) normally if CPL > 0. Otherwise, its operation depends on the setting of the “use MSR bitmaps” VM-execution control:
 - If the control is 0, the instruction causes a VM exit.
 - If the control is 1, the instruction commences normally, writing one MSR at a time. Writes to certain MSRs are treated specially as described above for WRMSR and WRMSRNS. In addition, attempts to access specific MSRs may cause VM exits; see Section 27.1.3 for details.
- **XRSTORS.** Behavior of the XRSTORS instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XRSTORS causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 26.6.20):
 - XRSTORS causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XRSTORS operates normally.
- **XSAVES.** Behavior of the XSAVES instruction is determined first by the setting of the “enable XSAVES/XRSTORS” VM-execution control:
 - If the “enable XSAVES/XRSTORS” VM-execution control is 0, XSAVES causes an invalid-opcode exception (#UD).
 - If the “enable XSAVES/XRSTORS” VM-execution control is 1, treatment is based on the value of the XSS-exiting bitmap (see Section 26.6.20):
 - XSAVES causes a VM exit if any bit is set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
 - Otherwise, XSAVES operates normally.

1. Software should read the VMX capability MSR IA32_VMX_MISC to determine whether the processor allows Intel PT to be used in VMX operation (see Appendix A.6).

27.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking, task switches, and certain shadow-stack updates may differ in VMX non-root operation as described in the following sections.

27.4.1 Event Blocking

Event blocking is modified in VMX non-root operation as follows:

- If the “external-interrupt exiting” VM-execution control is 1, RFLAGS.IF does not control the blocking of external interrupts. In this case, an external interrupt that is not blocked for other reasons causes a VM exit (even if RFLAGS.IF = 0).
- If the “external-interrupt exiting” VM-execution control is 1, external interrupts may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).
- If the “NMI exiting” VM-execution control is 1, non-maskable interrupts (NMIs) may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).

27.4.2 Treatment of Task Switches

Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. However, the following checks are performed (in the order indicated), possibly resulting in a fault, before there is any possibility of a VM exit due to task switch:

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
 - a. If CALL, INT n , INT1, INT3, INTO, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
 - b. If CALL, INT n , INT3, INTO, or JMP accesses a task gate outside IA-32e mode, privilege-levels checks are performed on the task gate but, if they pass, privilege levels are not checked on the referenced task-state segment (TSS) descriptor.
 - c. If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.
 - d. If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
 - e. If a non-maskable interrupt (NMI), an exception, or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.
 - f. If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP) and overflow exceptions (#OF), or an external interrupt accesses a task gate in the IDT outside IA-32e mode, no privilege checks are performed.
 - g. If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.
 - h. If IRET is executed with RFLAGS.NT = 1 outside IA-32e mode, a TSS descriptor is accessed directly and no privilege checks are made.
2. Checks are made on the new TSS selector (for example, that is within GDT limits).
3. The new TSS descriptor is read. (A page fault results if a relevant GDT page is not present).
4. The TSS descriptor is checked for proper values of type (depends on type of task switch), P bit, S bit, and limit.

Only if checks 1–4 all pass (do not generate faults) might a VM exit occur. However, the ordering between a VM exit due to a task switch and a page fault resulting from accessing the old TSS or the new TSS is implementation-specific. Some processors may generate a page fault (instead of a VM exit due to a task switch) if accessing either TSS would cause a page fault. Other processors may generate a VM exit due to a task switch even if accessing either TSS would cause a page fault.

If an attempt at a task switch through a task gate in the IDT causes an exception (before generating a VM exit due to the task switch) and that exception causes a VM exit, information about the event whose delivery that accessed the task gate is recorded in the IDT-vectoring information fields and information about the exception that caused

the VM exit is recorded in the VM-exit interruption-information fields. See Section 29.2. The fact that a task gate was being accessed is not recorded in the VMCS.

If an attempt at a task switch through a task gate in the IDT causes VM exit due to the task switch, information about the event whose delivery accessed the task gate is recorded in the IDT-vectoring fields of the VMCS. Since the cause of such a VM exit is a task switch and not an interruption, the valid bit for the VM-exit interruption information field is 0. See Section 29.2.

27.4.3 Shadow-Stack Updates

As noted in Section 18.2.3, “Supervisor Shadow Stack Token,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, a switch of shadow stack may occur as part of IDT event delivery or an execution of far CALL that changes the CPL, or as part of IDT event delivery that uses the interrupt stack table (IST).

As part of the shadow-stack switch, the processor gains exclusive access to the new stack through manipulation of the **supervisor shadow stack token** located at the base of the new shadow stack. The processor reads the token and, among other things, confirms that bit 0 of the token (its **busy bit**) is 0. If the busy bit is already 1, the transition (event delivery or far CALL) cause a general-protection fault and does not complete. If the busy bit is 0, the transition sets the busy bit by writing to the token in memory. (The update is atomic with the original read of the token.)

If the transition commenced with $CPL < 3$, it will follow the token update by pushing three items on the new shadow stack (for the old values of the CS selector, instruction pointer, and shadow-stack pointer). As noted in Section 18.2.3 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, if any of the pushes causes a VM exit, the processor will revert to the old shadow stack and the busy bit in the new shadow stack’s token remains set. The new shadow stack is said to be prematurely busy.

If the “prematurely busy shadow stack” VM-exit control is 1, a VM exit that results in a shadow stack becoming prematurely busy will indicate that fact through information saved in the VMCS. See Section 29.2.1.

27.5 FEATURES SPECIFIC TO VMX NON-ROOT OPERATION

Some VM-execution controls support features that are specific to VMX non-root operation. These are the VMX-preemption timer (Section 27.5.1) and the monitor trap flag (Section 27.5.2), translation of guest-physical addresses (Section 27.5.3 and Section 27.5.4), APIC virtualization (Section 27.5.5), VM functions (Section 27.5.6), and virtualization exceptions (Section 27.5.7).

27.5.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 28.7.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 27.2).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VMX-preemption timer operates in the C-states C0, C1, and C2; it also operates in the shutdown and wait-for-SIPI states. If the timer counts down to zero in any state other than the wait-for SIPI state, the logical processor transitions to the C0 C-state and causes a VM exit; the timer does not cause a VM exit if it counts down to zero in the wait-for-SIPI state. The timer is not decremented in C-states deeper than C2.

Treatment of the timer in the case of system management interrupts (SMIs) and system-management mode (SMM) depends on whether the treatment of SMIs and SMM:

- If the default treatment of SMIs and SMM (see Section 33.14) is active, the VMX-preemption timer counts across an SMI to VMX non-root operation, subsequent execution in SMM, and the return from SMM via the RSM instruction. However, the timer can cause a VM exit only from VMX non-root operation. If the timer expires

during SMI, in SMM, or during RSM, a timer-induced VM exit occurs immediately after RSM with its normal priority unless it is blocked based on activity state (Section 27.2).

- If the dual-monitor treatment of SMIs and SMM (see Section 33.15) is active, transitions into and out of SMM are VM exits and VM entries, respectively. The treatment of the VMX-preemption timer by those transitions is mostly the same as for ordinary VM exits and VM entries; Section 33.15.2 and Section 33.15.4 detail some differences.

27.5.2 Monitor Trap Flag

The **monitor trap flag** is a debugging feature that causes VM exits to occur on certain instruction boundaries in VMX non-root operation. Such VM exits are called **MTF VM exits**. An MTF VM exit may occur on an instruction boundary in VMX non-root operation as follows:

- If the “monitor trap flag” VM-execution control is 1 and VM entry is injecting a vectored event (see Section 28.6.1), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry.
- If VM entry is injecting a pending MTF VM exit (see Section 28.6.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
 - If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
 - If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is the XBEGIN instruction. In this case, an MTF VM exit is pending at the fallback instruction address of the XBEGIN instruction. This behavior applies regardless of whether advanced debugging of RTM transactional regions has been enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is neither a REP-prefixed string instruction or the XBEGIN instruction:
 - If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).¹
 - If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT1, INT3, or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

1. This item includes the cases of an invalid opcode exception—#UD—generated by the UD0, UD1, UD2, and UDB instructions and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 42.2 for details.

27.5.3 Translation of Guest-Physical Addresses Using EPT

The extended page-table mechanism (EPT) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as guest-physical addresses and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Details of the EPT mechanism are given in Section 30.3.

27.5.4 Translation of Guest-Physical Addresses Used by Intel Processor Trace

As described in Chapter 35, Intel® Processor Trace (Intel PT) captures information about software execution using dedicated hardware facilities.

Intel PT can be configured so that the trace output is written to memory using physical addresses. For example, when the ToPA (table of physical addresses) output mechanism is used, the `IA32_RTIT_OUTPUT_BASE` MSR contains the physical address of the base of the current ToPA. Each entry in that table contains the physical address of an output region in memory. When an output region becomes full, the ToPA output mechanism directs subsequent trace output to the next output region as indicated in the ToPA.

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the logical processor treats the addresses used by Intel PT (the output addresses as well as those used to discover the output addresses) as guest-physical addresses, translating to physical addresses using EPT before trace output is written to memory.

Translating these addresses through EPT implies that the trace-output mechanism may cause EPT violations and VM exits; details are provided in Section 27.5.4.1. Section 27.5.4.2 describes a mechanism that ensures that these VM exits do not cause loss of trace data.

27.5.4.1 Guest-Physical Address Translation for Intel PT: Details

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses and translated using EPT. These addresses include the addresses of the output regions as well as the addresses of the ToPA entries that contain the output-region addresses.

Translation of accesses by the trace-output process may result in EPT violations or EPT misconfigurations (Section 30.3.3), resulting in VM exits. EPT violations resulting for the trace-output process always cause VM exits and are never converted to virtualization exceptions (Section 27.5.7.1).

If no EPT violation or EPT misconfiguration occurs and if page-modification logging (Section 30.3.6) is enabled, the address of an output region may be added to the page-modification log. If the log is full, a page-modification log-full event occurs, resulting in a VM exit.

If the “virtualize APIC accesses” VM-execution control is 1, a guest-physical address used by the trace-output process may be translated to an address on the APIC-access page. In this case, the access by the trace-output process causes an APIC-access VM exit as discussed in Section 31.4.6.1.

27.5.4.2 Trace-Address Pre-Translation (TAPT)

Because it buffers trace data produced by Intel PT before it is written to memory, the processor ensures that buffered data is not lost when a VM exit disables Intel PT. Specifically, the processor ensures that there is sufficient space left in the current output page for the buffered data. If this were not done, buffered trace data could be lost and the resulting trace corrupted.

To prevent the loss of buffered trace data, the processor uses a mechanism called **trace-address pre-translation (TAPT)**. With TAPT, the processor translates using EPT the guest-physical address of the current output region before that address would be used to write buffered trace data to memory.

Because of TAPT, no translation (and thus no EPT violation) occurs at the time output is written to memory; the writes to memory use translations that were cached as part of TAPT. (The details given in Section 27.5.4.1 apply to TAPT.) TAPT ensures that, if a write to the output region would cause an EPT violation, the resulting VM exit is delivered at the time of TAPT, before the region would be used. This allows software to resolve the EPT violation at that time and ensures that, when it is necessary to write buffered trace data to memory, that data will not be lost due to an EPT violation.

TAPT (and resulting VM exits) may occur at any of the following times:

- When software in VMX non-root operation enables tracing by loading the IA32_RTIT_CTL MSR to set the TraceEn bit, using the WRMSR instruction or the XRSTORS instruction.
Any VM exit resulting from TAPT in this case is trap-like: the WRMSR or XRSTORS completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).
- At an instruction boundary when one output region becomes full and Intel PT transitions to the next output region.
VM exits resulting from TAPT in this case take priority over any pending debug exceptions. Such a VM exit will save information about such exceptions in the guest-state area of the VMCS.
- As part of a VM entry that enables Intel PT. See Section 28.5 for details.

TAPT may translate not only the guest-physical address of the current output region but those of subsequent output regions as well. (Doing so may provide better protection of trace data.) This implies that any VM exits resulting from TAPT may result from the translation of output-region addresses other than that of the current output region.

27.5.5 APIC Virtualization

APIC virtualization is a collection of features that can be used to support the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC). When APIC virtualization is enabled, the processor emulates many accesses to the APIC, tracks the state of the virtual APIC, and delivers virtual interrupts — all in VMX non-root operation without a VM exit.

Details of the APIC virtualization are given in Chapter 31.

27.5.6 VM Functions

A **VM function** is an operation provided by the processor that can be invoked from VMX non-root operation without a VM exit. VM functions are enabled and configured by the settings of different fields in the VMCS. Software in VMX non-root operation invokes a VM function with the **VMFUNC** instruction; the value of EAX selects the specific VM function being invoked.

Section 27.5.6.1 explains how VM functions are enabled. Section 27.5.6.2 specifies the behavior of the VMFUNC instruction. Section 27.5.6.3 describes a specific VM function called **EPTP switching**.

27.5.6.1 Enabling VM Functions

Software enables VM functions generally by setting the “enable VM functions” VM-execution control. A specific VM function is enabled by setting the corresponding VM-function control.

Suppose, for example, that software wants to enable EPTP switching (VM function 0; see Section 26.6.14). To do so, it must set the “activate secondary controls” VM-execution control (bit 31 of the primary processor-based VM-execution controls), the “enable VM functions” VM-execution control (bit 13 of the secondary processor-based VM-execution controls) and the “EPTP switching” VM-function control (bit 0 of the VM-function controls).

27.5.6.2 General Operation of the VMFUNC Instruction

The VMFUNC instruction causes an invalid-opcode exception (#UD) if the “enable VM functions” VM-execution controls is 0¹ or the value of EAX is greater than 63 (only VM functions 0–63 can be enable). Otherwise, the instruc-

tion causes a VM exit if the bit at position EAX is 0 in the VM-function controls (the selected VM function is not enabled). If such a VM exit occurs, the basic exit reason used is 59 (3BH), indicating “VMFUNC”, and the length of the VMFUNC instruction is saved into the VM-exit instruction-length field. If the instruction causes neither an invalid-opcode exception nor a VM exit due to a disabled VM function, it performs the functionality of the VM function specified by the value in EAX.

Individual VM functions may perform additional fault checking (e.g., one might cause a general-protection exception if $CPL > 0$). In addition, specific VM functions may include checks that might result in a VM exit. If such a VM exit occurs, VM-exit information is saved as described in the previous paragraph. The specification of a VM function may indicate that additional VM-exit information is provided.

The specific behavior of the EPTP-switching VM function (including checks that result in VM exits) is given in Section 27.5.6.3.

27.5.6.3 EPTP Switching

EPTP switching is VM function 0. This VM function allows software in VMX non-root operation to load a new value for the EPT pointer (EPTP), thereby establishing a different EPT paging-structure hierarchy (see Section 30.3 for details of the operation of EPT). Software is limited to selecting from a list of potential EPTP values configured in advance by software in VMX root operation.

Specifically, the value of ECX is used to select an entry from the EPTP list, the 4-KByte structure referenced by the EPTP-list address (see Section 26.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if $ECX \geq 512$). The EPTP value in the selected entry is evaluated to determine whether it is valid for EPTP switching: a value is valid if (1) it is the same as the current EPTP value in bits 5:3 (these bits specify the EPT page-walk length); and (2) it would not cause VM entry to fail (see Section 28.2.1.1). If the value is invalid, a VM exit occurs. Otherwise, the value is stored in the EPTP field of the current VMCS and is used for subsequent accesses using guest-physical addresses. The following pseudocode provides details:

```
IF ECX ≥ 512
    THEN VM exit;
ELSE
    tent_EPTP := 8 bytes from EPTP-list address + 8 * ECX;
    IF tent_EPTP is not a valid EPTP value (would cause VM entry to fail if in EPTP or change EPT page-walk length)
        THEN VM exit;
    ELSE
        write tent_EPTP to the EPTP field in the current VMCS;
        use tent_EPTP as the new EPTP value for address translation;
        IF processor supports the 1-setting of the “EPT-violation #VE” VM-execution control
            THEN
                write ECX[15:0] to EPTP-index field in current VMCS;
                use ECX[15:0] as EPTP index for subsequent EPT-violation virtualization exceptions (see Section 27.5.7.2);
        FI;
    FI;
FI;
```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

If the “Intel PT uses guest physical addresses” VM-execution control is 1 and $IA32\_RTIT\_CTL.TraceEn = 1$, any execution of the EPTP-switching VM function causes a VM exit.¹

As noted in Section 27.5.6.2, an execution of the EPTP-switching VM function that causes a VM exit (as specified above), uses the basic exit reason 59, indicating “VMFUNC”. The length of the VMFUNC instruction is saved into the VM-exit instruction-length field. No additional VM-exit information is provided.

1. “Enable VM functions” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable VM functions” VM-execution control were 0. See Section 26.6.2.

1. Such a VM exit ensures the proper recording of trace data that might otherwise be lost during the change of EPT paging-structure hierarchy. Software handling the VM exit can change emulate the VM function and then resume the guest.

An execution of VMFUNC loads EPTP from the EPTP list (and thus does not cause a fault or VM exit) is called an **EPTP-switching VMFUNC**. After an EPTP-switching VMFUNC, control passes to the next instruction. The logical processor starts creating and using guest-physical and combined mappings associated with the new value of bits 51:12 of EPTP; the combined mappings created and used are associated with the current VPID and PCID (these are not changed by VMFUNC).¹ If the “enable VPID” VM-execution control is 0, an EPTP-switching VMFUNC invalidates combined mappings associated with VPID 0000H (for all PCIDs and for all EPTRTA values, where EPTRTA is the value of bits 51:12 of EPTP).

Because an EPTP-switching VMFUNC may change the translation of guest-physical addresses, it may affect use of the guest-physical address in CR3. The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration due to the translation of that guest-physical address through the new EPT paging structures. The following items provide details that apply if CR0.PG = 1:

- If 32-bit paging or 4-level paging² is in use (either CR4.PAE = 0 or IA32_EFER.LMA = 1), the next memory access with a linear address uses the translation of the guest-physical address in CR3 through the new EPT paging structures. As a result, this access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.
- If PAE paging is in use (CR4.PAE = 1 and IA32_EFER.LMA = 0), an EPTP-switching VMFUNC **does not** load the four page-directory-pointer-table entries (PDPTEs) from the guest-physical address in CR3. The logical processor continues to use the four guest-physical addresses already present in the PDPTEs. The guest-physical address in CR3 is not translated through the new EPT paging structures (until some operation that would load the PDPTEs).

The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during the translation of a guest-physical address in any of the PDPTEs. A subsequent memory access with a linear address uses the translation of the guest-physical address in the appropriate PDPTE through the new EPT paging structures. As a result, such an access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.

If an EPTP-switching VMFUNC establishes an EPTP value that enables accessed and dirty flags for EPT (by setting bit 6), subsequent memory accesses may fail to set those flags as specified if there has been no appropriate execution of INVEPT since the last use of an EPTP value that does not enable accessed and dirty flags for EPT (because bit 6 is clear) and that is identical to the new value on bits 51:12.

If the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control, an EPTP-switching VMFUNC loads the value in ECX[15:0] into to EPTP-index field in current VMCS. Subsequent EPT-violation virtualization exceptions will save this value into the virtualization-exception information area (see Section 27.5.7.2).

27.5.7 Virtualization Exceptions

A **virtualization exception** is a new processor exception. It uses vector 20 and is abbreviated #VE.

A virtualization exception can occur only in VMX non-root operation. Virtualization exceptions occur only with certain settings of certain VM-execution controls. Generally, these settings imply that certain conditions that would normally cause VM exits instead cause virtualization exceptions

In particular, the 1-setting of the “EPT-violation #VE” VM-execution control causes some EPT violations to generate virtualization exceptions instead of VM exits. Section 27.5.7.1 provides the details of how the processor determines whether an EPT violation causes a virtualization exception or a VM exit.

When the processor encounters a virtualization exception, it saves information about the exception to the virtualization-exception information area; see Section 27.5.7.2.

After saving virtualization-exception information, the processor delivers a virtualization exception as it would any other exception; see Section 27.5.7.3 for details.

1. If the “enable VPID” VM-execution control is 0, the current VPID is 0000H; if CR4.PCIDE = 0, the current PCID is 000H.

2. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

27.5.7.1 Convertible EPT Violations

If the “EPT-violation #VE” VM-execution control is 0 (e.g., on processors that do not support this feature), EPT violations always cause VM exits. If instead the control is 1, certain EPT violations may be converted to cause virtualization exceptions instead; such EPT violations are **convertible**.

The values of certain EPT paging-structure entries determine which EPT violations are convertible. Specifically, bit 63 of certain EPT paging-structure entries may be defined to mean **suppress #VE**:

- If bits 2:0 of an EPT paging-structure entry are all 0, the entry is not **present**.¹ If the processor encounters such an entry while translating a guest-physical address, it causes an EPT violation. The EPT violation is convertible if and only if bit 63 of the entry is 0.
- If an EPT paging-structure entry is present, the following cases apply:
 - If the value of the EPT paging-structure entry is not supported, the entry is **misconfigured**. If the processor encounters such an entry while translating a guest-physical address, it causes an EPT misconfiguration (not an EPT violation). EPT misconfigurations always cause VM exits.
 - If the value of the EPT paging-structure entry is supported, the following cases apply:
 - If bit 7 of the entry is 1, or if the entry is an EPT PTE, the entry maps a page. If the processor uses such an entry to translate a guest-physical address, and if an access to that address causes an EPT violation, the EPT violation is convertible if and only if bit 63 of the entry is 0.
 - If bit 7 of the entry is 0 and the entry is not an EPT PTE, the entry references another EPT paging structure. The processor does not use the value of bit 63 of the entry to determine whether any subsequent EPT violation is convertible.

If an access to a guest-physical address causes an EPT violation, bit 63 of exactly one of the EPT paging-structure entries used to translate that address is used to determine whether the EPT violation is convertible: either a entry that is not present (if the guest-physical address does not translate to a physical address) or an entry that maps a page (if it does).

A convertible EPT violation instead causes a virtualization exception if the following all hold:

- CR0.PE = 1;
- the logical processor is not in the process of delivering an event through the IDT;
- the EPT violation did not cause a shadow stack to become prematurely busy (see Section 27.4.3);
- the EPT violation does not result from the output process of Intel Processor Trace (Section 27.5.4); and
- the 32 bits at offset 4 in the virtualization-exception information area are all 0.

Delivery of virtualization exceptions writes the value FFFFFFFFH to offset 4 in the virtualization-exception information area (see Section 27.5.7.2). Thus, once a virtualization exception occurs, another can occur only if software clears this field.

27.5.7.2 Virtualization-Exception Information

Virtualization exceptions save data into the virtualization-exception information area (see Section 26.6.19). Table 27-23 enumerates the data saved and the format of the area.

Table 27-23. Format of the Virtualization-Exception Information Area

Byte Offset	Contents
0	The 32-bit value that would have been saved into the VMCS as an exit reason had a VM exit occurred instead of the virtualization exception. For EPT violations, this value is 48 (00000030H)
4	FFFFFFFFH

1. If the “mode-based execute control for EPT” VM-execution control is 1, an EPT paging-structure entry is present if any of bits 2:0 or bit 10 is 1.

Table 27-23. Format of the Virtualization-Exception Information Area (Contd.)

Byte Offset	Contents
8	The 64-bit value that would have been saved into the VMCS as an exit qualification had a VM exit occurred instead of the virtualization exception
16	The 64-bit value that would have been saved into the VMCS as a guest-linear address had a VM exit occurred instead of the virtualization exception
24	The 64-bit value that would have been saved into the VMCS as a guest-physical address had a VM exit occurred instead of the virtualization exception
32	The current 16-bit value of the EPTP index VM-execution control (see Section 26.6.19 and Section 27.5.6.3)

A VMM may allow guest software to access the virtualization-exception information area. If it does, the guest software may modify that memory (e.g., to clear the 32-bit value at offset 4; see Section 27.5.7.1). (This is an exception to the general requirement given in Section 26.11.4.)

27.5.7.3 Delivery of Virtualization Exceptions

After saving virtualization-exception information, the processor treats a virtualization exception as it does other exceptions:

- If bit 20 (#VE) is 1 in the exception bitmap in the VMCS, a virtualization exception causes a VM exit (see below). If the bit is 0, the virtualization exception is delivered using gate descriptor 20 in the IDT.
- Virtualization exceptions produce no error code. Delivery of a virtualization exception pushes no error code on the stack.
- With respect to double faults, virtualization exceptions have the same severity as page faults. If delivery of a virtualization exception encounters a nested fault that is either contributory or a page fault, a double fault (#DF) is generated. See Chapter 7, “Interrupt 8—Double Fault Exception (#DF)” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

It is not possible for a virtualization exception to be encountered while delivering another exception (see Section 27.5.7.1).

If a virtualization exception causes a VM exit directly (because bit 20 is 1 in the exception bitmap), information about the exception is saved normally in the VM-exit interruption information field in the VMCS (see Section 29.2.2). Specifically, the event is reported as a hardware exception with vector 20 and no error code. Bit 12 of the field (NMI unblocking due to IRET) is set normally.

If a virtualization exception causes a VM exit indirectly (because bit 20 is 0 in the exception bitmap and delivery of the exception generates an event that causes a VM exit), information about the exception is saved normally in the IDT-vectoring information field in the VMCS (see Section 29.2.4). Specifically, the event is reported as a hardware exception with vector 20 and no error code.

27.5.8 PASID Translation

The ENQCMD and ENQCMLS instructions each performs a 64-byte enqueue store that includes a 20-bit PASID value in bits 19:0. For ENQCMD, the PASID is normally the value of IA32_PASID[19:0], while for ENQCMLS, the PASID is normally read from memory.

If the “PASID translation” VM-execution control is 1, the PASID value identified in the previous paragraph is treated as a **guest PASID**. PASID translation converts this guest PASID to a 20-bit **host PASID**. After this translation, the enqueue store is performed, using the host PASID in place of the guest PASID.

PASID translation is implemented by two hierarchies of data structures (**PASID-translation hierarchies**) configured by a VMM. Guest PASIDs 00000H to 7FFFFH are translated through the low PASID-translation hierarchy, while guest PASIDs 80000 to FFFFFH are translated through the high PASID-translation hierarchy.

The root of each PASID-translation hierarchy is a 4-KByte **PASID directory**. The low PASID directory is located at the low PASID directory address, and the high PASID directory is located at the high PASID directory address (these physical addresses are VM-execution control fields in the VMCS). A PASID directory comprises 512 8-byte entries, each of which has the following format:

- Bit 0 is the entry's present bit. The entry is used only if this bit is 1.
- Bits 11:1 are reserved and must be 0.
- Bits **MAXPHYADDR-1:12** specify the 4-KByte aligned address of a PASID table (see below), where **MAXPHYADDR** is defined as follows:
 - Usually, **MAXPHYADDR** is the processor's supported physical-address width as enumerated in **CPUID.80000008H:EAX[7:0]** (this width is at most 52).
 - If **IA32_TME_ACTIVATE[0] = 1** (indicating that TME has been configured), **MAXPHYADDR** is reduced by the value of **IA32_TME_ACTIVATE[39:36]** when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.¹
- Bits 63:**MAXPHYADDR** are reserved and must be 0.

A PASID-translation hierarchy also includes up to 512 4-KByte **PASID tables**; each of these is referenced by a PASID directory entry (see above). A PASID table comprises 1024 4-byte entries, each of which has the following format:

- Bits 19:0 are the host PASID specified by the entry.
- Bits 30:20 are reserved and must be 0.
- Bit 31 is the entry's valid bit. The entry is used only if this bit is 1.

When PASID translation is enabled, the guest PASID determined by the instruction (see above) is converted to a host PASID using the following process:

- If bit 19 of guest PASID is clear, the low PASID directory is used; otherwise, the high PASID directory is used.
- Bits 18:10 of the guest PASID select an entry from the PASID directory. A VM exit occurs if the entry's present bit is clear or if any reserved bit is set. Otherwise, bits **MAXPHYADDR-1:0** of the entry (with bit 0 cleared) contain the physical address of a PASID table.
- Bits 9:0 of the guest PASID select an entry from the PASID table. A VM exit occurs if the entry's valid bit is clear or if any reserved bit is set. Otherwise, bits 19:0 of the entry are the host PASID.

If PASID translation results in a VM exit (due to a present or valid bit being clear, or a reserved bit being set), the instruction does not complete and no enqueue store is performed.

27.6 UNRESTRICTED GUESTS

The first processors to support VMX operation require **CR0.PE** and **CR0.PG** to be 1 in VMX operation (see Section 25.8). This restriction implies that guest software cannot be run in unpaged protected mode or in real-address mode. Later processors support a VM-execution control called "unrestricted guest".² If this control is 1, **CR0.PE** and **CR0.PG** may be 0 in VMX non-root operation. Such processors allow guest software to run in unpaged protected mode or in real-address mode. The following items describe the behavior of such software:

- The **MOV CR0** instructions does not cause a general-protection exception simply because it would set either **CR0.PE** and **CR0.PG** to 0. See Section 27.3 for details.
- A logical processor treats the values of **CR0.PE** and **CR0.PG** in VMX non-root operation just as it does outside VMX operation. Thus, if **CR0.PE = 0**, the processor operates as it does normally in real-address mode (for example, it uses the 16-bit **interrupt table** to deliver interrupts and exceptions). If **CR0.PG = 0**, the processor operates as it does normally when paging is disabled.

1. **IA32_TME_ACTIVATE[39:36]** is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than **IA32_TME_ACTIVATE[35:32]**, which is the number physical-address bits used for key identifiers generally.

2. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "unrestricted guest" VM-execution control were 0. See Section 26.6.2.

- Processor operation is modified by the fact that the processor is in VMX non-root operation and by the settings of the VM-execution controls just as it is in protected mode or when paging is enabled. Instructions, interrupts, and exceptions that cause VM exits in protected mode or when paging is enabled also do so in real-address mode or when paging is disabled. The following examples should be noted:
 - If CR0.PG = 0, page faults do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, invalid-TSS exceptions do not occur and thus cannot cause VM exits.
 - If CR0.PE = 0, the following instructions cause invalid-opcode exceptions and do not cause VM exits: INVEPT, INVVPID, LLDT, LTR, SLDT, STR, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.
- If CR0.PG = 0, each linear address is passed directly to the EPT mechanism for translation to a physical address.¹ The guest memory type passed on to the EPT mechanism is WB (writeback).

1. As noted in Section 28.2.1.1, the “enable EPT” VM-execution control must be 1 if the “unrestricted guest” VM-execution control is 1.

21. Updates to Chapter 28, Volume 3B

Change bars and violet text show changes to Chapter 28 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updated Section 28.2.1, "Checks on VMX Controls" and its subsections 28.2.1.1, 28.3.1.2, and 28.2.1.3 for SEAM.
- Updated Section 28.2.2, "Checks on Host Control Registers, MSRs, and SSP" with field width information for SEAM.
- In Section 28.3.1.1, "Checks on Guest Control Registers, Debug Registers, and MSRs", provided details for field width of the CR3 field in the list of checks performed on processors that support Intel 64 architecture.
- In Section 28.3.1.5, "Checks on Guest Non-Register State", provided additional information on field width for the VMCS link pointer in the list of checks performed on fields in the guest-state area corresponding to non-register state.
- In Section 28.7.2, "Activity State", updated reference to Intel® Trusted Execution Technology Measured Launched Environment Programming Guide.

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is clear and VMRESUME can be used only with a VMCS whose the launch state is launched. VMLAUNCH should be used for the first VM entry after VMCLEAR; VMRESUME should be used for subsequent VM entries with the same VMCS.

Each VM entry performs the following steps in the order indicated:

1. Basic checks are performed to ensure that VM entry can commence (Section 28.1).
2. The control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation and that the VMCS is correctly configured to support the next VM exit (Section 28.2).
3. The following are performed in parallel or in any order (Section 28.3):
 - The guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures.
 - Processor state is loaded from the guest-state area and based on controls in the VMCS.
 - Address-range monitoring is cleared.
4. MSRs are loaded from the VM-entry MSR-load area (Section 28.4).
5. If VMLAUNCH is being executed, the launch state of the VMCS is set to “launched.”
6. If the “Intel PT uses guest physical addresses” VM-execution control is 1, trace-address pre-translation (TAPT) may occur (see Section 27.5.4 and Section 28.5).
7. An event may be injected in the guest context (Section 28.6).

Steps 1–4 above perform checks that may cause VM entry to fail. Such failures occur in one of the following three ways:

- Some of the checks in Section 28.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 28.1 and all the checks in Section 28.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF¹ (if there is a current VMCS) or RFLAGS.CF (if there is no current VMCS). If there is a current VMCS, an error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 31 for the error numbers.
- The checks in Section 28.3 and Section 28.4 cause processor state to be loaded from the host-state area of the VMCS (as would be done on a VM exit). Information about the failure is stored in the VM-exit information fields. See Section 28.8 for details.

EFLAGS.TF = 1 causes a VM-entry instruction to generate a single-step debug exception only if failure of one of the checks in Section 28.1 and Section 28.2 causes control to pass to the following instruction. A VM-entry does not generate a single-step debug exception in any of the following cases: (1) the instruction generates a fault; (2) failure of one of the checks in Section 28.3 or in loading MSRs causes processor state to be loaded from the host-state area of the VMCS; or (3) the instruction passes all checks in Section 28.1, Section 28.2, and Section 28.3 and there is no failure in loading MSRs.

Section 33.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, code running in SMM returns using VM entries instead of the RSM instruction. A VM entry **returns from SMM** if it is executed in SMM and the “entry to SMM” VM-entry control is 0. VM entries that return from SMM differ from ordinary VM entries in ways that are detailed in Section 33.15.4.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

28.1 BASIC VM-ENTRY CHECKS

Before a VM entry commences, the current state of the logical processor is checked in the following order:

1. If the logical processor is in virtual-8086 mode or compatibility mode, an invalid-opcode exception is generated.
2. If the current privilege level (CPL) is not zero, a general-protection exception is generated.
3. If there is no current VMCS, RFLAGS.CF is set to 1 and control passes to the next instruction.
4. If there is a current VMCS but the current VMCS is a shadow VMCS (see Section 26.10), RFLAGS.CF is set to 1 and control passes to the next instruction.
5. If there is a current VMCS that is not a shadow VMCS, the following conditions are evaluated in order; any of these cause VM entry to fail:
 - a. If there is MOV-SS blocking (see Table 26-3).
 - b. If the VM entry is invoked by VMLAUNCH and the VMCS launch state is not clear.
 - c. If the VM entry is invoked by VMRESUME and the VMCS launch state is not launched.

If any of these checks fail, RFLAGS.ZF is set to 1 and control passes to the next instruction. An error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 31 for the error numbers.

28.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 28.1 do not cause VM entry to fail, the control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation, that the VMCS is correctly configured to support the next VM exit, and that, after the next VM exit, the processor's state is consistent with the Intel 64 and IA-32 architectures.

VM entry fails if any of these checks fail. When such failures occur, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with an error number that indicates whether the failure was due to the controls or the host-state area (see Chapter 31).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, host state) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same VMCS. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The checks on the controls and the host-state area are presented in Section 28.2.1 through Section 28.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

28.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

Some of these fields contain physical addresses. These fields are checked against the allowed address width:

- If `IA32_VMX_BASIC[48]` is read as 1 (implying that the processor does not support Intel 64 architecture; see Section A.1, "Basic VMX Information"), these addresses must not set any bits in the range 63:32.
- Otherwise, the widths of physical addresses are limited by `MAXPHYADDR`, a value derived from the value enumerated in `CPUID.80000008H:EAX[7:0]` (at most 52). If `TME_ACTIVATE[0] = 1` (indicating that TME has been configured), `MAXPHYADDR` is reduced by the value of `IA32_TME_ACTIVATE[39:36]` when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.¹ Physical address should not set bits in the range 63:MAXPHYADDR.

28.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:¹

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).
- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the “activate secondary controls” primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).
If the “activate secondary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.
- If the “activate tertiary controls” primary processor-based VM-execution control is 1, reserved bits in the tertiary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.4).
If the “activate tertiary controls” primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the tertiary processor-based VM-execution controls. The logical processor operates as if all the tertiary processor-based VM-execution controls were 0.
- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- Bits 11:0 of the following fields must be 0, and the fields are also subject to the checks on physical-address width described above in Section 28.2.1:
 - If the “use I/O bitmaps” VM-execution control is 1, each I/O-bitmap address.
 - If the “use MSR bitmaps” VM-execution control is 1, the MSR-bitmap address.
 - If the “use TPR shadow” VM-execution control is 1, the virtual-APIC address.
If the checks on that address are satisfied, bytes 3:1 of VTPR (see Section 31.1.1) may be cleared (behavior may be implementation-specific).
The clearing of these bytes may occur even if the VM entry fails. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it causes processor state to be loaded from the host-state area of the VMCS.
 - If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address.
 - If the “enable PML” VM-execution control is 1, the PML address.
 - If the “sub-page write permissions for EPT” VM-execution control is 1, the SPPTP VM-execution control field (see Table 26-11 in Section 26.6.21).
 - If the “enable VM functions” processor-based VM-execution control is 1 and the “EPTP switching” VM-function control is 1, the EPTP-list address.
 - If the “VMCS shadowing” VM-execution control is 1, the VMREAD-bitmap and VMWRITE-bitmap addresses.
 - If the “EPT-violation #VE” VM-execution control is 1, the virtualization-exception information address.
 - If the “PASID translation” VM-execution control is 1, the low PASID directory address and the high PASID directory address.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

1. If the “activate secondary controls” primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0. Similarly, if the “activate tertiary controls” primary processor-based VM-execution control is 0, VM entry operates as if each tertiary processor-based VM-execution control were 0. See Section 26.6.2.

- If the “use TPR shadow” VM-execution control is 1 and the “virtual-interrupt delivery” VM-execution control is 0, bits 31:4 of the TPR threshold VM-execution control field must be 0.
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” and “virtual-interrupt delivery” VM-execution controls are both 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 of VTPR (see Section 31.1.1).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “use TPR shadow” VM-execution control is 0, the following VM-execution controls must also be 0: “virtualize x2APIC mode”, “APIC-register virtualization”, “virtual-interrupt delivery”, and “IPI virtualization”.
- If the “virtualize x2APIC mode” VM-execution control is 1, the “virtualize APIC accesses” VM-execution control must be 0.
- If the “virtual-interrupt delivery” VM-execution control is 1, the “external-interrupt exiting” VM-execution control must be 1.
- If the “process posted interrupts” VM-execution control is 1, the following must be true:
 - The “virtual-interrupt delivery” VM-execution control is 1.
 - The “acknowledge interrupt on exit” VM-exit control is 1.
 - The posted-interrupt notification vector has a value in the range 0–255 (bits 15:8 are all 0).
 - Bits 5:0 of the posted-interrupt descriptor address are all 0.
 - [The posted-interrupt descriptor address satisfies the checks on physical-address width described above in Section 28.2.1.](#)
- If the “IPI virtualization” VM-execution control is 1, the following must be true:
 - Bits 2:0 of the PID-pointer table address are all 0.
 - The PID-pointer table address [satisfies the checks on physical-address width described above in Section 28.2.1.](#)
 - [The address of the last entry in the PID-pointer table satisfies the checks on physical-address width described above.](#) (This address is the PID-pointer table address plus 8 times the last PID-pointer index.)
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.
- If the “enable EPT” VM-execution control is 1, the EPTP VM-execution control field (see Table 26-9 in Section 26.6.11) must satisfy the following checks:
 - The EPT memory type (bits 2:0) must be a value supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).
 - Bits 5:3 must contain a value 1 less than an EPT page-walk length supported by the processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Section 30.3.2 and Appendix A.10).
 - Bit 6 (enable bit for accessed and dirty flags for EPT) must be 0 if bit 21 of the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10) is read as 0, indicating that the processor does not support accessed and dirty flags for EPT.
 - Reserved bits 11:7 must all be 0.
 - [The 4-KByte-aligned address defined by bits 63:12 must satisfy the checks on physical-address width described above in Section 28.2.1.](#)
- The “enable EPT” VM-execution control must be 1 if any of the following VM-execution controls is 1: “enable PML”, “unrestricted guest”, “mode-based execute control for EPT”, “sub-page write permissions for EPT”, “Intel PT uses guest physical addresses”, “enable HLAT”, “EPT paging-write control”, or “guest-paging verification”.
- [The following checks apply if the logical processor is in SEAM at the time of VM entry:](#)
 - [The “enable EPT” VM-execution control must be 1.](#)

- Bits 11:0 and bits 63:M must be zero in the SEAM shared EPT pointer, where M is the value enumerated by CPUID.80000008H:EAX[7:0].
- If the “enable VM functions” processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 26.6.14):
 - If “EPTP switching” VM-function control is 1, the “enable EPT” VM-execution control must also be 1.
If the “enable VM functions” processor-based VM-execution control is 0, no checks are performed on the VM-function controls.
- If the logical processor is operating with Intel PT enabled (if IA32_RTIT_CTL.TraceEn = 1) at the time of VM entry, the “load IA32_RTIT_CTL” VM-entry control must be 0.
- If the “Intel PT uses guest physical addresses” VM-execution control is 1, the “load IA32_RTIT_CTL” VM-entry control and the “clear IA32_RTIT_CTL” VM-exit control must both be 1.
- If the “use TSC scaling” VM-execution control is 1, the TSC-multiplier must not be zero.
- If the “enable HLAT” VM-execution control is 1, the following bits in the HLATP VM-execution control field (see Table 26-12 in Section 26.6.22) must be zero: bits 2:0, bits 11:5. In addition, the 4-KByte-aligned address defined by bits 63:12 must satisfy the checks on physical-address width described above in Section 28.2.1.

28.2.1.2 VM-Exit Control Fields

VM entries perform the following checks on the VM-exit control fields.

- Reserved bits in the primary VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.4.1).
- If the “activate secondary controls” primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).
- If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.
- If the “activate VMX-preemption timer” VM-execution control is 0, the “save VMX-preemption timer value” VM-exit control must also be 0.
- The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-store address must be 0. The address must satisfy the checks on physical-address width described above in Section 28.2.1.
 - The address of the last byte in the VM-exit MSR-store area must satisfy the checks on physical-address width described above. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)
- The following checks are performed for the VM-exit MSR-load address if the VM-exit MSR-load count field is non-zero:
 - The lower 4 bits of the VM-exit MSR-load address must be 0. The address must satisfy the checks on physical-address width described above.
 - The address of the last byte in the VM-exit MSR-load area must satisfy the checks on physical-address width described above. The address of this last byte is VM-exit MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

28.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.5).

- Fields relevant to VM-entry event injection must be set properly. These fields are the VM-entry interruption-information field (see Table 26-18 in Section 26.8.3, “VM-Entry Controls for Event Injection”), the VM-entry exception error code, and the VM-entry instruction length. If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the following must hold:
 - The field’s interruption type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that do not support the 1-setting of the “monitor trap flag” VM-execution control.
 - The field’s vector (bits 7:0) is consistent with the interruption type:
 - If the interruption type is non-maskable interrupt (NMI), the vector is 2.
 - If the interruption type is hardware exception, the vector is at most 31.
 - If the interruption type is other event, the vector is 0 (pending MTF VM exit).
 - The field’s deliver-error-code bit (bit 11) is 1 if each of the following holds: (1) the interruption type is hardware exception; (2) either (a) the “unrestricted guest” VM-execution control is 0; or (b) bit 0 (corresponding to CR0.PE) is set in the CR0 field in the guest-state area; (3) IA32_VMX_BASIC[56] is read as 0 (see Appendix A.1); and (4) the vector indicates one of the following exceptions: #DF (vector 8), #TS (10), #NP (11), #SS (12), #GP (13), #PF (14), or #AC (17).
 - The field’s deliver-error-code bit is 0 if any of the following holds: (1) the interruption type is not hardware exception; (2) both (a) the “unrestricted guest” VM-execution control is 1; and (b) bit 0 is clear in the CR0 field in the guest-state area; or (3) IA32_VMX_BASIC[56] is read as 0 and the vector is in one of the following ranges: 0–7, 9, 15, 16, or 18–31.
 - Reserved bits in the field (30:12) are 0.
 - If the deliver-error-code bit (bit 11) is 1, bits 31:16 of the VM-entry exception error-code field are 0.
 - If the interruption type is software interrupt, software exception, or privileged software exception, the VM-entry instruction-length field is in the range 0–15. A VM-entry instruction length of 0 is allowed only if IA32_VMX_MISC[30] is read as 1; see Appendix A.6.
- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
 - The lower 4 bits of the VM-entry MSR-load address must be 0. The address must [satisfy the checks on physical-address width described above in Section 28.2.1](#).
 - [The address of the last byte in the VM-entry MSR-load area must satisfy the checks on physical-address width described above](#). The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1, “Basic VMX Information”.
- If the processor is not in SMM, the “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls must be 0.
- The “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls cannot both be 1.

28.2.2 Checks on Host Control Registers, MSRs, and SSP

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 25.8).¹
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 25.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- On processors that support Intel 64 architecture, the CR3 field must be such that bits reserved in CR3 are 0.² [In particular, bits 63:32 are checked for allowed address width](#):

1. The bits corresponding to CR0.NW (bit 29) and CR0.CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 29.5.1.

- If IA32_VMX_BASIC[48] is read as 1 (implying that the processor does not support Intel 64 architecture; see Appendix A.1, “Basic VMX Information”), the field must not set any of bits 63:32.
- Otherwise, the field’s width is limited by MAXPHYADDR, which is derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. The field should not set bits in the range 63:MAXPHYADDR.
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 21-3).
- If the “load IA32_PAT” VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-exit control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the “host address-space size” VM-exit control.
- If the “load CET state” VM-exit control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) in the field cannot both be set.
- If the “load CET state” VM-exit control is 1, bits 1:0 must be 0 in the SSP field.
- If the “load PKRS” VM-exit control is 1, bits 63:32 must be 0 in the IA32_PKRS field.

28.2.3 Checks on Host Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area that correspond to segment and descriptor-table registers:

- In the selector field for each of CS, SS, DS, ES, FS, GS, and TR, the RPL (bits 1:0) and the TI flag (bit 2) must be 0.
- The selector fields for CS and TR cannot be 0000H.
- The selector field for SS cannot be 0000H if the “host address-space size” VM-exit control is 0.
- On processors that support Intel 64 architecture, the base-address fields for FS, GS, GDTR, IDTR, and TR must contain canonical addresses.

28.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if IA32_EFER.LMA = 0) at the time of VM entry, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.
 - The “host address-space size” VM-exit control is 0.
- If the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1) at the time of VM entry, the “host address-space size” VM-exit control must be 1.
- If the “host address-space size” VM-exit control is 0, the following must hold:
 - The “IA-32e mode guest” VM-entry control is 0.

-
2. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

- Bit 17 of the CR4 field (corresponding to CR4.PCIDE) is 0.
- Bits 63:32 in the RIP field are 0.
- If the “load CET state” VM-exit control is 1, bits 63:32 in the IA32_S_CET field and in the SSP field are 0.
- If the “host address-space size” VM-exit control is 1, the following must hold:
 - Bit 5 of the CR4 field (corresponding to CR4.PAE) is 1.
 - The RIP field contains a canonical address.
 - If the “load CET state” VM-exit control is 1, the IA32_S_CET field and the SSP field contain canonical addresses.
- If the “load CET state” VM-exit control is 1, the IA32_INTERRUPT_SSP_TABLE_ADDR field contains a canonical address.

On processors that do not support Intel 64 architecture, checks are performed to ensure that the “IA-32e mode guest” VM-entry control and the “host address-space size” VM-exit control are both 0.

28.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 28.2), the following operations take place concurrently: (1) the guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures; (2) processor state is loaded from the guest-state area or as specified by the VM-entry control fields; and (3) address-range monitoring is cleared.

Because the checking and the loading occur concurrently, a failure may be discovered only after some state has been loaded. For this reason, the logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 28.8.

28.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area. These checks may be performed in any order. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The following subsections reference fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

28.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 25.8). The following are exceptions:
 - Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the “unrestricted guest” VM-execution control is 1.¹
 - Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 28.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.²

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 26.6.2.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 25.8).
- If bit 23 in the CR4 field (corresponding to CET) is 1, bit 16 in the CR0 field (WP) must also be 1.
- If the “load debug controls” VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
 - If the “IA-32e mode guest” VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.¹
 - If the “IA-32e mode guest” VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must be 0.
 - The CR3 field must be such that bits reserved in CR3 are 0.² In particular, the field’s width is limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If the “enable EPT” VM-execution control is 0 (implying that the logical processor is not in SEAM) and IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36]. The CR3 field should not set any of bits 63:MAXPHYADDR.
 - If the “load debug controls” VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
 - The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
 - If the “load CET state” VM-entry control is 1, the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field must contain canonical addresses.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 21-3).
- If the “load IA32_PAT” VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the “load IA32_EFER” VM-entry control is 1, the following checks are performed on the field for the IA32_EFER MSR:
 - Bits reserved in the IA32_EFER MSR must be 0.
 - Bit 10 (corresponding to IA32_EFER.LMA) must equal the value of the “IA-32e mode guest” VM-entry control. It must also be identical to bit 8 (LME) if bit 31 in the CR0 field (corresponding to CR0.PG) is 1.³
- If the “load IA32_BNDCFGS” VM-entry control is 1, the following checks are performed on the field for the IA32_BNDCFGS MSR:
 - Bits reserved in the IA32_BNDCFGS MSR must be 0.
 - The linear address in bits 63:12 must be canonical.
- If the “load IA32_RTIT_CTL” VM-entry control is 1, bits reserved in the IA32_RTIT_CTL MSR must be 0 in the field for that register (see Table 35-6).
- If the “load CET state” VM-entry control is 1, the IA32_S_CET field must not set any bits reserved in the IA32_S_CET MSR, and bit 10 (corresponding to SUPPRESS) and bit 11 (TRACKER) of the field cannot both be set.
- If the “load guest IA32_LBR_CTL” VM-entry control is 1, bits reserved in the IA32_LBR_CTL MSR must be 0 in the field for that register.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

2. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

3. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- If the “load PKRS” VM-entry control is 1, bits 63:32 must be 0 in the IA32_PKRS field.
- If the “load UINV” VM-entry control is 1, bits 15:8 must be 0 in the guest UINV field.

28.3.1.2 Checks on Guest Segment Registers

This section specifies the checks on the fields for CS, SS, DS, ES, FS, GS, TR, and LDTR. The following terms are used in defining these checks:

- The guest will be **virtual-8086** if the VM flag (bit 17) is 1 in the RFLAGS field in the guest-state area.
- The guest will be **IA-32e mode** if the “IA-32e mode guest” VM-entry control is 1. (This is possible only on processors that support Intel 64 architecture.)
- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

- Selector fields.
 - TR. The TI flag (bit 2) must be 0.
 - LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
 - SS. If the guest will not be virtual-8086 and the “unrestricted guest” VM-execution control is 0, the RPL (bits 1:0) must equal the RPL of the selector field for CS.¹
- Base-address fields.
 - CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the address must be the selector field shifted left 4 bits (multiplied by 16).
 - The following checks are performed on processors that support Intel 64 architecture:
 - TR, FS, GS. The address must be canonical.
 - LDTR. If LDTR is usable, the address must be canonical.
 - CS. Bits 63:32 of the address must be zero.
 - SS, DS, ES. If the register is usable, bits 63:32 of the address must be zero.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
 - CS, SS, DS, ES, FS, GS.
 - If the guest will be virtual-8086, the field must be 000000F3H. This implies the following:
 - Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
 - Bit 4 (S) must be 1.
 - Bits 6:5 (DPL) must be 3.
 - Bit 7 (P) must be 1.
 - Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.
 - If the guest will not be virtual-8086, the different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - CS. The values allowed depend on the setting of the “unrestricted guest” VM-execution control:
 - If the control is 0, the Type must be 9, 11, 13, or 15 (accessed code segment).

1. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 26.6.2.

- If the control is 1, the Type must be either 3 (read/write accessed expand-up data segment) or one of 9, 11, 13, and 15 (accessed code segment).
- SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
- DS, ES, FS, GS. The following checks apply if the register is usable:
 - Bit 0 of the Type must be 1 (accessed).
 - If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
- Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
- Bits 6:5 (DPL).
 - CS.
 - If the Type is 3 (read/write accessed expand-up data segment), the DPL must be 0. The Type can be 3 only if the “unrestricted guest” VM-execution control is 1.
 - If the Type is 9 or 11 (non-conforming code segment), the DPL must equal the DPL in the access-rights field for SS.
 - If the Type is 13 or 15 (conforming code segment), the DPL cannot be greater than the DPL in the access-rights field for SS.
 - SS.
 - If the “unrestricted guest” VM-execution control is 0, the DPL must equal the RPL from the selector field.
 - The DPL must be 0 either if the Type in the access-rights field for CS is 3 (read/write accessed expand-up data segment) or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.¹
 - DS, ES, FS, GS. The DPL cannot be less than the RPL in the selector field if (1) the “unrestricted guest” VM-execution control is 0; (2) the register is usable; and (3) the Type in the access-rights field is in the range 0 – 11 (data segment or non-conforming code segment).
- Bit 7 (P). If the register is CS or if the register is usable, P must be 1.
- Bits 11:8 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- Bit 14 (D/B). For CS, D/B must be 0 if the guest will be IA-32e mode and the L bit (bit 13) in the access-rights field is 1.
- Bit 15 (G). The following checks apply if the register is CS or if the register is usable:
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
- Bits 31:17 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- TR. The different sub-fields are considered separately:
 - Bits 3:0 (Type).
 - If the guest will not be IA-32e mode, the Type must be 3 (16-bit busy TSS) or 11 (32-bit busy TSS).
 - If the guest will be IA-32e mode, the Type must be 11 (64-bit busy TSS).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).

1. The following apply if either the “unrestricted guest” VM-execution control or bit 31 of the primary processor-based VM-execution controls is 0: (1) bit 0 in the CR0 field must be 1 if the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation; and (2) the Type in the access-rights field for CS cannot be 3.

- If any bit in the limit field in the range 11:0 is 0, G must be 0.
- If any bit in the limit field in the range 31:20 is 1, G must be 1.
- Bit 16 (Unusable). The unusable bit must be 0.
- Bits 31:17 (reserved). These bits must all be 0.
- LDTR. The following checks on the different sub-fields apply only if LDTR is usable:
 - Bits 3:0 (Type). The Type must be 2 (LDT).
 - Bit 4 (S). S must be 0.
 - Bit 7 (P). P must be 1.
 - Bits 11:8 (reserved). These bits must all be 0.
 - Bit 15 (G).
 - If any bit in the limit field in the range 11:0 is 0, G must be 0.
 - If any bit in the limit field in the range 31:20 is 1, G must be 1.
 - Bits 31:17 (reserved). These bits must all be 0.

28.3.1.3 Checks on Guest Descriptor-Table Registers

The following checks are performed on the fields for GDTR and IDTR:

- On processors that support Intel 64 architecture, the base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

28.3.1.4 Checks on Guest RIP, RFLAGS, and SSP

The following checks are performed on fields in the guest-state area corresponding to RIP, RFLAGS, and SSP (shadow-stack pointer):

- RIP. The following checks are performed on processors that support Intel 64 architecture:
 - Bits 63:32 must be 0 if the “IA-32e mode guest” VM-entry control is 0 or if the L bit (bit 13) in the access-rights field for CS is 0.
 - If the processor supports $N < 64$ linear-address bits, bits 63:N must be identical if the “IA-32e mode guest” VM-entry control is 1 and the L bit in the access-rights field for CS is 1.¹ (No check applies if the processor supports 64 linear-address bits.) The guest RIP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.
- RFLAGS.
 - Reserved bits 63:22 (bits 31:22 on processors that do not support Intel 64 architecture), bit 15, bit 5 and bit 3 must be 0 in the field, and reserved bit 1 must be 1.
 - The VM flag (bit 17) must be 0 either if the “IA-32e mode guest” VM-entry control is 1 or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.²
 - The IF flag (RFLAGS[bit 9]) must be 1 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) is external interrupt.
- SSP. The following checks are performed if the “load CET state” VM-entry control is 1
 - Bits 1:0 must be 0.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- If the processor supports the Intel 64 architecture, bits 63:N must be identical, where N is the CPU's maximum linear-address width. (This check does not apply if the processor supports 64 linear-address bits.) The guest SSP value is not required to be canonical; the value of bit N-1 may differ from that of bit N.

28.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

- Activity state.
 - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 26.4.2). Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.
 - The activity-state field must not indicate the HLT state if the DPL (bits 6:5) in the access-rights field for SS is not 0.¹
 - The activity-state field must indicate the active state if the interruptibility-state field indicates blocking by either MOV-SS or by STI (if either bit 0 or bit 1 in that field is 1).
 - If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the interruption to be delivered (as defined by interruption type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following items enumerate the interruptions (as specified in the VM-entry interruption-information field) whose injection is allowed for the different activity states:
 - Active. Any interruption is allowed.
 - HLT. The only events allowed are the following:
 - Those with interruption type external interrupt or non-maskable interrupt (NMI).
 - Those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
 - Those with interruption type other event and vector 0 (pending MTF VM exit).

See Table 26-18 in Section 26.8.3 for details regarding the format of the VM-entry interruption-information field.
 - Shutdown. Only NMIs and machine-check exceptions are allowed.
 - Wait-for-SIPI. No interruptions are allowed.
- The activity-state field must not indicate the wait-for-SIPI state if the "entry to SMM" VM-entry control is 1.
- Interruptibility state.
 - The reserved bits (bits 31:5) must be 0.
 - The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
 - Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
 - Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 0, indicating external interrupt, or value 2, indicating non-maskable interrupt (NMI).
 - Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
 - Bit 2 (blocking by SMI) must be 1 if the "entry to SMM" VM-entry control is 1.
 - Bit 3 (blocking by NMI) must be 0 if the "virtual NMIs" VM-execution control is 1, the valid bit (bit 31) in the VM-entry interruption-information field is 1, and the interruption type (bits 10:8) in that field has value 2 (indicating NMI).
 - If bit 4 (enclave interruption) is 1, bit 1 (blocking by MOV-SS) must be 0 and the processor must support for SGX by enumerating CPUID.07H.00H:EBX.SGX[2] as 1.

1. As noted in Section 26.4.1, SS.DPL corresponds to the logical processor's current privilege level (CPL).

NOTE

If the “virtual NMIs” VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the VM-entry interruption-information field is 1 and the interruption type in that field has value 2.

- Pending debug exceptions.
 - Bits 11:4, bit 13, bit 15, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0.
 - The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
 - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
 - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
 - The following checks are performed if bit 16 (RTM) is 1:
 - Bits 11:0, bits 15:13, and bits 63:17 (bits 31:17 on processors that do not support Intel 64 architecture) must be 0; bit 12 must be 1.
 - The processor must support for RTM by enumerating CPUID.07H.00H:EBX[11] as 1.
 - The interruptibility-state field must not indicate blocking by MOV SS (bit 1 in that field must be 0).
- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
 - Bits 11:0 must be 0.
 - The field’s width is limited by MAXPHYADDR, a value derived from the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. The VMCS link pointer should not set bits in the range 63:MAXPHYADDR.
 - The 4 bytes located in memory referenced by the value of the field (as a physical address) must satisfy the following:
 - Bits 30:0 must contain the processor’s VMCS revision identifier (see Section 26.2).¹
 - Bit 31 must contain the setting of the “VMCS shadowing” VM-execution control.² This implies that the referenced VMCS is a shadow VMCS (see Section 26.10) if and only if the “VMCS shadowing” VM-execution control is 1.
 - If the processor is not in SMM or the “entry to SMM” VM-entry control is 1, the field must not contain the current VMCS pointer.
 - If the processor is in SMM and the “entry to SMM” VM-entry control is 0, the field must differ from the executive-VMCS pointer.

28.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0, the logical processor uses **PAE paging** (see Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A).³ When PAE paging is in use, the

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field. For all processors produced prior to this change, bit 31 of the VMCS revision identifier was 0.
2. “VMCS shadowing” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “VMCS shadowing” VM-execution control were 0. See Section 26.6.2.
3. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs.

A VM entry is to a guest that uses PAE paging if (1) bit 31 (corresponding to CR0.PG) is set in the CR0 field in the guest-state area; (2) bit 5 (corresponding to CR4.PAE) is set in the CR4 field; and (3) the “IA-32e mode guest” VM-entry control is 0. Such a VM entry checks the validity of the PDPTEs:

- If the “enable EPT” VM-execution control is 0, VM entry checks the validity of the PDPTEs referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.¹
- If the “enable EPT” VM-execution control is 1, VM entry checks the validity of the PDPTE fields in the guest-state area (see Section 26.4.2).

A VM entry to a guest that does not use PAE paging does not check the validity of any PDPTEs.

A VM entry that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use.² If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

28.3.2 Loading Guest State

Processor state is updated on VM entries in the following ways:

- Some state is loaded from the guest-state area.
- Some state is determined by VM-entry controls.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order and in parallel with the checking of VMCS contents (see Section 28.3.1).

The loading of guest state is detailed in Section 28.3.2.1 to Section 28.3.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

In addition to the state loading described in this section, VM entries may load MSRs from the VM-entry MSR-load area (see Section 28.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 28.3.1.

28.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).³ The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the “load debug controls” VM-entry control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored.

The first processors to support the virtual-machine extensions supported only the 1-setting of the “load debug controls” VM-entry control and thus always loaded DR7 from the DR7 field.

1. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “enable EPT” VM-execution control were 0. See Section 26.6.2.

2. This implies that (1) bits 11:9 in each PDPTE are ignored; and (2) if bit 0 (present) is clear in one of the PDPTEs, bits 63:1 of that PDPTE are ignored.

3. Bits 15:6, bit 17, and bit 28:19 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. Bits 15:6, bit 17, and bit 28:19 of CR0 are always 0 and CR0.ET is always 1.

- The following describes how certain MSRs are loaded using fields in the guest-state area:
 - If the “load debug controls” VM-entry control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
 - The following are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 28.3.2.2).
 - If the “load IA32_EFER” VM-entry control is 0, bits in the IA32_EFER MSR are modified as follows:
 - IA32_EFER.LMA is loaded with the setting of the “IA-32e mode guest” VM-entry control.
 - If CR0 is being loaded so that CR0.PG = 1, IA32_EFER.LME is also loaded with the setting of the “IA-32e mode guest” VM-entry control.¹ Otherwise, IA32_EFER.LME is unmodified.
- See below for the case in which the “load IA32_EFER” VM-entry control is 1
- If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
 - If the “load IA32_PAT” VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
 - If the “load IA32_EFER” VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.
 - If the “load IA32_BNDCFGS” VM-entry control is 1, the IA32_BNDCFGS MSR is loaded from the IA32_BNDCFGS field.
 - If the “load IA32_RTIT_CTL” VM-entry control is 1, the IA32_RTIT_CTL MSR is loaded from the IA32_RTIT_CTL field.
 - If the “load CET” VM-entry control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET field and the IA32_INTERRUPT_SSP_TABLE_ADDR field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
 - If the “load guest IA32_LBR_CTL” VM-entry control is 1, the IA32_LBR_CTL MSR is loaded from the IA32_LBR_CTL guest state field.
 - If the “load PKRS” VM-entry control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field.
 - If the “load UINV” VM-entry control is 1, UINV is loaded with the low 8 bits of the UINV field. UINV is represented in bits 39:32 of the IA32_UINTR_MISC MSR. The remainder of the MSR is not modified.
- With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 28.4.
- The SMBASE register is unmodified by all VM entries except those that return from SMM.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, VM entry must be loading CR0 so that CR0.PG = 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

28.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR, fields are loaded from the guest-state area as follows:

- The unusable bit is loaded from the access-rights field. This bit can never be set for TR (see Section 28.3.1.2). If it is set for one of the other registers, the following apply:
 - For each of CS, SS, DS, ES, FS, and GS, uses of the segment cause faults (general-protection exception or stack-fault exception) outside 64-bit mode, just as they would had the segment been loaded using a null selector. This bit does not cause accesses to fault in 64-bit mode.
 - If this bit is set for LDTR, uses of LDTR cause general-protection exceptions in all modes, just as they would had LDTR been loaded using a null selector.

If this bit is clear for any of CS, SS, DS, ES, FS, GS, TR, and LDTR, a null selector value does not cause a fault (general-protection exception or stack-fault exception).
- TR. The selector, base, limit, and access-rights fields are loaded.
- CS.
 - The following fields are always loaded: selector, base address, limit, and (from the access-rights field) the L, D, and G bits.
 - For the other fields, the unusable bit of the access-rights field is consulted:
 - If the unusable bit is 0, all of the access-rights field is loaded.
 - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, GS, and LDTR.
 - The selector fields are loaded.
 - For the other fields, the unusable bit of the corresponding access-rights field is consulted:
 - If the unusable bit is 0, the base-address, limit, and access-rights fields are loaded.
 - If the unusable bit is 1, the base address, the segment limit, and the remainder of the access rights are undefined after VM entry with the following exceptions:
 - Bits 3:0 of the base address for SS are cleared to 0.
 - SS.DPL is always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
 - SS.B is always set to 1.
 - The base addresses for FS and GS are loaded from the corresponding fields in the VMCS. On processors that support Intel 64 architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
 - On processors that support Intel 64 architecture, bits 63:32 of the base addresses for SS, DS, and ES are cleared to 0.

GDTR and IDTR are loaded using the base and limit fields.

28.3.2.3 Loading Guest RIP, RSP, RFLAGS, and SSP

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively.

If the “load CET” VM-entry control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

The following items regard the upper 32 bits of these fields on VM entries that are not to 64-bit mode:

- Bits 63:32 of RSP are undefined outside 64-bit mode. Thus, a logical processor may ignore the contents of bits 63:32 of the RSP field on VM entries that are not to 64-bit mode.
- As noted in Section 28.3.1.4, bits 63:32 of the RIP and RFLAGS fields must be 0 on VM entries that are not to 64-bit mode. (The same is true for SSP for VM entries that are not to 64-bit mode when the “load CET” VM-entry control is 1.)

28.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 28.3.1.6, the logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LME = 0. A VM entry to a guest that uses PAE paging loads the PDPTs into internal, non-architectural registers based on the setting of the “enable EPT” VM-execution control:

- If the control is 0, the PDPTs are loaded from the page-directory-pointer table referenced by the physical address in the value of CR3 being loaded by the VM entry (see Section 28.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTs are loaded from corresponding fields in the guest-state area (see Section 26.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

28.3.2.5 Updating Non-Register State

Section 30.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP).
- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

If the “virtual-interrupt delivery” VM-execution control is 1, VM entry loads the values of RVI and SVI from the guest interrupt-status field in the VMCS (see Section 26.4.2). After doing so, the logical processor first causes PPR virtualization (Section 31.1.3) and then evaluates pending virtual interrupts (Section 31.2.1).

If a virtual interrupt is recognized, it may be delivered in VMX non-root operation immediately after VM entry (including any specified event injection) completes; see Section 28.7.5. See Section 31.2.2 for details regarding the delivery of virtual interrupts.

28.3.3 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 10.10.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. VM entries clear any address-range monitoring that may be in effect.

28.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 26.8.2). Specifically each entry in that area (up to the number specified in the VM-entry MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.¹

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101 (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM entry did not commence in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM entries for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

1. Because attempts to modify the value of IA32_EFER.LMA by WRMSR are ignored, attempts to modify it using the VM-entry MSR-load area are also ignored.

- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.¹

The VM entry fails if processing fails for any entry. The logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 28.8.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition.

28.5 TRACE-ADDRESS PRE-TRANSLATION (TAPT)

When the “Intel PT uses guest physical addresses” VM-execution control is 1, the addresses used by Intel PT are treated as guest-physical addresses, and these are translated to physical addresses using EPT.

VM entry uses **trace-address pre-translation (TAPT)** to prevent buffered trace data from being lost due to an EPT violation; see Section 27.5.4.2. VM entry uses TAPT only if Intel PT will be enabled following VM entry (IA32_RTIT_CTL.TraceEn = 1) and only if the “Intel PT uses guest physical addresses” VM-execution control is 1

As noted in Section 27.5.4, TAPT may cause a VM exit due to an EPT violation, EPT misconfiguration, page-modification log-full event, or APIC access. If such a VM exit occurs as a result of TAPT during VM entry, the VM exit operates as if it had occurred in VMX non-root operation after the VM entry completed (in the guest context).

If TAPT during VM entry causes a VM exit, the VM entry does not perform event injection (Section 28.6), even if the valid bit in the VM-entry interruption-information field is 1. Such VM exits save the contents of VM-entry interruption-information and VM-entry exception error code fields into the IDT-vectoring information and IDT-vectoring error code fields, respectively.

28.6 EVENT INJECTION

If the valid bit in the VM-entry interruption-information field (see Section 26.8.3) is 1, VM entry causes an event to be delivered (or made pending) after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.

- If the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), the event is delivered as described in Section 28.6.1.
- If the interruption type in the field is 7 (other event) and the vector field is 0, an MTF VM exit is pending after VM entry. See Section 28.6.2.

28.6.1 Vectored-Event Injection

VM entry delivers an injected vectored event within the guest context established by VM entry. This means that delivery occurs after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.² The event is delivered using the vector in that field to select a descriptor in the IDT. Since event injection occurs after loading IDTR from the guest-state area, this is the guest IDT.

Section 28.6.1.1 provides details of vectored-event injection. In general, the event is delivered exactly as if it had been generated normally.

1. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. If VM entry has established CR0.PG = 1, the IA32_EFER MSR should not be included in the VM-entry MSR-load area for the purpose of modifying the LME bit.

2. This does not imply that injection of an exception or interrupt will cause a VM exit due to the settings of VM-execution control fields (such as the exception bitmap) that would cause a VM exit if the event had occurred in VMX non-root operation. In contrast, a nested exception encountered during event delivery may cause a VM exit; see Section 28.6.1.1.

An exception is made if the following all hold: bit 25 (UINTR) is set to 1 in the guest CR4 field and the “IA-32e mode guest” VM-entry control is 1, and VM entry is modified if it is injecting an external interrupt whose vector is the value that UINV would have after VM entry. In this case, the logical processor then performs user-interrupt notification processing as specified in Section 8.5.2 instead of the process described in Section 28.6.1.1. (If the guest activity-state field indicated the HLT state, the logical processor enters the HLT state following user-interrupt notification processing.)

If event delivery (or user-interrupt notification processing; see above) encounters a nested exception (for example, a general-protection exception because the vector indicates a descriptor beyond the IDT limit), the exception bitmap is consulted using the vector of that exception:

- If the bit for the nested exception is 0, the nested exception is delivered normally. If the nested exception is benign, it is delivered through the IDT. If it is contributory or a page fault, a double fault may be generated, depending on the nature of the event whose delivery encountered the nested exception. See Chapter 7, “Interrupt 8—Double Fault Exception (#DF)” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.¹
- If the bit for the nested exception is 1, a VM exit occurs. Section 28.6.1.2 details cases in which event injection causes a VM exit.

28.6.1.1 Details of Vectored-Event Injection

The event-injection process is controlled by the contents of the VM-entry interruption information field (format given in Table 26-18), the VM-entry exception error-code field, and the VM-entry instruction-length field. The following items provide details of the process:

- The value pushed on the stack for RFLAGS is generally that which was loaded from the guest-state area. The value pushed for the RF flag is not modified based on the type of event being delivered. However, the pushed value of RFLAGS may be modified if a software interrupt is being injected into a guest that will be in virtual-8086 mode (see below). After RFLAGS is pushed on the stack, the value in the RFLAGS register is modified as is done normally when delivering an event through the IDT.
- The instruction pointer that is pushed on the stack depends on the type of event and whether nested exceptions occur during its delivery. The term **current guest RIP** refers to the value to be loaded from the guest-state area. The value pushed is determined as follows:²
 - If VM entry successfully injects (with no nested exception) an event with interruption type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.
 - If VM entry successfully injects (with no nested exception) an event with interruption type software interrupt, privileged software exception, or software exception, the current guest RIP is incremented by the VM-entry instruction length before being pushed on the stack.
 - If VM entry encounters an exception while injecting an event and that exception does not cause a VM exit, the current guest RIP is pushed on the stack regardless of event type or VM-entry instruction length. If the encountered exception does cause a VM exit that saves RIP, the saved RIP is current guest RIP.
- If the deliver-error-code bit (bit 11) is set in the VM-entry interruption-information field, the contents of the VM-entry exception error-code field is pushed on the stack as an error code would be pushed during delivery of an exception.
- DR6, DR7, and the IA32_DEBUGCTL MSR are not modified by event injection, even if the event has vector 1 (normal deliveries of debug exceptions, which have vector 1, do update these registers).
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode (RFLAGS.VM = 1), no general-protection exception can occur due to RFLAGS.IOPL < 3. A VM monitor should check RFLAGS.IOPL before injecting such an event and, if desired, inject a general-protection exception instead of a software interrupt.

1. Hardware exceptions with the following unused vectors are considered benign: 15 and 21–31. A hardware exception with vector 20 is considered benign unless the processor supports the 1-setting of the “EPT-violation #VE” VM-execution control; in that case, it has the same severity as page faults.

2. While these items refer to RIP, the width of the value pushed (16 bits, 32 bits, or 64 bits) is determined normally.

- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode with virtual-8086 mode extensions ($RFLAGS.VM = CR4.VME = 1$), event delivery is subject to VME-based interrupt redirection based on the software interrupt redirection bitmap in the task-state segment (TSS) as follows:
 - If bit n in the bitmap is clear (where n is the number of the software interrupt), the interrupt is directed to an 8086 program interrupt handler: the processor uses a 16-bit interrupt-vector table (IVT) located at linear address zero. If the value of $RFLAGS.IOPL$ is less than 3, the following modifications are made to the value of $RFLAGS$ that is pushed on the stack: $IOPL$ is set to 3, and IF is set to the value of VIF .
 - If bit n in the bitmap is set (where n is the number of the software interrupt), the interrupt is directed to a protected-mode interrupt handler. (In other words, the injection is treated as described in the next item.) In this case, the software interrupt does not invoke such a handler if $RFLAGS.IOPL < 3$ (a general-protection exception occurs instead). However, as noted above, $RFLAGS.IOPL$ cannot cause an injected software interrupt to cause such an exception. Thus, in this case, the injection invokes a protected-mode interrupt handler independent of the value of $RFLAGS.IOPL$.

Injection of events of other types are not subject to this redirection.

- If VM entry is injecting a software interrupt (not redirected as described above) or software exception, privilege checking is performed on the IDT descriptor being accessed as would be the case for executions of $INT\ n$, $INT3$, or $INTO$ (the descriptor's DPL cannot be less than CPL). There is no checking of $RFLAGS.IOPL$, even if the guest will be in virtual-8086 mode. Failure of this check may lead to a nested exception. Injection of an event with interruption type external interrupt, NMI, hardware exception, and privileged software exception, or with interruption type software interrupt and being redirected as described above, do not perform these checks.
- If VM entry is injecting a non-maskable interrupt (NMI) and the "virtual NMIs" VM-execution control is 1, virtual-NMI blocking is in effect after VM entry.
- The transition causes a last-branch record to be logged if the LBR bit is set in the $IA32\_DEBUGCTL$ MSR. This is true even for events such as debug exceptions, which normally clear the LBR bit before delivery.
- The last-exception record MSRs (LERs) may be updated based on the setting of the LBR bit in the $IA32\_DEBUGCTL$ MSR. Events such as debug exceptions, which normally clear the LBR bit before they are delivered, and therefore do not normally update the LERs, may do so as part of VM-entry event injection.
- If injection of an event encounters a nested exception, the value of the EXT bit (bit 0) in any error code for that nested exception is determined as follows:
 - If event being injected has interruption type external interrupt, NMI, hardware exception, or privileged software exception and encounters a nested exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If event being injected is a software interrupt or a software exception and encounters a nested exception, the error code for that exception clears the EXT bit.
 - If event delivery encounters a nested exception and delivery of that exception encounters another exception (but does not produce a double fault), the error code for that exception sets the EXT bit.
 - If a double fault is produced, the error code for the double fault is 0000H (the EXT bit is clear).

28.6.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the "NMI exiting" VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the VM-entry interruption-information field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 27.4.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 27.2).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 27.2).

- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 27.2).
- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access is treated as described in Section 31.4 and may cause a VM exit.¹

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the IDT-vectoring information field (see Section 29.2.4).

The material in this section applies also if injection of an external interrupt results in user-interrupt notification processing instead of event delivery (see Section 28.6.1 earlier).

28.6.1.3 Event Injection for VM Entries to Real-Address Mode

If VM entry is loading CR0.PE with 0, any injected vectored event is delivered as would normally be done in real-address mode.² Specifically, VM entry uses the vector provided in the VM-entry interruption-information field to select a 4-byte entry from an interrupt-vector table at the linear address in IDTR.base. Further details are provided in Section 15.1.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

Because bit 11 (deliver error code) in the VM-entry interruption-information field must be 0 if CR0.PE will be 0 after VM entry (see Section 28.2.1.3), vectored events injected with CR0.PE = 0 do not push an error code on the stack. This is consistent with event delivery in real-address mode.

If event delivery encounters a fault (due to a violation of IDTR.limit or of SS.limit), the fault is treated as if it had occurred during event delivery in VMX non-root operation. Such a fault may lead to a VM exit as discussed in Section 28.6.1.2.

28.6.2 Injection of Pending MTF VM Exits

If the interruption type in the VM-entry interruption-information field is 7 (other event) and the vector field is 0, VM entry causes an MTF VM exit to be pending on the instruction boundary following VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0. See Section 27.5.2 for the treatment of pending MTF VM exits.

28.7 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **vectoring** if the valid bit (bit 31) of the VM-entry interruption information field is 1 and the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

28.7.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 26-3) contains bits that control blocking by STI, blocking by MOV SS, and blocking by NMI. This field impacts event blocking after VM entry as follows:

- If the VM entry is vectoring, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.

1. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 26.6.2.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, VM entry must be loading CR0.PE with 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- If the VM entry is not vectoring, the following apply:
 - Events are blocked by STI if and only if bit 0 in the interruptibility-state field is 1. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry; see Section 28.7.3).
 - Events are blocked by MOV SS if and only if bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 28.7.3. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).
 - The blocking of non-maskable interrupts (NMIs) is determined as follows:
 - If the “virtual NMIs” VM-execution control is 0, NMIs are blocked if and only if bit 3 (blocking by NMI) in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, execution of the IRET instruction removes this blocking (even if the instruction generates a fault). If the “NMI exiting” control is 1, IRET does not affect this blocking.
 - The following items describe the use of bit 3 (blocking by NMI) in the interruptibility-state field if the “virtual NMIs” VM-execution control is 1:
 - The bit’s value does not affect the blocking of NMIs after VM entry. NMIs are not blocked in VMX non-root operation (except for ordinary blocking for other reasons, such as by the MOV SS instruction, the wait-for-SIPI state, etc.)
 - The bit’s value determines whether there is virtual-NMI blocking after VM entry. If the bit is 1, virtual-NMI blocking is in effect after VM entry. If the bit is 0, there is no virtual-NMI blocking after VM entry unless the VM entry is injecting an NMI (see Section 28.6.1.1). Execution of IRET removes virtual-NMI blocking (even if the instruction generates a fault).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 28.2.1.1.
- Blocking of system-management interrupts (SMIs) is determined as follows:
 - If the VM entry was not executed in system-management mode (SMM), SMI blocking is unchanged by VM entry.
 - If the VM entry was executed in SMM, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.

28.7.2 Activity State

The activity-state field in the guest-state area controls whether, after VM entry, the logical processor is active or in one of the inactive states identified in Section 26.4.2. The use of this field is determined as follows:

- If the VM entry is vectoring, the logical processor is in the active state after VM entry. While the consistency checks described in Section 28.3.1.5 on the activity-state field do apply in this case, the contents of the activity-state field do not determine the activity state after VM entry.
- If the VM entry is not vectoring, the logical processor ends VM entry in the activity state specified in the guest-state area. If VM entry ends with the logical processor in an inactive activity state, the VM entry generates any special bus cycle that is normally generated when that activity state is entered from the active state. If VM entry would end with the logical processor in the shutdown state and the logical processor is in SMX operation,¹ an Intel® TXT shutdown condition occurs. [See the Intel® Trusted Execution Technology Measured Launched Environment Developer’s Guide. The error code used is 0000H, indicating “legacy shutdown.”](#)
- Some activity states unconditionally block certain events. The following blocking is in effect after any VM entry that puts the processor in the indicated state:
 - The active state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the active state and in VMX non-root operation are discarded and do not cause VM exits.
 - The HLT state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the HLT state and in VMX non-root operation are discarded and do not cause VM exits.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

- The shutdown state blocks external interrupts and SIPIs. External interrupts that arrive while a logical processor is in the shutdown state and in VMX non-root operation do not cause VM exits even if the “external-interrupt exiting” VM-execution control is 1. SIPIs that arrive while a logical processor is in the shutdown state and in VMX non-root operation are discarded and do not cause VM exits.
- The wait-for-SIPI state blocks external interrupts, non-maskable interrupts (NMIs), INIT signals, and system-management interrupts (SMIs). Such events do not cause VM exits if they arrive while a logical processor is in the wait-for-SIPI state and in VMX non-root operation.

28.7.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section 26.4.2). This section describes how these are treated on VM entry.

There are no pending debug exceptions after VM entry if any of the following are true:

- The VM entry is vectoring with one of the following interruption types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is vectoring with either of the following interruption type: software interrupt or software exception.
- The VM entry is not vectoring and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not vectoring, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
 - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
 - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.
- If the VM entry is vectoring (with interruption type software interrupt or software exception and with blocking by MOV SS), the following items apply:
 - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF) — or a privileged software exception with vector 1 (#DB) — the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT1, INT3, or INTO) were executed after a MOV SS that encountered a debug trap.
 - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of “traps on the previous instruction” (see Section 7.9 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A). Thus, INIT signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR threshold (see Section 28.7.7) and pending MTF VM exits (see Section 28.7.8). The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt and also over VM exits due to the 1-settings of the “interrupt-window exiting” and “NMI-window exiting” VM-execution controls.

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

28.7.4 VMX-Preemption Timer

If the “activate VMX-preemption timer” VM-execution control is 1, VM entry starts the VMX-preemption timer with the unsigned value in the VMX-preemption timer-value field.

It is possible for the VMX-preemption timer to expire during VM entry (e.g., if the value in the VMX-preemption timer-value field is zero). If this happens (and if the VM entry was not to the wait-for-SIPI state), a VM exit occurs with its normal priority after any event injection and before execution of any instruction following VM entry. For example, any pending debug exceptions established by VM entry (see Section 28.7.3) take priority over a timer-induced VM exit. (The timer-induced VM exit will occur after delivery of the debug exception, unless that exception or its delivery causes a different VM exit.)

See Section 27.5.1 for details of the operation of the VMX-preemption timer in VMX non-root operation, including the blocking and priority of the VM exits that it causes.

28.7.5 Interrupt-Window Exiting and Virtual-Interrupt Delivery

If “interrupt-window exiting” VM-execution control is 1, an open interrupt window may cause a VM exit immediately after VM entry (see Section 27.2 for details). If the “interrupt-window exiting” VM-execution control is 0 but the “virtual-interrupt delivery” VM-execution control is 1, a virtual interrupt may be delivered immediately after VM entry (see Section 28.3.2.5 and Section 31.2.1).

The following items detail the treatment of these events:

- These events occur after any event injection specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over these events. These events take priority over external interrupts and lower priority events.
- These events wake the logical processor if it just entered the HLT state because of a VM entry (see Section 28.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

28.7.6 NMI-Window Exiting

The “NMI-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 27.2 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Debug-trap exceptions (see Section 28.7.3) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered either the HLT state or the shutdown state because of a VM entry (see Section 28.7.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

28.7.7 VM Exits Induced by the TPR Threshold

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1 and the “virtual-interrupt delivery” VM-execution control is 0, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 of VTPR (see Section 31.1.1).¹

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
- The VM exits follow event injection if such injection is specified for VM entry.
- VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 28.7.5, Section 28.7.6, and Section 28.7.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.

1. “Virtualize APIC accesses” and “virtual-interrupt delivery” are secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if these controls were 0. See Section 26.6.2.

- These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 28.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state. If such a VM exit is suppressed because the processor just entered the shutdown state, it occurs after the delivery of any event that cause the logical processor to leave the shutdown state while remaining in VMX non-root operation (e.g., due to an NMI that occurs while the “NMI-exiting” VM-execution control is 0).
- The basic exit reason is “TPR below threshold.”

28.7.8 Pending MTF VM Exits

As noted in Section 28.6.2, VM entry may cause an MTF VM exit to be pending immediately after VM entry. The following items detail the treatment of these VM exits:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these VM exits. These VM exits take priority over debug-trap exceptions and lower priority events.
- These VM exits wake the logical processor if it just entered the HLT state because of a VM entry (see Section 28.7.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

28.7.9 VM Entries and Advanced Debugging Features

VM entries are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

28.7.10 User-Interrupt Recognition After VM Entry

A VM entry results in recognition of a pending user interrupt if it completes with $CR4.UINTR = IA32\_EFER.LMA = 1$ and with $UIRR \neq 0$; otherwise, no pending user interrupt is recognized.

28.8 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 28.3.1 and failures during the MSR loading identified in Section 28.4 are treated differently from those that occur earlier in VM entry. In these cases, the following steps take place:

1. Information about the VM-entry failure is recorded in the VM-exit information fields:
 - Exit reason.
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM-entry failure. The following numbers are used:
 33. VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 28.3.1.
 34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 28.4).
 41. VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 28.9).
 - Bit 31 is set to 1 to indicate a VM-entry failure.
 - The remainder of the field (bits 30:16) is cleared.
 - Exit qualification. This field is set based on the exit reason.
 - VM-entry failure due to invalid guest state. In most cases, the exit qualification is cleared to 0. The following non-zero values are used in the cases indicated:
 1. Not used.
 2. Failure was due to a problem loading the PDPTEs (see Section 28.3.1.6).

3. Failure was due to an attempt to inject a non-maskable interrupt (NMI) into a guest that is blocking events through the STI blocking bit in the interruptibility-state field.
4. Failure was due to an invalid VMCS link pointer (see Section 28.3.1.5).

VM-entry checks on guest-state fields may be performed in any order. Thus, an indication by exit qualification of one cause does not imply that there are not also other errors. Different processors may give different exit qualifications for the same VMCS.

- VM-entry failure due to MSR loading. The exit qualification is loaded to indicate which entry in the VM-entry MSR-load area caused the problem (1 for the first entry, 2 for the second, etc.).

— All other VM-exit information fields are unmodified.

2. Processor state is loaded as would be done on a VM exit (see Section 29.5). If this results in $[CR4.PAE \ \& \ CR0.PG \ \& \ \sim IA32\_EFER.LMA] = 1$, page-directory-pointer-table entries (PDPTes) may be checked and loaded (see Section 29.5.4).
3. The state of blocking by NMI is what it was before VM entry.
4. MSRs are loaded as specified in the VM-exit MSR-load area (see Section 29.6).

Although this process resembles that of a VM exit, many steps taken during a VM exit do not occur for these VM-entry failures:

- Most VM-exit information fields are not updated (see step 1 above).
- The valid bit in the VM-entry interruption-information field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

28.9 MACHINE-CHECK EVENTS DURING VM ENTRY

If a machine-check event occurs during a VM entry, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM entry:
 - If $CR4.MCE = 0$, operation of the logical processor depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If $CR4.MCE = 1$, a machine-check exception (#MC) is delivered through the IDT.
- The machine-check event is handled after VM entry completes:
 - If the VM entry ends with $CR4.MCE = 0$, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM entry ends with $CR4.MCE = 1$, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- A VM-entry failure occurs as described in Section 28.8. The basic exit reason is 41, for “VM-entry failure due to machine-check event.”

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

The first option is not used if the machine-check event occurs after any guest state has been loaded. The second option is used only if VM entry is able to load all guest state.

22. Updates to Chapter 29, Volume 3C

Change bars and **violet** text show changes to Chapter 29 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Added UDB in the description of Section 29.2.2, "Information for VM Exits Due to Vectored Events". UDB was also added UDB to list of sources that generate BOUND-range exceeded exceptions and invalid opcode exceptions.
- In Section 29.2.4, "Information for VM Exits During Event Delivery", added UDB to list of sources that generate BOUND-range exceeded exceptions and invalid opcode exceptions.
- Added SEAMCALL and TDCALL to the list of instructions that can cause fault-like exits under for VM-exit instruction length in Section 29.2.5, "Information for VM Exits Due to Instruction Execution".
- Updated the range of CR3 bits that cannot be modified with MAXPHYADDR in Section 29.5.1, "Loading Host Control Registers, Debug Registers, MSRs".

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 27.1 through Section 27.2. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 29.2.
2. Processor state is saved in the guest-state area (Section 29.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 29.4). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.
4. The following may be performed in parallel and in any order (Section 29.5):
 - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 33.15.6 for information on how processor state is loaded by such VM exits.
 - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 29.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 29.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 29.2 through Section 29.6.

Section 33.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, ordinary transitions to SMM are replaced by VM exits to a separate SMM monitor. Called **SMM VM exits**, these are caused by the arrival of an SMI or the execution of VMCALL in VMX root operation. SMM VM exits differ from other VM exits in ways that are detailed in Section 33.15.2.

29.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered through the IDT. Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the “NMI exiting” VM-execution control is 1. An external interrupt causes a VM exit directly if the “external-interrupt exiting” VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.
- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 31.4), EPT violation, EPT misconfiguration, page-modification log-full event (see Section 30.3.6), or SPP-related event (see Section 30.3.4) that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had it not caused the VM exit:
 - A debug exception does not update DR6, DR7, or IA32_DEBUGCTL. (Information about the nature of the debug exception is saved in the exit qualification field.)
 - A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)

- An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
 - An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the “acknowledge interrupt on exit” VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
 - The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
 - If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT¹; or the TR register.
 - No last-exception record is made if the event that would do so directly causes a VM exit.
 - If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.
 - If the logical processor is in an inactive state (see Section 26.4.2) and not executing instructions, some events may be blocked but others may return the logical processor to the active state. Unblocked events may cause VM exits.² If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.³ The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.
- MTF VM exits (see Section 27.5.2 and Section 28.7.8) are not blocked in the HLT activity state. If an MTF VM exit occurs in the HLT activity state, the logical processor returns to the active state only after the VM exit completes. MTF VM exits are blocked the shutdown state and the wait-for-SIPI state.
- If an event causes a VM exit indirectly, the event does update architectural state:
 - A debug exception updates DR6, DR7, and the IA32_DEBUGCTL MSR. No debug exceptions are considered pending.
 - A page fault updates CR2.
 - An NMI causes subsequent NMIs to be blocked before the VM exit commences.
 - An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.
 - If the logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.
 - There is no blocking by STI or by MOV SS when the VM exit commences.
 - Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
 - The treatment of last-exception records is implementation dependent:
 - Some processors make a last-exception record when beginning the delivery of an event through the IDT (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
 - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-exception record if a VM exit or triple fault occurs before an event handler is reached.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

2. If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.

3. An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

- If the “virtual NMIs” VM-execution control is 1, VM entry injects an NMI, and delivery of the NMI causes a nested exception, double fault, task switch, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event, or APIC access that causes a VM exit, virtual-NMI blocking is in effect before the VM exit commences.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the VM-exit interruption-information field; see Section 29.2.3.
- If a VM exit results from a fault, EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the exit qualification or in the VM-exit interruption-information field; see Section 29.2.3.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered through the IDT. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MC_G_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, APIC access (see Section 31.4), EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that is encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (unless the VM exit clears that field; see Section 29.3.4).
- The following VM exits are considered to happen after an instruction is executed:
 - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
 - VM exits resulting from debug exceptions (data breakpoints) whose recognition was delayed by blocking by MOV SS.
 - VM exits resulting from some machine-check exceptions.
 - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1 (see Section 31.3). (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)
 - Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1; the value of ECX is in the range 800H–8FFH; and the bit corresponding to the ECX value in write bitmap for low MSRs is 0; and the “virtualize x2APIC mode” VM-execution control is 1. See Section 31.5.
 - VM exits caused by APIC-write emulation (see Section 31.4.3.2) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 26-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

A VM exit that occurs in enclave mode sets bit 27 of the exit-reason field and bit 4 of the guest interruptibility-state field. Before such a VM exit is delivered, an Asynchronous Enclave Exit (AEX) occurs (see Chapter 39, “Enclave Exiting Events”). An AEX modifies architectural state (Section 39.3). In particular, the processor establishes the following architectural state as indicated:

- The following bits in RFLAGS are cleared: CF, PF, AF, ZF, SF, OF, and RF.
- FS and GS are restored to the values they had prior to the most recent enclave entry.
- RIP is loaded with the AEP of interrupted enclave thread.
- RSP is loaded from the URSP field in the enclave’s state-save area (SSA).

29.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 29.2.1 to Section 29.2.5 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the VM-entry interruption-information field. If bit 5 of the IA32_VMX_MISC MSR (index 485H) is read as 1 (see Appendix A.6), the value of IA32_EFER.LMA is stored into the “IA-32e mode guest” VM-entry control.¹

29.2.1 Basic VM-Exit Information

Section 26.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
 - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.

Future processors may introduce new VM exits with new basic exit reasons. Any such VM exit will occur only with VM-execution control settings introduced by the same processor or due to execution of an instruction that would cause an invalid-opcode exception (#UD) on previous processors. Software written for a particular processor generation should be aware that, if run on a later processor generation, it may experience VM exits with new basic exit reasons; it should also be aware that such a VM exit would result only from execution of an instruction that would have caused #UD on the earlier processor generation.
 - Bit 25 is set if the “prematurely busy shadow stack” VM-exit control is 1 and the VM exit caused a shadow stack become prematurely busy (see Section 27.4.3). Otherwise, the bit is cleared.
 - Bit 26 of this field is set to 1 if the VM exit occurred after assertion of a bus lock while the “VMM bus-lock detection” VM-execution control was 1. Such VM exits include those that occur due to the 1-setting of that control as well as others that might occur during execution of an instruction that asserted a bus lock.
 - Bit 27 of this field is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

Such VM exits include those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field indicates an enclave interruption (bit 4 of the field was 1); or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
 - The remainder of the field (bits 31:28 and bits 24:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 33.15.2.3).²
- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPs); system-management interrupts (SMIs) that arrive immediately after the execution of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; RDMSRLIST; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; WBINVD; WBNOINVD; WRMSR; WRMSRLIST; WRMSRNS; XRSTORS; XSAVES; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 31.4); EPT violations (see Section 30.3.3.2); EOI virtualization (see Section 31.1.4); APIC-write emulation (see Section 31.4.3.3); page-modification log full (see Section 30.3.6); SPP-related events (see Section 30.3.4); and instruction timeout (see Section 27.2). For all other VM exits, this field is cleared. The following items provide details:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.

2. Bit 31 of this field is set on certain VM-entry failures; see Section 28.8.

- For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 29-1.

Table 29-1. Exit Qualification for Debug Exceptions

Bit Position(s)	Contents
3:0	B3 – B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
10:4	Not currently defined.
11	BLD. When set, this bit indicates that a bus lock was asserted while OS bus-lock detection was enabled and CPL > 0 (see Section 19.3.1.6 (“OS Bus-Lock Detection”)). ¹
12	Not currently defined.
13	BD. When set, this bit indicates that the cause of the debug exception is “debug register access detected.”
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
15	Not currently defined.
16	RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). ²
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 11 to indicate detection of a bus lock, while this field **sets** the bit to indicate that condition.
2. In general, the format of this field matches that of DR6. However, DR6 **clears** bit 16 to indicate an RTM-related exception, while this field **sets** the bit to indicate that condition.

- For a page-fault exception, the exit qualification contains the linear address that caused the page fault. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.

If the page-fault exception occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

- For a start-up IPI (SIPI), the exit qualification contains the SIPI vector information in bits 7:0. Bits 63:8 of the exit qualification are cleared to 0.
- For a task switch, the exit qualification contains details about the task switch, encoded as shown in Table 29-2.
- For INVLPG, the exit qualification contains the linear-address operand of the instruction.
 - On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
 - If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. This address is not architecturally defined and may be implementation-specific.
- For INVEPT, INVPCID, INVVPID, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, and XSAVES, the exit qualification receives the value of the instruction’s displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors

Table 29-2. Exit Qualification for Task Switches

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Not currently defined
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction's address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 26.9.4 and Section 29.2.5) reports an n -bit address size. Then bits 63: n (bits 31: n on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

- For a control-register access, the exit qualification contains information about the access and has the format given in Table 29-3.
- For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 29-4.
- For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 29-5.
- For MWAIT, the exit qualification contains a value that indicates whether address-range monitoring hardware was armed. The exit qualification is set either to 0 (if address-range monitoring hardware is not armed) or to 1 (if address-range monitoring hardware is armed).
- For RDMSRLIST and WRMSRLIST, the exit qualification depends on the setting of the “use MSR bitmaps” VM-execution control. If the control is 0, the exit qualification is zero. If the control is 1, the exit qualification is the index of the MSR whose access caused the VM exit (see Section 27.1.3).
- WBINVD and WBNOINVD use the same basic exit reason (see Appendix C). For WBINVD, the exit qualification is 0, while for WBNOINVD it is 1.
- WRMSR and WRMSRNS use the same basic exit reason (see Appendix C). For WRMSR, the exit qualification is 0, while for WRMSRNS it is 1.
- For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 31.4), the exit qualification contains information about the access and has the format given in Table 29-6.¹

If the access to the APIC-access page occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of the exit qualification are cleared.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data

1. The exit qualification is undefined if the access was part of the logging of a branch record or a processor-event-based-sampling (PEBS) record to the DS save area. It is recommended that software configure the paging structures so that no address in the DS save area translates to an address on the APIC-access page.

write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For an APIC-access VM exit caused by the CLFLUSH and CLFLUSHOPT instructions, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused by the ENTER instruction, the access type is “data write during instruction execution.”

Table 29-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Not currently defined
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Not currently defined
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

- For an APIC-access VM exit caused by the MASKMOVQ instruction or the MASKMOVDQU instruction, the access type is “data write during instruction execution.”
- For an APIC-access VM exit caused by the MONITOR instruction, the access type is “data read during instruction execution.”
- For an APIC-access VM exit caused directly by an access to a linear address in the DS save area (BTS or PEBS), the access type is “linear access for monitoring.”
- For an APIC-access VM exit caused by a guest-physical access performed for an access to the DS save area (e.g., to access a paging structure to translate a linear address), the access type is “guest-physical access for monitoring or trace.”

- For an APIC-access VM exit caused by trace-address pre-translation (TAPT) when the “Intel PT uses guest physical addresses” VM-execution control is 1, the access type is “guest-physical access for monitoring or trace.”

Such a VM exit stores 1 for bit 31 for IDT-vectoring information field (see Section 29.2.4) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 31.4.4 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses to the APIC-access page (see Section 31.4.6), the exit qualification is undefined.

- For an EPT violation, the exit qualification contains information about the access causing the EPT violation and has the format given in Table 29-7.

As noted in that table, the format and meaning of the exit qualification depends on the setting of the “mode-based execute control for EPT” VM-execution control and whether the processor supports advanced VM-exit information for EPT violations.¹

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

Table 29-4. Exit Qualification for MOV DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Not currently defined
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Not currently defined
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 - 15 = R8 - R15, respectively
63:12	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

1. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

Table 29-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Not currently defined
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Not currently defined. These bits exist only on processors that support Intel 64 architecture.

Table 29-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	- If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. - Undefined if the APIC-access VM exit is due a guest-physical access
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 4 = linear access for monitoring 10 = guest-physical access during event delivery 11 = guest-physical access for monitoring or trace 15 = guest-physical access for an instruction fetch or during instruction execution Other values not used
16	This bit is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These includes guest-physical accesses related to trace output by Intel PT (see Section 27.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 21.9.5), and accesses that occur during user-interrupt delivery (see Section 8.4.2).
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

Bit 12 reports “NMI unblocking due to IRET”; see Section 29.2.3.

Bit 16 is set for certain accesses that are asynchronous to instruction execution and not part of event delivery. These include trace-address pre-translation (TAPT) for Intel PT (see Section 27.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 21.9.5), and accesses as part of user-interrupt delivery (see Section 8.4.2).

- For VM exits caused as part of EOI virtualization (Section 31.1.4), bits 7:0 of the exit qualification are set to vector of the virtual interrupt that was dismissed by the EOI virtualization. Bits above bit 7 are cleared.

- For APIC-write VM exits (Section 31.4.3.3), bits 11:0 of the exit qualification are set to the page offset of the write access that caused the VM exit.¹ Bits above bit 11 are cleared.
- For a VM exit due to a page-modification log-full event (Section 30.3.6), bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 29.2.3). Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, or user-interrupt delivery. All other bits of the exit qualification are undefined.
- For a VM exit due to an SPP-related event (Section 30.3.4), bit 11 of the exit qualification indicates the type of event: 0 indicates an SPP misconfiguration and 1 indicates an SPP miss. Bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 29.2.3). Bit 16 is set if the VM exit occurs during TAPT, EPT-friendly PEBS, or user-interrupt delivery. All other bits of the exit qualification are undefined.
- If the “PASID translation” VM-execution control, PASID translation is performed for executions of the ENQCMD and ENQCMLS instructions (see Section 27.5.8). PASID translation may fail, resulting in a VM exit. Such a VM exit saves an exit qualification specified in the following items:
 - For ENQCMD, the exit qualification is IA32_PASID[19:0].
 - For ENQCMLS, the exit qualification contains the low 32 bits of the instruction’s source operand (which had been read from memory prior to PASID translation).
- For a VM exit due to an instruction timeout (Section 27.2), bit 0 indicates (if set) that the context of the virtual machine is invalid and that the VM should not be resumed. Bit 12 of the exit qualification reports “NMI unblocking due to IRET” (see Section 29.2.3). All other bits of the exit qualification are undefined.
- **Guest linear address.** For some VM exits, this field receives a linear address that pertains to the VM exit. The field is set for different VM exits as follows:
 - VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded is the original address before any masking (and may thus contain any metadata).
 - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 29-7; these are all EPT violations except those resulting from an attempt to load the PDPTES as of execution of the MOV CR instruction and those due to TAPT). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.

If the EPT violation occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.
 - VM exits due to SPP-related events. If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - If the “prematurely busy shadow stack” VM-exit control is 1, certain VM exits (besides those noted above) save the linear address that pertains to the VM exit if the VM exit caused a shadow stack to become prematurely busy (see Section 27.4.3). This is true for VM exits due for these reasons: EPT misconfiguration, page-modification log-full event, and instruction timeout. (A VM exit due to instruction timeout that sets bit 0 of the exit qualification, indicating that VM context is invalid, does not save a valid linear address.) If linear-address masking had been in effect (Section 4.4), the address recorded reflects the result of that masking and does not contain any masked metadata.
 - For all other VM exits, the field is undefined.

1. Execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit; the exit qualification for such an APIC-write VM exit is 3F0H.

- **Guest-physical address.** For a VM exit due to an EPT violation, an EPT misconfiguration, or an SPP-related event, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

If the EPT violation or EPT misconfiguration occurred during execution of an instruction in enclave mode (and not during delivery of an event incident to enclave mode), bits 11:0 of this field are cleared.

Table 29-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read. ¹
1	Set if the access causing the EPT violation was a data write. ¹
2	Set if the access causing the EPT violation was an instruction fetch.
3	The logical-AND of bit 0 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was readable). ²
4	The logical-AND of bit 1 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation (indicates whether the guest-physical address was writeable). ²
5	The logical-AND of bit 2 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. ² If the “mode-based execute control for EPT” VM-execution control is 0, this indicates whether the guest-physical address was executable. If that control is 1, this indicates whether the guest-physical address was executable for supervisor-mode linear addresses.
6	If the “mode-based execute control” VM-execution control is 0, the value of this bit is undefined. If that control is 1, this bit is the logical-AND of bit 10 in the EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation. In this case, it indicates whether the guest-physical address was executable for user-mode linear addresses. ³
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTes as part of the execution of the MOV CR instruction and those due to trace-address pre-translation (TAPT; Section 27.5.4).
8	If bit 7 is 1: ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
9	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if the linear address is a supervisor-mode linear address and 1 if it is a user-mode linear address. (If CR0.PG = 0, the translation of every linear address is a user-mode linear address and thus this bit will be 1.) Otherwise, this bit is undefined.
10	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if paging translates the linear address to a read-only page and 1 if it translates to a read/write page. (If CR0.PG = 0, every linear address is read/write and thus this bit will be 1.) Otherwise, this bit is undefined.
11	If bit 7 is 1, bit 8 is 1, and the processor supports advanced VM-exit information for EPT violations, ⁴ this bit is 0 if paging translates the linear address to an executable page and 1 if it translates to an execute-disable page. (If CR0.PG = 0, CR4.PAE = 0, or IA32_EFER.NXE = 0, every linear address is executable and thus this bit will be 0.) Otherwise, this bit is undefined.
12	NMI unblocking due to IRET (see Section 29.2.3).

Table 29-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
13	Set if the access causing the EPT violation was a shadow-stack access.
14	If supervisor shadow-stack control is enabled (by setting bit 7 of EPTP), this bit is the same as bit 60 in the EPT paging-structure entry that maps the page of the guest-physical address of the access causing the EPT violation. Otherwise (or if translation of the guest-physical address terminates before reaching an EPT paging-structure entry that maps a page), this bit is undefined.
15	This bit is set if the EPT violation was caused as a result of guest-paging verification. See Section 30.3.3.2.
16	This bit is set if the access was asynchronous to instruction execution not the result of event delivery. The bit is set if the access is related to trace output by Intel PT (see Section 27.5.4), accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 21.9.5), or to user-interrupt delivery (see Section 8.4.2). Otherwise, this bit is cleared.
63:17	Not currently defined. Bits 63:32 exist only on processors that support Intel 64 architecture.

NOTES:

1. If accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations (see Section 30.3.3.2). If such an access causes an EPT violation, the processor sets both bit 0 and bit 1 of the exit qualification.
2. Bits 5:3 are cleared to 0 if either (1) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (2) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 30.3.2).
3. Bit 6 is cleared to 0 if (1) the “mode-based execute control” VM-execution control is 1; and (2) either (a) any of EPT paging-structure entries used to translate the guest-physical address of the access causing the EPT violation is not present; or (b) 4-level EPT is in use and the guest-physical address sets any bits in the range 51:48 (see Section 30.3.2).
4. Software can determine whether advanced VM-exit information for EPT violations is supported by consulting the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10).

29.2.2 Information for VM Exits Due to Vectored Events

Section 26.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT1, INT3, INTO, BOUND, UD1, UD2, and UDB); external interrupts that occur while the “acknowledge interrupt on exit” VM-exit control is 1; and non-maskable interrupts (NMIs).¹ Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **VM-exit interruption information** (format given in Table 26-20). The following items detail how this field is established for VM exits due to these events:
 - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 5 (privileged software exception), or 6 (software exception). Hardware exceptions comprise all exceptions except the following:
 - Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
 - Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

1. INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, UD2, and UDB are hardware exceptions.

- Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).¹ If bit 11 is set to 1, the error code is placed in the VM-exit interruption error code (see below).
- Bit 12 reports “NMI unblocking due to IRET”; see Section 29.2.3. The value of this bit is undefined if the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- **VM-exit interruption error code.**

- For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).
- For other VM exits, the value of this field is undefined.

29.2.3 Information About NMI Unblocking Due to IRET

A VM exit may occur during execution of the IRET instruction for reasons including the following: faults, EPT violations, page-modification log-full events, SPP-related events, or instruction timeouts.

An execution of IRET that commences while non-maskable interrupts (NMIs) are blocked will unblock NMIs even if a fault or VM exit occurs; the state saved by such a VM exit will indicate that NMIs were not blocked.

VM exits for the reasons enumerated above provide more information to software by saving a bit called “NMI unblocking due to IRET.” This bit is defined if (1) either the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1; (2) the VM exit does not set the valid bit in the IDT-vectoring information field (see Section 29.2.4); and (3) the VM exit is not due to a double fault. In these cases, the bit is defined as follows:

- The bit is 1 if the VM exit resulted from a memory access as part of execution of the IRET instruction and one of the following holds:
 - The “virtual NMIs” VM-execution control is 0 and blocking by NMI (see Table 26-3) was in effect before execution of IRET.
 - The “virtual NMIs” VM-execution control is 1 and virtual-NMI blocking was in effect before execution of IRET.
- The bit is 0 for all other relevant VM exits.

For VM exits due to faults, NMI unblocking due to IRET is saved in bit 12 of the VM-exit interruption-information field (Section 29.2.2). For VM exits due to EPT violations, page-modification log-full events, SPP-related events, and instruction timeouts, NMI unblocking due to IRET is saved in bit 12 of the exit qualification (Section 29.2.1).

(Executions of IRET may also incur VM exits due to APIC accesses and EPT misconfigurations. These VM exits do not report information about NMI unblocking due to IRET.)

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

29.2.4 Information for VM Exits During Event Delivery

Section 26.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT and as a result of any of the following cases:¹

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 27.4.2).
- Event delivery causes an APIC-access VM exit (see Section 31.4).
- An EPT violation, EPT misconfiguration, page-modification log-full event, or SPP-related event that occurs during event delivery.
- Any of the above VM exits that occur during user-interrupt notification processing (see Section 8.5.2). Such VM exits will be treated as if they occurred during delivery of an external interrupt with the vector UINV.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 28.6.1.2).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the “NMI exiting” VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.
- The original event was a software interrupt (INT *n*) executed in virtual-8086 mode with EFLAGS.IOPL < 3 and the VM exit was due to a general-protection exception (#GP) that occurred because either CR4.VME = 0 or bit *n* of the software interrupt redirection bit map in the TSS is set.

The following items detail the use of these fields:

- IDT-vectoring information (format given in Table 26-21). The following items detail how this field is established for VM exits that occur during event delivery:
 - If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the vector.
 - Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

Hardware exceptions comprise all exceptions except the following:²

- Debug exceptions (#DB) generated by the INT1 instruction; these are privileged software exceptions. (Other debug exceptions are considered hardware exceptions, as are those caused by executions of INT1 in enclave mode.)
- Breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. (A #BP that occurs in enclave mode is considered a hardware exception.)

BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD0, UD1, UD2, and UDB are hardware exceptions.

- Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in

1. This includes the case in which a VM exit occurs while delivering a software interrupt (INT *n*) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).

2. In the following items, INT1 and INT3 refer to the instructions with opcodes F1 and CC, respectively, and not to INT *n* with value 1 or 3 for *n*.

real-address mode (CR0.PE=0).¹ If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).

- Bit 12 is undefined.
- Bits 30:13 are always set to 0.
- Bit 31 is always set to 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
 - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
 - For other VM exits, the value of this field is undefined.

29.2.5 Information for VM Exits Due to Instruction Execution

Section 26.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
 - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 27.1.2) or based on the settings of VM-execution controls (see Section 27.1.3): CLTS, CPUID, ENCLS, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLD, LMSW, LOADIWKEY, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, PCONFIG, RDMSR, RDP, RDRAND, RDSEED, RDTSC, RDTSCP, RSM, SEAMCALL, SGDT, SIDT, SLDT, STR, TDCALL, SGDT, SIDT, SLDT, STR, TPAUSE, UMWAIT, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WBNOINVD, WRMSR, XRSTORS, XSETBV, and XSAVES.²
 - For VM exits due to software exceptions (those generated by executions of INT3 or INTO) or privileged software exceptions (those generated by executions of INT1).
 - For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, or software exception.
 - For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
 - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
 - An exit qualification indicating a task gate in the IDT and an IDT-vectoring information field indicating that the task gate was encountered during delivery of a software interrupt, privileged software exception, or software exception.
 - For APIC-access VM exits and for VM exits caused by EPT violations, page-modification log-full events, and SPP-related events encountered during delivery of a software interrupt, privileged software exception, or software exception.³
 - For VM exits due executions of VMFUNC that fail because one of the following is true:

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
2. This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.
3. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 31.4.6) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

- EAX indicates a VM function that is not enabled (the bit at position EAX is 0 in the VM-function controls; see Section 27.5.6.2).
- EAX = 0 and either ECX $\geq$ 512 or the value of ECX selects an invalid tentative EPTP value (see Section 27.5.6.3).

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 28.6.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

If the VM exit occurred in enclave mode, this field is cleared (none of the previous items apply).

Table 29-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS

Bit Position(s)	Content
6:0	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
14:10	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for VM exits due to execution of INS.
31:18	Undefined.

- **VM-exit instruction information.** For VM exits due to attempts to execute INS, INVEPT, INVPCID, INVVPID, LIDT, LGDT, LLDT, LOADIWKEY, LTR, OUTS, RDRAND, RDSEED, SIDT, SGDT, SLDT, STR, TPAUSE, UMWAIT, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON, XRSTORS, or XSAVES, this field receives information about the instruction that caused the VM exit. The format of the field depends on the identity of the instruction causing the VM exit:
 - For VM exits due to attempts to execute INS or OUTS, the field has the format is given in Table 29-8.¹
 - For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 29-9.
 - For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 29-10.
 - For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 29-11.
 - For VM exits due to attempts to execute RDRAND or RDSEED, the field has the format is given in Table 29-12.

1. The format of the field was undefined for these VM exits on the first processors to support the virtual-machine extensions. Software can determine whether the format specified in Table 29-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

- For VM exits due to attempts to execute TPAUSE or UMWAIT, the field has the format is given in Table 29-13.
- For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, or XSAVES, the field has the format is given in Table 29-14.
- For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 29-15.
- For VM exits due to attempts to execute LOADIWKEY, the field has the format is given in Table 29-16.

For all other VM exits, the field is undefined, unless the VM exit occurred in enclave mode, in which case the field is cleared.

- **I/O RCX, I/O RSI, I/O RDI, I/O RIP.** These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 33.15.2.3. Note that, if the VM exit occurred in enclave mode, these fields are all cleared.
- **MSR data.** An execution of WRMSRLIST may cause a VM exit if it would write to an MSR for which the MSR bitmaps do not allow writes (see Section 27.1.3). Such VM exits save the 64-bit data that would have been written to the MSR into this field in the VMCS.

Table 29-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).

Table 29-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID (Contd.)

Bit Position(s)	Content
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for memory instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Reg2 (same encoding as IndexReg above)

Table 29-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
11	Operand size: 0: 16-bit 1: 32-bit Undefined for VM exits from 64-bit mode.
14:12	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)

Table 29-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
29:28	Instruction identity: 0: SGDT 1: SIDT 2: LGDT 3: LIDT
31:30	Undefined.

Table 29-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).

Table 29-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)

Bit Position(s)	Content
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
29:28	Instruction identity: 0: SLDT 1: STR 2: LLDT 3: LTR
31:30	Undefined.

Table 29-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND and RDSEED

Bit Position(s)	Content
2:0	Undefined.
6:3	Operand register (destination register): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8-15 represent R8-R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

Table 29-13. Format of the VM-Exit Instruction-Information Field as Used for TPAUSE and UMWAIT

Bit Position(s)	Content
2:0	Undefined.

Table 29-13. Format of the VM-Exit Instruction-Information Field as Used for TPAUSE and UMWAIT (Contd.)

Bit Position(s)	Content
6:3	Operand register (source register): 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture)
31:7	Undefined.

Table 29-14. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).

Table 29-14. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, VMXON, XRSTORS, and XSAVES (Contd.)

Bit Position(s)	Content
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

Table 29-15. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).

Table 29-15. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE (Contd.)

Bit Position(s)	Content
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

Table 29-16. Format of the VM-Exit Instruction-Information Field as Used for LOADIWKEY

Bit Position(s)	Content
2:0	Undefined.
6:3	Reg1: identifies the first XMM register operand (XMM0–XMM15; values 8–15 are used only on processors that support Intel 64 architecture).
30:7	Undefined.
31:28	Reg2: identifies the second XMM register operand (see above).

29.3 SAVING GUEST STATE

VM exits save certain components of processor state into corresponding fields in the guest-state area of the VMCS (see Section 26.4). On processors that support Intel 64 architecture, the full value of each natural-width field (see Section 26.11.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 29.1 for a discussion of which architectural updates occur at that time.

Section 29.3.1 through Section 29.3.4 provide details for how various components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

29.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs are saved as follows:

- The contents of CR0, CR3, CR4, and the IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are saved into the corresponding fields. Bits 63:32 of the IA32_SYSENTER_CS MSR are not saved. On processors that do not support Intel 64 architecture, bits 63:32 of the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are not saved.
- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the IA32_DEBUGCTL MSR are saved into the corresponding fields. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save IA32_PAT” VM-exit control is 1, the contents of the IA32_PAT MSR are saved into the corresponding field.
- If the “save IA32_EFER” VM-exit control is 1, the contents of the IA32_EFER MSR are saved into the corresponding field.
- If the processor supports either the 1-setting of the “load IA32_BNDCFGS” VM-entry control or that of the “clear IA32_BNDCFGS” VM-exit control, the contents of the IA32_BNDCFGS MSR are saved into the corresponding field.

- If the processor supports either the 1-setting of the “load IA32_RTIT_CTL” VM-entry control or that of the “clear IA32_RTIT_CTL” VM-exit control, the contents of the IA32_RTIT_CTL MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are saved into the corresponding fields. On processors that do not support Intel 64 architecture, bits 63:32 of these MSRs are not saved.
- If the processor supports either the 1-setting of the “load guest IA32_LBR_CTL” VM-entry control or that of the “clear IA32_LBR_CTL” VM-exit control, the contents of the IA32_LBR_CTL MSR are saved into the corresponding field.
- If the processor supports the 1-setting of the “load PKRS” VM-entry control, the contents of the IA32_PKRS MSR are saved into the corresponding field.
- If a processor supports user interrupts, every VM exit saves UINV into the guest UINV field in the VMCS (bits 15:8 of the field are cleared).
- If the “save IA32_PERF_GLOBAL_CTL” VM-exit control is 1, the contents of the IA32_PERF_GLOBAL_CTL MSR are saved into the corresponding field.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 33.15.2.

29.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 26.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
 - CS.
 - The base-address and segment-limit fields are saved.
 - The L, D, and G bits are saved in the access-rights field.
 - SS.
 - DPL is saved in the access-rights field.
 - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
 - DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
 - FS and GS. The base-address field is saved.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

29.3.3 Saving RIP, RSP, RFLAGS, and SSP

The contents of the RIP, RSP, RFLAGS, and SSP (shadow-stack pointer) registers are saved as follows:

- The value saved in the RIP field is determined by the nature and cause of the VM exit:
 - If the VM exit occurred in enclave mode, the value saved is the AEP of interrupted enclave thread (the remaining items do not apply).
 - If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.

- If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
- If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
- If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 29.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,¹ or into the old task-state segment had the event been delivered through a task gate).
- If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
- If the VM exit is due to a software exception (due to an execution of INT3 or INTO) or a privileged software exception (due to an execution of INT1), the value saved references the INT3, INTO, or INT1 instruction that caused that exception.
- Suppose that the VM exit is due to a task switch that was caused by execution of CALL, IRET, or JMP or by execution of a software interrupt (INT *n*), software exception (due to execution of INT3 or INTO), or privileged software exception (due to execution of INT1) that encountered a task gate in the IDT. The value saved references the instruction that caused the task switch (CALL, IRET, JMP, INT *n*, INT3, INTO, INT1).
- Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The value saved is that which would have been saved in the old task-state segment had the task switch completed normally.
- If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of bits 7:4 of VTPR (see Section 31.1.1) below that of TPR threshold VM-execution control field (see Section 31.1.2), the value saved references the instruction following the MOV to CR8 or WRMSR.
- If the VM exit was caused by APIC-write emulation (see Section 31.4.3.2) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the APIC-write emulation.
- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
 - If the VM exit occurred in enclave mode, the value saved is 0 (the remaining items do not apply).
 - If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate² or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
 - If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
 - If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.

1. The reference here is to the full value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

2. The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.

- If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.¹
- For APIC-access VM exits and for VM exits caused by EPT violations, EPT misconfigurations, page-modification log-full events, or SPP-related events, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:
 - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 29.2.4), the value saved is 1.
 - If the VM exit stored 1 for bit 31 for IDT-vectoring information field (because the VM exit did occur during delivery of an event through the IDT), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered through the IDT (see above).
- For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.
- If the processor supports the 1-setting of the “load CET” VM-entry control, the contents of the SSP register are saved into the SSP field.

29.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor’s activity state before the VM exit.² See Section 29.1 for details of how events leading to a VM exit may affect the activity state. If the VM exit occurred during user-interrupt notification processing (see Section 8.5.2) and the logical processor would have entered the HLT state following user-interrupt notification processing, the saved activity state is “HLT”.
- The interruptibility-state field is saved to reflect the logical processor’s interruptibility before the VM exit.
 - See Section 29.1 for details of how events leading to a VM exit may affect this state.
 - VM exits that end outside system-management mode (SMM) save bit 2 (blocking by SMI) as 0 regardless of the state of such blocking before the VM exit.
 - Bit 3 (blocking by NMI) is treated specially if the “virtual NMIs” VM-execution control is 1. In this case, the value saved for this field does not indicate the blocking of NMIs but rather the state of virtual-NMI blocking.
 - Bit 4 (enclave interruption) is set to 1 if the VM exit occurred while the logical processor was in enclave mode.

Such VM exits includes those caused by interrupts, non-maskable interrupts, system-management interrupts, INIT signals, and exceptions occurring in enclave mode as well as exceptions encountered during the delivery of such events incident to enclave mode.

A VM exit also sets this bit if it was due to or incident to delivery of an event that was either (1) made pending or injected by VM entry while the guest interruptibility-state field sets this bit; or (2) pending following an execution of RSM that occurred while bit 1 was set in the byte at offset 7EE0H in the state save image in SMRAM.
- The pending debug exceptions field is saved as clear for all VM exits except the following:
 - A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
 - A VM exit with basic exit reason “TPR below threshold”,³ “virtualized EOI”, “APIC write”, “monitor trap flag”, or “bus-lock detected.”
 - A VM exit due to trace-address pre-translation (TAPT; see Section 27.5.4) or due to accesses related to PEBS on processors with the “EPT-friendly” enhancement (see Section 21.9.5). Such VM exits can have

1. This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.
2. If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.
3. This item includes VM exits that occur as a result of certain VM entries (Section 28.7.7).

basic exit reason “APIC access,” “EPT violation,” “EPT misconfiguration,” “page-modification log full,” or “SPP-related event.” When due to TAPT or PEBS, these VM exits (with the exception of those due to EPT misconfigurations) set bit 16 of the exit qualification, indicating that they are asynchronous to instruction execution and not part of event delivery.

- VM exits that are not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

For VM exits that do not clear the field, the value saved is determined as follows:

- Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.
- Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason “TPR below threshold” or “monitor trap flag.” In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 28.7.3). Otherwise, the following items apply:

- Bit 12 (enabled breakpoint) is set to 1 in any of the following cases:
 - If there was at least one matched data or I/O breakpoint that was enabled in DR7.
 - If it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 28.7.3) and the VM exit occurred before those exceptions were either delivered or lost.
 - If the XBEGIN instruction was executed immediately before the VM exit and advanced debugging of RTM transactional regions had been enabled (see Section 17.3.7, “RTM-Enabled Debugger Support,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1). (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
 - If a bus lock was asserted while CPL > 0 and OS bus-lock detection was enabled.

In other cases, bit 12 is cleared to 0.

- Bit 14 (BS) is set if RFLAGS.TF = 1 in either of the following cases:
 - IA32_DEBUGCTL.BTF = 0 and the cause of a pending debug exception was the execution of a single instruction.
 - IA32_DEBUGCTL.BTF = 1 and the cause of a pending debug exception was a taken branch.
- Bit 16 (RTM) is set if a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions had been enabled. (This does not apply to VM exits with basic exit reason “monitor trap flag.”)
- Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 28.7.3). Otherwise, the following items apply:
 - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 28.7.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
 - The setting of bit 14 (BS) is implementation-specific. However, it is not set if RFLAGS.TF = 0 or IA32_DEBUGCTL.BTF = 1.
- The reserved bits in the field are cleared.
- If the “save VMX-preemption timer value” VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 27.5.1). VM exits due to timer expiration save the value 0. Other VM exits may also save the value 0 if the timer expired during VM exit. (If the “save VMX-preemption timer value” VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
- If the logical processor supports the 1-setting of the “enable EPT” VM-execution control, values are saved into the four (4) PDPTE fields as follows:

- If the “enable EPT” VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTe values currently in use are saved:¹
 - The values saved into bits 11:9 of each of the fields is undefined.
 - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
 - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.
- If the “enable EPT” VM-execution control is 0 or the logical processor was not using PAE paging at the time of the VM exit, the values saved are undefined.

29.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 26.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMBASE is an MSR that can be read only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 29.7.

29.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 29.5.1 to Section 29.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

1. A logical processor uses PAE paging if CRO.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the “enable EPT” VM-execution control were 0. See Section 26.6.2.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 29.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 29.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 29.6). This loading occurs only after the state loading described in this section.

29.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
 - The following bits are not modified:
 - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 25.8).¹
 - For CR3, bits 63:MAXPHYADDR, where MAXPHYADDR is defined as follows:
 - Usually, MAXPHYADDR is the processor’s supported physical-address width as enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52).
 - If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.²
 - This item applies only to processors that support Intel 64 architecture.
 - For CR4, any bits that are fixed in VMX operation (see Section 25.8).
 - CR4.PAE is set to 1 if the “host address-space size” VM-exit control is 1.
 - CR4.PCIDE is set to 0 if the “host address-space size” VM-exit control is 0.
 - DR7 is set to 400H.
 - If the “clear UINV” VM-exit control is 1, VM exit clears UINV.
 - The following MSRs are established as follows:
 - The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
 - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
 - The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP fields, respectively.
- If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
- If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit $N-1$.³
- The following steps are performed on processors that support Intel 64 architecture:
 - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 29.5.2).

1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.

2. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

3. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

- The LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the “host address-space size” VM-exit control.
- If the “load IA32_PERF_GLOBAL_CTRL” VM-exit control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field. Bits that are reserved in that MSR are maintained with their reserved values.
- If the “load IA32_PAT” VM-exit control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field. Bits that are reserved in that MSR are maintained with their reserved values.
- If the “load IA32_EFER” VM-exit control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field. Bits that are reserved in that MSR are maintained with their reserved values.
- If the “clear IA32_BNDCFGS” VM-exit control is 1, the IA32_BNDCFGS MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
- If the “clear IA32_RTIT_CTL” VM-exit control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H; otherwise, it is not modified.
- If the “load CET” VM-exit control is 1, the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR MSRs are loaded from the IA32_S_CET and IA32_INTERRUPT_SSP_TABLE_ADDR fields, respectively.
If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
If the processor supports the Intel 64 architecture with $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.
- If the “load PKRS” VM-exit control is 1, the IA32_PKRS MSR is loaded from the IA32_PKRS field. Bits 63:32 of that MSR are maintained with zeroes.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 29.6.

29.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. The checks specified in Section 28.2.3 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).
- The base address is set as follows:
 - CS. Cleared to zero.
 - SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
 - FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field.
If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.¹ The values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
 - TR. Loaded from the host-state area. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N is set to the value of bit N-1.
- The segment limit is set as follows:
 - CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFFFH and a G-bit setting of 1).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

- TR. Set to 00000067H.
- The type field and S bit are set as follows:
 - CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
 - TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
 - CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
 - CS, TR. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 29.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
 - CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
 - SS. Set to 1.
 - DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.
- G.
 - CS. Set to 1.
 - SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
 - TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined.

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports $N < 64$ linear-address bits, each of bits 63:N of each base address is set to the value of bit N–1 of that base address. The GDTR and IDTR limits are each set to FFFFH.

29.5.3 Loading Host RIP, RSP, RFLAGS, and SSP

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

If the “load CET” VM-exit control is 1, SSP (shadow-stack pointer) is loaded from the SSP field.

29.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 5.4 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.¹ When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

when PAE paging is in use checks the validity of the PDPTes and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the “host address-space size” VM-exit control is 0. Such a VM exit may check the validity of the PDPTes referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTes.

A VM exit that checks the validity of the PDPTes uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTes that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 29.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTes are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

29.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.
- Event blocking is affected as follows:
 - There is no blocking by STI or by MOV SS after a VM exit.
 - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 26-3). Other VM exits do not affect blocking by NMI. (See Section 29.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 30.4 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

29.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 10.10.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. VM exits clear any address-range monitoring that may be in effect.

29.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 26.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101H (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 0000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)

- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.¹

If processing fails for any entry, a VMX abort occurs. See Section 29.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

29.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shut-down state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 26.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 29.4).
2. Host checking of the page-directory-pointer-table entries (PDPTes) failed (see Section 29.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.
4. There was a failure on loading host MSRs (see Section 29.6).
5. There was a machine-check event during VM exit (see Section 29.8).
6. The logical processor was in IA-32e mode before the VM exit and the “host address-space size” VM-exit control was 0 (see Section 29.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 29.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTes might not be properly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:²

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating “VMX abort.” See the Intel® Trusted Execution Technology Measured Launched Environment Programming Guide.

1. Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

2. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference,” in Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

29.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
 - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:²
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating “unrecoverable machine-check condition.”
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If CR4.MCE = 1, a machine-check exception (#MC) is generated:
 - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
 - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
 - If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
 - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
 - If the logical processor is outside SMX operation, it goes to the shutdown state.
 - If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered through the host IDT.
- A VMX abort is generated (see Section 29.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine-check event during VM exit.”

The first option is not used if the machine-check event occurs after any host state has been loaded. The second option is used only if VM entry is able to load all host state.

29.9 USER-INTERRUPT RECOGNITION AFTER VM EXIT

A VM exit results in recognition of a pending user interrupt if it completes with CR4.UINTR = IA32_EFER.LMA = 1 and with UIRR ≠ 0; otherwise, no pending user interrupt is recognized.

23. Updates to Chapter 30, Volume 3C

Change bars and violet text show changes to Chapter 30 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Added physical and guest-physical addresses in Section 30.3.1, "EPT Overview".
- In Section 30.3.3.1, "EPT Misconfigurations", updated the definition for the reserved bit to MAXPHYADDR or above.
- Updated the range of reserved bits to 63:MAXPHYADDR for entry of SPPL4E, SPPL3E or SPPL2E in Section 30.3.4.2, "Determining an Access's Sub-Page Write Permission".
- Added footnote to clarify "M" as an abbreviation for MAXPHYADDR in
 - Table 30-1, "Format of an EPT PML5 Entry (PML5E) that References an EPT PML4 Table"
 - Table 30-2, "Format of an EPT PML4 Entry (PML4E) that References an EPT Page-Directory-Pointer Table"
 - Table 30-3, "Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GBYTE Page"
 - Table 30-4, "Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that References an EPT Page Directory"
 - Table 30-5, "Format of an EPT Page-Directory Entry (PDE) that Maps a 2-MByte Page"
 - Table 30-6, "Format of an EPT Page-Directory Entry (PDE) that References an EPT Page Table"
 - Table 30-7, "Format of an EPT Page-Table Entry that Maps a 4-KByte Page"
 - Figure 30-1, "Formats of EPTP and EPT Paging-Structure Entries"

CHAPTER 30

VMX SUPPORT FOR ADDRESS TRANSLATION

The architecture for VMX operation includes two features that support address translation: virtual-processor identifiers (VPIDs) and the extended page-table mechanism (EPT). VPIDs are a mechanism for managing translations of linear addresses. EPT defines a layer of address translation that augments the translation of linear addresses.

Section 30.1 details the architecture of VPIDs. Section 30.3 provides the details of EPT. Section 30.4 explains how a logical processor may cache information from the paging structures, how it may use that cached information, and how software can managed the cached information.

30.1 VIRTUAL PROCESSOR IDENTIFIERS (VPIDS)

The original architecture for VMX operation required VMX transitions to flush the TLBs and paging-structure caches. This ensured that translations cached for the old linear-address space would not be used after the transition.

Virtual-processor identifiers (**VPIDs**) introduce to VMX operation a facility by which a logical processor may cache information for multiple linear-address spaces. When VPIDs are used, VMX transitions may retain cached information and the logical processor switches to a different linear-address space.

Section 30.4 details the mechanisms by which a logical processor manages information cached for multiple address spaces. A logical processor may tag some cached information with a 16-bit VPID. This section specifies how the current VPID is determined at any point in time:

- The current VPID is 0000H in the following situations:
 - Outside VMX operation. (This includes operation in system-management mode under the default treatment of SMIs and SMM with VMX operation; see Section 33.14.)
 - In VMX root operation.
 - In VMX non-root operation when the “enable VPID” VM-execution control is 0.
- If the logical processor is in VMX non-root operation and the “enable VPID” VM-execution control is 1, the current VPID is the value of the VPID VM-execution control field in the VMCS. (VM entry ensures that this value is never 0000H; see Section 28.2.1.1.)

VPIDs and PCIDs (see Section 5.10.1) can be used concurrently. When this is done, the processor associates cached information with both a VPID and a PCID. Such information is used only if the current VPID and PCID **both** match those associated with the cached information.

30.2 HYPERVISOR-MANAGED LINEAR-ADDRESS TRANSLATION (HLAT)

Hypervisor-managed linear-address translation (HLAT) is a feature that changes the way in which linear addresses are translated in VMX non-root operation. Instead of ordinary paging, translation uses a modified process called **HLAT paging**.

HLAT paging is used only if the “enable HLAT” VM-execution control is 1. HLAT paging is used only for the paging modes 4-level paging and 5-level paging. Because HLAT paging is a modification of ordinary paging, details of the feature are given in Section 5.5, which describes the operation of 4-level paging and 5-level paging.

30.3 THE EXTENDED PAGE TABLE MECHANISM (EPT)

The extended page-table mechanism (**EPT**) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain addresses that would normally be treated as physical addresses (and used to access memory) are instead treated as **guest-physical addresses**. Guest-physical addresses are translated by traversing a set of **EPT paging structures** to produce physical addresses that are used to access memory.

- Section 30.3.1 gives an overview of EPT.
- Section 30.3.2 describes operation of EPT-based address translation.
- Section 30.3.3 discusses VM exits that may be caused by EPT.
- Section 30.3.7 describes interactions between EPT and memory typing.

30.3.1 EPT Overview

EPT is used when the “enable EPT” VM-execution control is 1.¹ It translates the guest-physical addresses used in VMX non-root operation and those used by VM entry for event injection.

The translation from guest-physical addresses to physical addresses is determined by a set of **EPT paging structures**. The EPT paging structures are similar to those used to translate linear addresses while the processor is in IA-32e mode. Section 30.3.2 gives the details of the EPT paging structures.

If CR0.PG = 1, linear addresses are translated through paging structures referenced through control register CR3.² While the “enable EPT” VM-execution control is 1, these are called **guest paging structures**. There are no guest paging structures if CR0.PG = 0.³

When the “enable EPT” VM-execution control is 1, the identity of **guest-physical addresses** depends on the value of CR0.PG:

- If CR0.PG = 0, each linear address is treated as a guest-physical address.
- If CR0.PG = 1, guest-physical addresses are those derived from the contents of control register CR3 and the guest paging structures. (This includes the values of the PDPTes, which logical processors store in internal, non-architectural registers.) The latter includes (in page-table entries and in other paging-structure entries for which bit 7—PS—is 1) the addresses to which linear addresses are translated by the guest paging structures.

If CR0.PG = 1, the translation of a linear address to a physical address requires multiple translations of guest-physical addresses using EPT. Assume, for example, that CR4.PAE = CR4.PSE = 0. The translation of a 32-bit linear address then operates as follows:

- Bits 31:22 of the linear address select an entry in the guest page directory located at the guest-physical address in CR3. The guest-physical address of the guest page-directory entry (PDE) is translated through EPT to determine the guest PDE’s physical address.
- Bits 21:12 of the linear address select an entry in the guest page table located at the guest-physical address in the guest PDE. The guest-physical address of the guest page-table entry (PTE) is translated through EPT to determine the guest PTE’s physical address.
- Bits 11:0 of the linear address is the offset in the page frame located at the guest-physical address in the guest PTE. The guest-physical address determined by this offset is translated through EPT to determine the physical address to which the original linear address translates.

In addition to translating a guest-physical address to a physical address, EPT specifies the privileges that software is allowed when accessing the address. Attempts at disallowed accesses are called **EPT violations** and cause VM exits. See Section 30.3.3.

A processor uses EPT to translate guest-physical addresses only when those addresses are used to access memory. This principle implies the following:

- The MOV to CR3 instruction loads CR3 with a guest-physical address. Whether that address is translated through EPT depends on whether PAE paging is being used.⁴

1. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, the logical processor operates as if the “enable EPT” VM-execution control were 0. See Section 26.6.2.
2. If HLAT paging is enabled, the processor uses the HLATP VMCS field instead of CR3. See Section 5.5.1. For simplicity, the remainder of this section refers only to CR3.
3. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
4. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 5.4 in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

- If PAE paging is not being used, the instruction does not use that address to access memory and does **not** cause it to be translated through EPT. (If CR0.PG = 1, the address will be translated through EPT on the next memory accessing using a linear address.)
- If PAE paging is being used, the instruction loads the four (4) page-directory-pointer-table entries (PDPTEs) from that address and it **does** cause the address to be translated through EPT.
- Section 5.4.1 identifies executions of MOV to CR0 and MOV to CR4 that load the PDPTEs from the guest-physical address in CR3. Such executions cause that address to be translated through EPT.
- The PDPTEs contain guest-physical addresses. The instructions that load the PDPTEs (see above) do not use those addresses to access memory and do **not** cause them to be translated through EPT. The address in a PDPTE will be translated through EPT on the next memory accessing using a linear address that uses that PDPTE.

The widths of physical and guest-physical addresses are limited. CPUID.80000008H:EAX[7:0] reports the physical-address width supported by the processor. This width is at most 52 and limits the width of both physical and guest-physical addresses:

- **Physical addresses.** The limit on physical addresses is denoted MAXPHYADDR and is derived from CPUID.80000008H:EAX[7:0]. If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.¹
For example, if CPUID.80000008H:EAX[7:0] = 48, IA32_TME_ACTIVATE[39:36] = 4, and TME has been configured, MAXPHYADDR is 44 outside SEAM (but remains 48 in SEAM).
- **Guest-physical addresses.** The width of these addresses is limited by the value enumerated by CPUID.80000008H:EAX[7:0] without any regard to TME or SEAM.

30.3.2 EPT Translation Mechanism

The EPT translation mechanism can be configured in either of two modes: **4-level EPT** or **5-level EPT**. 4-level EPT accesses at most 4 EPT paging-structure entries (an EPT page-walk length of 4) to translate a guest-physical address and uses only bits 47:0 of each guest-physical address. In contrast, 5-level EPT may access up to 5 EPT paging-structure entries (an EPT page-walk length of 5) and uses guest-physical address bits 56:0.²

The EPT page-walk length is configured using the extended-page-table pointer (EPTP), a VM-execution control field (see Table 26-9 in Section 26.6.11). Specifically, bits 5:3 contain a value one less than EPT page-walk length. Thus, a value of 3 configures 4-level EPT, while a value of 4 configures 5-level EPT.³

The remainder of this section describes the translation process used by 4-level EPT and 5-level EPT. Because the processes used by the two EPT modes are similar, they are described together in the following items (with any differences identified):

- With 5-level EPT, 4-KByte naturally aligned EPT PML5 table is located at the physical address specified in bits 51:12 of the EPTP. An EPT PML5 table comprises 512 64-bit entries (EPT PML5Es). An EPT PML5E is selected using the physical address defined as follows:
 - Bits 63:52 are all 0.
 - Bits 51:12 are from the EPTP.
 - Bits 11:3 are bits 56:48 of the guest-physical address.⁴
 - Bits 2:0 are all 0.

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.
2. Physical addresses and guest-physical addresses are architecturally limited to 52 bits (e.g., by paging), so in practice bits 56:52 are zero. See Section 30.3.1
3. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT page-walk lengths are supported.
4. Bits 56:52 of each guest-physical address are necessarily zero because guest-physical addresses are architecturally limited to 52 bits.

Because an EPT PML5E is identified using bits 56:48 of the guest-physical address, it controls access to a 256-TByte region of the guest-physical-address space. The format of an EPT PML5E is given in Table 30-1.

Table 30-1. Format of an EPT PML5 Entry (PML5E) that References an EPT PML4 Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 256-TByte region controlled by this entry.
1	Write access; indicates whether writes are allowed to the 256-TByte region controlled by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 256-TByte region controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 256-TByte region controlled by this entry.
7:3	Reserved (must be 0).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 256-TByte region controlled by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	Ignored.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 256-TByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M-1:12	Physical address of 4-KByte aligned EPT PML4 table referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

- With 4-level EPT, bits 51:48 of the guest-physical address must all be zero; otherwise, an EPT violation occurs (see Section 30.3.3).
- A 4-KByte naturally aligned EPT PML4 table is located at the physical address specified in the EPTP (for 4-level EPT) or in the EPT PML5E (for 5-level EPT). An EPT PML4 table comprises 512 64-bit entries (EPT PML4Es). An EPT PML4E is selected using the physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the EPTP (for 4-level EPT) or from bits 51:12 of the EPT PML4E (for 5-level EPT).
- Bits 11:3 are bits 47:39 of the guest-physical address.
- Bits 2:0 are all 0.

Because an EPT PML4E is identified using bits 47:39 of the guest-physical address, it controls access to a 512-GByte region of the guest-physical-address space. The format of an EPT PML4E is given in Table 30-2.

- A 4-KByte naturally aligned EPT page-directory-pointer table is located at the physical address specified in bits 51:12 of the EPT PML4E. An EPT page-directory-pointer table comprises 512 64-bit entries (EPT PDPTEs). An EPT PDPTE is selected using the physical address defined as follows:
- Bits 63:52 are all 0.
- Bits 51:12 are from the EPT PML4E.
- Bits 11:3 are bits 38:30 of the guest-physical address.
- Bits 2:0 are all 0.

Table 30-2. Format of an EPT PML4 Entry (PML4E) that References an EPT Page-Directory-Pointer Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 512-GByte region controlled by this entry.
1	Write access; indicates whether writes are allowed to the 512-GByte region controlled by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 512-GByte region controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 512-GByte region controlled by this entry.
7:3	Reserved (must be 0).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 512-GByte region controlled by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	Ignored.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 512-GByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M-1:12	Physical address of 4-KByte aligned EPT page-directory-pointer table referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

Because an EPT PDPTE is identified using bits 47:30 of the guest-physical address, it controls access to a 1-GByte region of the guest-physical-address space. Use of the EPT PDPTE depends on the value of bit 7 in that entry:¹

- If bit 7 of the EPT PDPTE is 1, the EPT PDPTE maps a 1-GByte page. The final physical address is computed as follows:
 - Bits 63:52 are all 0.
 - Bits 51:30 are from the EPT PDPTE.
 - Bits 29:0 are from the original guest-physical address.

The format of an EPT PDPTE that maps a 1-GByte page is given in Table 30-3.

- If bit 7 of the EPT PDPTE is 0, a 4-KByte naturally aligned EPT page directory is located at the physical address specified in bits 51:12 of the EPT PDPTE. The format of an EPT PDPTE that references an EPT page directory is given in Table 30-4.

1. Not all processors allow bit 7 of an EPT PDPTE to be set to 1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether this is allowed.

Table 30-3. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte page referenced by this entry.
1	Write access; indicates whether writes are allowed to the 1-GByte page referenced by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 1-GByte page controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 1-GByte page controlled by this entry.
5:3	EPT memory type for this 1-GByte page (see Section 30.3.7).
6	Ignore PAT memory type for this 1-GByte page (see Section 30.3.7).
7	Must be 1 (otherwise, this entry references an EPT page directory).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 1-GByte page referenced by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 1-GByte page referenced by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 1-GByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
29:12	Reserved (must be 0).
M-1:30	Physical address of the 1-GByte page referenced by this entry. ¹
51:M	Reserved (must be 0).
56:52	Ignored.
57	Verify guest paging. If the “guest-paging verification” VM-execution control is 1, indicates limits on the guest paging structures used to access the 1-GByte page controlled by this entry (see Section 30.3.3.2). If that control is 0, this bit is ignored.
58	Paging-write access. If the “EPT paging-write control” VM-execution control is 1, indicates that guest paging may update the 1-GByte page controlled by this entry (see Section 30.3.3.2). If that control is 0, this bit is ignored.
59	Ignored.
60	Supervisor shadow stack. If bit 7 of EPTP is 1, indicates whether supervisor shadow stack accesses are allowed to guest-physical addresses in the 1-GByte page mapped by this entry (see Section 30.3.3.2). Ignored if bit 7 of EPTP is 0.
62:61	Ignored.
63	Suppress #VE. If the “EPT-violation #VE” VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 27.5.7.1). If “EPT-violation #VE” VM-execution control is 0, this bit is ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

Table 30-4. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that References an EPT Page Directory

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte region controlled by this entry.
1	Write access; indicates whether writes are allowed to the 1-GByte region controlled by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 1-GByte region controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 1-GByte region controlled by this entry.
7:3	Reserved (must be 0).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 1-GByte region controlled by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	Ignored.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 1-GByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M-1:12	Physical address of 4-KByte aligned EPT page directory referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

An EPT page-directory comprises 512 64-bit entries (PDEs). An EPT PDE is selected using the physical address defined as follows:

- Bits 63:52 are all 0.
- Bits 51:12 are from the EPT PDPTE.
- Bits 11:3 are bits 29:21 of the guest-physical address.
- Bits 2:0 are all 0.

Because an EPT PDE is identified using bits 47:21 of the guest-physical address, it controls access to a 2-MByte region of the guest-physical-address space. Use of the EPT PDE depends on the value of bit 7 in that entry:

- If bit 7 of the EPT PDE is 1, the EPT PDE maps a 2-MByte page. The final physical address is computed as follows:
 - Bits 63:52 are all 0.
 - Bits 51:21 are from the EPT PDE.
 - Bits 20:0 are from the original guest-physical address.

The format of an EPT PDE that maps a 2-MByte page is given in Table 30-5.

- If bit 7 of the EPT PDE is 0, a 4-KByte naturally aligned EPT page table is located at the physical address specified in bits 51:12 of the EPT PDE. The format of an EPT PDE that references an EPT page table is given in Table 30-6.

An EPT page table comprises 512 64-bit entries (PTEs). An EPT PTE is selected using a physical address defined as follows:

- Bits 63:52 are all 0.

Table 30-5. Format of an EPT Page-Directory Entry (PDE) that Maps a 2-MByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte page referenced by this entry.
1	Write access; indicates whether writes are allowed to the 2-MByte page referenced by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 2-MByte page controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 2-MByte page controlled by this entry.
5:3	EPT memory type for this 2-MByte page (see Section 30.3.7).
6	Ignore PAT memory type for this 2-MByte page (see Section 30.3.7).
7	Must be 1 (otherwise, this entry references an EPT page table).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 2-MByte page referenced by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 2-MByte page referenced by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 2-MByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
20:12	Reserved (must be 0).
M-1:21	Physical address of the 2-MByte page referenced by this entry. ¹
51:M	Reserved (must be 0).
56:52	Ignored.
57	Verify guest paging. If the “guest-paging verification” VM-execution control is 1, indicates limits on the guest paging structures used to access the 2-MByte page controlled by this entry (see Section 30.3.3.2). If that control is 0, this bit is ignored.
58	Paging-write access. If the “EPT paging-write control” VM-execution control is 1, indicates that guest paging may update the 2-MByte page controlled by this entry (see Section 30.3.3.2). If that control is 0, this bit is ignored.
59	Ignored.
60	Supervisor shadow stack. If bit 7 of EPTP is 1, indicates whether supervisor shadow stack accesses are allowed to guest-physical addresses in the 2-MByte page mapped by this entry (see Section 30.3.3.2). Ignored if bit 7 of EPTP is 0.
62:61	Ignored.
63	Suppress #VE. If the “EPT-violation #VE” VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 27.5.7.1). If “EPT-violation #VE” VM-execution control is 0, this bit is ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

— Bits 51:12 are from the EPT PDE.

- Bits 11:3 are bits 20:12 of the guest-physical address.
- Bits 2:0 are all 0.
- Because an EPT PTE is identified using bits 47:12 of the guest-physical address, every EPT PTE maps a 4-KByte page. The final physical address is computed as follows:
 - Bits 63:52 are all 0.
 - Bits 51:12 are from the EPT PTE.
 - Bits 11:0 are from the original guest-physical address.

The format of an EPT PTE is given in Table 30-7.

An EPT paging-structure entry is **present** if any of bits 2:0 is 1; otherwise, the entry is **not present**. The processor ignores bits 62:3 and uses the entry neither to reference another EPT paging-structure entry nor to produce a physical address. A reference using a guest-physical address whose translation encounters an EPT paging-structure that is not present causes an EPT violation (see Section 30.3.3.2). (If the “EPT-violation #VE” VM-execution control is 1, the EPT violation is convertible to a virtualization exception only if bit 63 is 0; see Section 27.5.7.1. If the “EPT-violation #VE” VM-execution control is 0, this bit is ignored.)

Table 30-6. Format of an EPT Page-Directory Entry (PDE) that References an EPT Page Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte region controlled by this entry.
1	Write access; indicates whether writes are allowed to the 2-MByte region controlled by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 2-MByte region controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 2-MByte region controlled by this entry.
6:3	Reserved (must be 0).
7	Must be 0 (otherwise, this entry maps a 2-MByte page).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 2-MByte region controlled by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	Ignored.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 2-MByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M-1:12	Physical address of 4-KByte aligned EPT page table referenced by this entry. ¹
51:M	Reserved (must be 0).
63:52	Ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

NOTE

If the “mode-based execute control for EPT” VM-execution control is 1, an EPT paging-structure entry is present if any of bits 2:0 **or bit 10** is 1. If bits 2:0 are all 0 but bit 10 is 1, the entry is used normally to reference another EPT paging-structure entry or to produce a physical address.

The discussion above describes how the EPT paging structures reference each other and how the logical processor traverses those structures when translating a guest-physical address. It does not cover all details of the translation process. Additional details are provided as follows:

- Situations in which the translation process may lead to VM exits (sometimes before the process completes) are described in Section 30.3.3.
- Interactions between the EPT translation mechanism and memory typing are described in Section 30.3.7.

Figure 30-1 gives a summary of the formats of the EPTP and the EPT paging-structure entries. For the EPT paging structure entries, it identifies separately the format of entries that map pages, those that reference other EPT paging structures, and those that do neither because they are not present; bits 2:0 and bit 7 are highlighted because they determine how a paging-structure entry is used. (Figure 30-1 does not comprehend the fact that, if the “mode-based execute control for EPT” VM-execution control is 1, an entry is present if any of bits 2:0 or bit 10 is 1.)

30.3.3 EPT-Induced VM Exits

Accesses using guest-physical addresses may cause VM exits due to EPT misconfigurations, EPT violations, and page-modification log-full events. An **EPT misconfiguration** occurs when, in the course of translating a guest-physical address, the logical processor encounters an EPT paging-structure entry that contains an unsupported value (see Section 30.3.3.1). An **EPT violation** occurs when there is no EPT misconfiguration but the EPT paging-structure entries disallow an access using the guest-physical address (see Section 30.3.3.2). A **page-modification log-full event** occurs when the logical processor determines a need to create a page-modification log entry and the current log is full (see Section 30.3.6).

These events occur only due to an attempt to access memory with a guest-physical address. Loading CR3 with a guest-physical address with the MOV to CR3 instruction can cause neither an EPT configuration nor an EPT violation until that address is used to access a paging structure.¹

If the “EPT-violation #VE” VM-execution control is 1, certain EPT violations may cause virtualization exceptions instead of VM exits. See Section 27.5.7.1.

30.3.3.1 EPT Misconfigurations

An EPT misconfiguration occurs if translation of a guest-physical address encounters an EPT paging-structure entry that meets any of the following conditions:

- Bit 0 of the entry is clear (indicating that data reads are not allowed) and any of the following hold:
 - Bit 1 is set (indicating that data writes are allowed).
 - The processor does not support execute-only translations and either of the following hold:
 - Bit 2 is set (indicating that instruction fetches are allowed).²
 - The “mode-based execute control for EPT” VM-execution control is 1 and bit 10 is set (indicating that instruction fetches are allowed from user-mode linear addresses).
- Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP to determine whether execute-only translations are supported (see Appendix A.10).
- The “EPT paging-write control” VM-execution control is 1, the entry maps a page, and bit 58 is set (indicating that paging writes are allowed).
- The entry is present (see Section 30.3.2) and of the following holds:

1. If the logical processor is using PAE paging—because CR0.PG = CR4.PAE = 1 and IA32_EFER.LMA = 0—the MOV to CR3 instruction loads the PDPTs from memory using the guest-physical address being loaded into CR3. In this case, therefore, the MOV to CR3 instruction may cause an EPT misconfiguration, an EPT violation, or a page-modification log-full event.

2. If the “mode-based execute control for EPT” VM-execution control is 1, setting bit 2 indicates that instruction fetches are allowed from supervisor-mode linear addresses.

Table 30-7. Format of an EPT Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 4-KByte page referenced by this entry.
1	Write access; indicates whether writes are allowed to the 4-KByte page referenced by this entry.
2	If the “mode-based execute control for EPT” VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 4-KByte page controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 4-KByte page controlled by this entry.
5:3	EPT memory type for this 4-KByte page (see Section 30.3.7).
6	Ignore PAT memory type for this 4-KByte page (see Section 30.3.7).
7	Ignored.
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 30.3.5). Ignored if bit 6 of EPTP is 0.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 4-KByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M-1:12	Physical address of the 4-KByte page referenced by this entry. ¹
51:M	Reserved (must be 0).
56:52	Ignored.
57	Verify guest paging. If the “guest-paging verification” VM-execution control is 1, indicates limits on the guest paging structures used to access the 4-KByte page controlled by this entry (see Section 30.3.3.2). If that control is 0, this bit is ignored.
58	Paging-write access. If the “EPT paging-write control” VM-execution control is 1, indicates that guest paging may update the 4-KByte page controlled by this entry (see Section 30.3.3.2). If that control is 0, this bit is ignored.
59	Ignored.
60	Supervisor shadow stack. If bit 7 of EPTP is 1, indicates whether supervisor shadow stack accesses are allowed to guest-physical addresses in the 4-KByte page mapped by this entry (see Section 30.3.3.2). Ignored if bit 7 of EPTP is 0.
61	Sub-page write permissions. If the “sub-page write permissions for EPT” VM-execution control is 1, writes to individual 128-byte regions of the 4-KByte page referenced by this entry may be allowed even if the page would normally not be writable (see Section 30.3.4). If “sub-page write permissions for EPT” VM-execution control is 0, this bit is ignored.
62	Ignored.
63	Suppress #VE. If the “EPT-violation #VE” VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 27.5.7.1). If “EPT-violation #VE” VM-execution control is 0, this bit is ignored.

NOTES:

1. M is an abbreviation for MAXPHYADDR. See Section 30.3.1.

- A reserved bit is set. This includes the setting of a bit in the range 51:12 in position MAXPHYADDR or above (see Section 30.3.1). See Section 30.3.2 for details of which bits are reserved in which EPT paging-structure entries.
- The entry is the last one used to translate a guest physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE) and the value of bits 5:3 (EPT memory type) is 2, 3, or 7 (these values are reserved).

EPT misconfigurations result when an EPT paging-structure entry is configured with settings reserved for future functionality. Software developers should be aware that such settings may be used in the future and that an EPT paging-structure entry that causes an EPT misconfiguration on one processor might not do so in the future.

30.3.3.2 EPT Violations

An EPT violation may occur during an access using a guest-physical address whose translation does not cause an EPT misconfiguration. An EPT violation occurs in any of the following situations:

- Translation of the guest-physical address encounters an EPT paging-structure entry that is not present (see Section 30.3.2).
- The access is a data read and, for any byte to be read, bit 0 (read access) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte. Reads by the logical processor of guest paging structures to translate a linear address are considered to be data reads.

6	6	6	6	5	5	5	5	5	5	5		M ¹	M-1			3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	
Reserved													Address of EPT PML5 table or PML4 table													Rsvd.		S	A	EPT	EPT		EPTP ³													
Ignored													Rsvd.		Address of EPT PML4 table													Ig	X	Ig	A	Reserved		X	W	R										
S	Ignored																																	0		0		0		PML5E: not present						
V	Ignored													Rsvd.		Address of EPT page-directory-pointer table													Ig	X	Ig	A	Reserved		X	W	R									
E	Ignored																																	0		0		0		PML4E: not present						
E	Ignored													Rsvd.		Address of EPT page-directory-pointer table													Ig	X	Ig	A	Reserved		X	W	R									
S	Ignored																																	0		0		0		PML4E: not present						
V	Ign.	S	Ig	P	V	Ignored					Rsvd.		Physical address of 1GB page					Reserved										Ig	X	D	A	1	P	EPT	X		W	R								
E		S	n.	W	G								address of 1GB page															n.	U			A	A	MT	X		W	R								
Ignored													Rsvd.		Address of EPT page directory													Ig	X	Ig	A	0	Rsvd.		X	W	R	PDPTE: page directory								
S	Ignored																																	0		0			0		PDPTE: not present					
V	Ignored													Rsvd.		Physical address of 2MB page										Reserved										Ig	X	D	A	1		P	EPT	X		W
E		S	n.	W	G								address of 2MB page																				n.	U			A	A	MT	X		W	R			
Ignored													Rsvd.		Address of EPT page table													Ig	X	Ig	A	0	Rsvd.		X	W	R	PDE: page table								
S	Ignored																																	0		0			0		PDE: not present					
V	Ign.	S	Ig	P	V	Ignored					Rsvd.		Physical address of 4KB page										Ig	X	D	A	Ig	P	EPT	X		W	R													
E		S	n.	W	G								address of 4KB page										n.	U			A	n.	A	MT	X		W	R												
S	Ignored																																	0		0		0		PTE: not present						
V	Ignored													Rsvd.		Physical address of 4KB page													Ig	X	D	A	Ig	P	EPT	X		W	R							
E	Ignored																																	0		0		0		PTE: not present						

Figure 30-1. Formats of EPTP and EPT Paging-Structure Entries

NOTES:

1. **M** is an abbreviation for MAXPHYADDR. See Section 30.3.1.
2. Supervisor shadow-stack control.
3. See Section 26.6.11 for details of the EPTP.
4. Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 0, this bit is ignored.
5. Execute access. If the “mode-based execute control for EPT” VM-execution control is 1, this bit controls execute access for supervisor-mode linear addresses.

6. If the “mode-based execute control for EPT” VM-execution control is 1, an EPT paging-structure entry is present if any of bits 2:0 or bit 10 is 1. This table does not comprehend that fact.
7. Suppress #VE. If the “EPT-violation #VE” VM-execution control is 0, this bit is ignored.
8. Supervisor shadow-stack page. If bit 7 of the EPTP is 0, this bit is ignored.
9. Paging-write access. If the “EPT paging-write control” VM-execution control is 0, this bit is ignored.
10. Verify guest paging. If the “guest-paging verification” VM-execution control is 0, this bit is ignored.
11. Sub-page write permissions. If the “sub-page write permissions for EPT” VM-execution control is 0, this bit is ignored.

- The access is a data write and, for any byte to be written, bit 1 (write access) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte. Writes by the logical processor to guest paging structures to update accessed and dirty flags are considered to be data writes.

If bit 6 of the EPT pointer (EPTP) is 1 (enabling accessed and dirty flags for EPT), processor accesses to guest paging-structure entries are treated as writes with regard to EPT violations. Thus, if bit 1 is clear in any of the EPT paging-structure entries used to translate the guest-physical address of a guest paging-structure entry, an attempt to use that entry to translate a linear address causes an EPT violation.

(This does not apply to loads of the PDPT registers by the MOV to CR instruction for PAE paging; see Section 5.4.1. Those loads of guest PDPTes are treated as reads and do not cause EPT violations due to a guest-physical address not being writable.)

Processor writes to guest paging structures to update accessed and dirty flags are called **paging writes**.

(When accessed and dirty flags for EPT are enabled all processor accesses to guest paging structures are considered paging writes.) If the “EPT paging-write control” VM-execution control is 1, clearing the write-access bit in the EPT paging-structure entry that maps a page does not prevent a paging write if bit 58 (paging-write access) in that entry is 1. (EPT paging-structure entries that reference other EPT paging structures do not use bit 58, and it remains the case that they must set the write-access bit for paging writes to be allowed.)

If the “sub-page write permissions for EPT” VM-execution control is 1, data writes to a guest-physical address that would cause an EPT violation (as indicated above) do not do so in certain situations. If the guest-physical address is mapped using a 4-KByte page and bit 61 (sub-page write permissions) of the EPT PTE used to map the page is 1, writes to certain 128-byte sub-pages may be allowed. See Section 30.3.4 for details.

- The access is an instruction fetch and the EPT paging structures prevent execute access to any of the bytes being fetched. Whether this occurs depends upon the setting of the “mode-based execute control for EPT” VM-execution control:
 - If the control is 0, an instruction fetch from a byte is prevented if bit 2 (execute access) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte.
 - If the control is 1, an instruction fetch from a byte is prevented in either of the following cases:
 - Paging maps the linear address of the byte as a supervisor-mode address and bit 2 (execute access for supervisor-mode linear addresses) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte.
Paging maps a linear address as a supervisor-mode address if the U/S flag (bit 2) is 0 in at least one of the paging-structure entries controlling the translation of the linear address.
 - Paging maps the linear address of the byte as a user-mode address and bit 10 (execute access for user-mode linear addresses) was clear in any of the EPT paging-structure entries used to translate the guest-physical address of the byte.
Paging maps a linear address as a user-mode address if the U/S flag is 1 in all of the paging-structure entries controlling the translation of the linear address. If paging is disabled (CR0.PG = 0), every linear address is a user-mode address.
- If supervisor shadow-stack control is enabled (by setting bit 7 of EPTP), the access is a supervisor shadow-stack access, and the EPT paging-structure entries used to translate the guest-physical address of the access disallow supervisor shadow-stack accesses. Such an access is disallowed if any of the following hold:
 - Bit 0 (read access) is clear in any EPT paging-structure entry used to translate the guest-physical address of the access.

- Bit 1 (write access) is clear in any EPT paging-structure entry that references an EPT paging structure in the translation of the guest-physical address. (Clearing bit 1 in the EPT paging-structure entry that maps the page of the guest-physical address does not disallow shadow-stack reads and writes.)
- Bit 60 (supervisor-shadow stack access) is clear in the EPT paging-structure entry that maps the page of the guest-physical address.

Supervisor shadow-stack control and the supervisor-shadow stack access bits in EPT paging-structure entries do not affect other accesses (including user shadow-stack accesses).

- If the “guest-paging verification” and “EPT paging-write control” VM-execution controls are both 1, **guest-paging verification** is performed for certain accesses related to translation of a linear address.

Specifically, guest-paging verification may apply to an access using a guest-physical address to the guest paging-structure entry that maps a page for the linear address (see Chapter 5, “Paging”). It applies if bit 57 (verify guest paging) is set in the EPT paging-structure entry that maps a page for that guest-physical address.

When guest-paging verification occurs, there is an EPT violation if bit 58 (paging-write access) is clear in the EPT paging-structure entry that was used to map a page for the guest-physical address of any guest paging-structure entry that was used during translation of the original linear address.¹

See Section 30.3.3.3 for details regarding the prioritization of such EPT violations relative to any page faults that may occur due to the original reference to the linear address being translated.

30.3.3.3 Prioritization of EPT Misconfigurations and EPT Violations

The translation of a linear address to a physical address requires one or more translations of guest-physical addresses using EPT (see Section 30.3.1). This section specifies the relative priority of EPT-induced VM exits with respect to each other and to other events that may be encountered when accessing memory using a linear address.

For an access to a guest-physical address, determination of whether an EPT misconfiguration or an EPT violation occurs is based on an iterative process:²

1. An EPT paging-structure entry is read (initially, this is an EPT PML5 entry or an EPT PML4 entry):
 - a. If the entry is not present (see Section 30.3.2), an EPT violation occurs.
 - b. If the entry is present but its contents are not configured properly (see Section 30.3.3.1), an EPT misconfiguration occurs.
 - c. If the entry is present and its contents are configured properly, operation depends on whether the entry references another EPT paging structure (or if instead it is an EPT PDPTE or EPT PDE with bit 7 set to 1, or an EPT PTE):
 - i) If the entry does reference another EPT paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
 - ii) Otherwise, the entry is used to produce the ultimate physical address (the translation of the original guest-physical address); step 2 is executed.
2. Once the ultimate physical address is determined, the privileges determined by the EPT paging-structure entries are evaluated:
 - a. If the access to the guest-physical address is not allowed by these privileges (see Section 30.3.3.2), an EPT violation occurs.
 - b. If the access to the guest-physical address is allowed by these privileges, memory is accessed using the ultimate physical address.

If CR0.PG = 1, the translation of a linear address is also an iterative process, with the processor first accessing an entry in the guest paging structure referenced by the guest-physical address in CR3,³ then accessing an entry in

-
1. When there is a restart of HLAT paging (see Section 5.5), the guest paging-structure entries used by HLAT paging (prior to the restart) are not relevant to guest-paging verification.
 2. This is a simplification of the more detailed description given in Section 30.3.2.
 3. If PAE paging is in use, the processor instead uses the guest-physical address in the appropriate PDPTE register. If HLAT paging is enabled, the processor instead uses the HLATP VMCS field. For simplicity, the remainder of this section refers only to CR3.

another guest paging structure referenced by the guest-physical address in the first guest paging-structure entry, etc. Each guest-physical address is itself translated using EPT and may cause an EPT-induced VM exit. The following items detail how page faults and EPT-induced VM exits are recognized during this iterative process:

1. An attempt is made to access a guest paging-structure entry with a guest-physical address (initially, the address in CR3).
 - a. If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs.
 - b. If the access does not cause an EPT-induced VM exit, the translation continues. If guest-paging verification is enabled (see Section 30.3.3.2), the processor notes whether the EPT paging-structure entry used to map a page for the guest-physical address set bit 58 (paging write). Then, bit 0 (the present flag) of the guest paging-structure entry is consulted:
 - i) If the present flag is 0 or any reserved bit is set, a page fault occurs.
 - ii) If the present flag is 1, no reserved bit is set, operation depends on whether the entry references another guest paging structure (whether it is a guest PDE with PS = 1 or a guest PTE):
 - If the entry does reference another guest paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
 - Otherwise, the entry is used to produce the ultimate guest-physical address (the translation of the original linear address); step 2 is executed.
2. Once the ultimate guest-physical address is determined, the privileges determined by the guest paging-structure entries are evaluated:
 - a. If the access to the linear address is not allowed by these privileges (e.g., it was a write to a read-only page), a page fault occurs.
 - b. If the access to the linear address is allowed by these privileges, an attempt is made to access memory at the ultimate guest-physical address:
 - i) If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs. It is at this point that guest-paging verification occurs (if enabled), using the paging-write bits that were noted at occurrences of step 1b above (see Section 30.3.3.2 for details of guest-paging verification).
 - ii) If the access does not cause an EPT-induced VM exit, memory is accessed using the ultimate physical address (the translation, using EPT, of the ultimate guest-physical address).

If CR0.PG = 0, a linear address is treated as a guest-physical address and is translated using EPT (see above). This process, if it completes without an EPT violation or EPT misconfiguration, produces a physical address and determines the privileges allowed by the EPT paging-structure entries. If these privileges do not allow the access to the physical address (see Section 30.3.3.2), an EPT violation occurs. Otherwise, memory is accessed using the physical address.

30.3.4 Sub-Page Write Permissions

Section 30.3.3.2 explained how EPT enforces the access rights for guest-physical addresses using EPT violations. Since these access rights are determined using the EPT paging-structure entries that are used to translate a guest-physical address, their granularity is limited to that which is used to map pages (1-GByte, 2-MByte, or 4-KByte).

The **sub-page write-permission** feature allows the control of write accesses to guest-physical addresses to be controlled at finer granularity. Sub-page write permissions allow write accesses to be controlled at the granularity of naturally aligned 128-byte **sub-pages**. Specifically, the feature allows writes to selected sub-pages of 4-KByte page that would otherwise not be writable.

Sub-page write permissions are enabled setting the “sub-page write permissions for EPT” VM-execution control to 1. The remainder of this section describes changes to processor operation with this control setting.

Section 30.3.4.1 identifies the data accesses that are eligible for sub-page write permissions. Section 30.3.4.2 explains how the processor determines whether to allow such an access.

30.3.4.1 Write Accesses That Are Eligible for Sub-Page Write Permissions

A guest-physical address is eligible for sub-page write permissions if writes to it would be disallowed following Section 30.3.3.2: bit 1 (write access) is clear in any of the EPT paging-structure entries used to translate the guest-physical address. Guest-physical addresses to which writes would be disallowed for other reasons (e.g., the translation encounters an EPT paging-structure entry that is not present) are not eligible for sub-page write permissions.

In addition, a guest-physical address is eligible for sub-page write permissions only if it is mapped using a 4-KByte page and bit 61 (sub-page write permissions) of the EPT PTE used to map the page is 1. (Guest-physical addresses mapped with larger pages are not eligible for sub-page write permissions.)

For some memory accesses, the processor ignores bit 61 in an EPT PTE used to map a 4-KByte page and does not apply sub-page write permissions to the access. (In such a case, the access causes an EPT violation when indicated by the conditions given in Section 30.3.3.2.) Sub-page write permissions never apply to the following accesses:

- A write access performed within a transactional region.
- A write access by an enclave to an address within the enclave's ELRANGE. (Sub-page write permissions may apply to write accesses by an enclaves to addresses outside its ELRANGE.)
- A write access to the enclave page cache (EPC) by an Intel SGX instruction.
- A write access to a guest paging structure to update an accessed or dirty flag.
- Processor accesses to guest paging-structure entries when accessed and dirty flags for EPT are enabled (such accesses are treated as writes with regard to EPT violations).

There are additional accesses to which sub-page write permissions might not be applied (behavior is model-specific). The following items enumerate examples:

- A write access that crosses two 4-KByte pages. In this case, sub-page permissions may be applied to neither or to only one of the pages. (There is no write to either page unless the write is allowed to both pages.)
- A write access by an instruction that performs multiple write accesses (sub-page write permissions are intended principally for basic instructions such as AND, MOV, OR, TEST, XCHG, and XOR).

If a guest-physical address is eligible for sub-page write permissions, the processor determines whether to allow write to the address using the process described in Section 30.3.4.2.

If a guest-physical address is eligible for sub-page write permissions and that address translates to an address on the APIC-access page (see Section 31.4), the processor may treat a write access to the address as if the “virtualize APIC accesses” VM-execution control were 0. For that reason, it is recommended that software not configure any guest-physical address that translates to an address on the APIC-access page to be eligible for sub-page write permissions.

30.3.4.2 Determining an Access's Sub-Page Write Permission

Sub-page write permissions control write accesses individually to each of the 32 128-byte sub-pages of a 4-KByte page. Bits 11:7 of guest-physical address identify the sub-page.

For each guest-physical address eligible for sub-page write permissions, there is a 64-bit **sub-page permission vector (SPP vector)**. All addresses on a 4-KByte page use the same SPP vector. If an address's sub-page number (bits 11:7 of the address) is *S*, writes to address are allowed if and only if bit 2*S* of the sub-page permission is set to 1. (The bits at odd positions in a SPP vector are not used and must be zero.)

Each page's SPP vector is located in memory. For a write to a guest-physical address eligible for sub-page write permissions, the processor uses the following process to locate the address's SPP vector:

1. The SPPTP (sub-page-permission-table pointer) VM-execution control field contains the physical address of the 4-KByte root SPP table (SSPL4 table). Bits 47:39 of the guest-physical address identify a 64-bit entry in that table, called an SPPL4E.
2. A 4-KByte SPPL3 table is located at the physical address in the selected SPPL4E. Bits 38:30 of the guest-physical address identify a 64-bit entry in that table, called an SPPL3E.
3. A 4-KByte SPPL2 table is located at the physical address in the selected SPPL3E. Bits 29:21 of the guest-physical address identify a 64-bit entry in that table, called an SPPL2E.

4. A 4-KByte SPP-vector table (SSPL1 table) is located at the physical address in the selected SPPL2E. Bits 20:12 of the guest-physical address identify the 64-bit SPP vector for the address. As noted earlier, bit 2S of the sub-page permission vector determines whether the address may be written, where S is the value of address bits 11:7.

(The memory type used to access these tables is reported in bits 53:50 of the IA32_VMX_BASIC MSR. See Appendix A.1.)

A write access to multiple 128-byte sub-pages on a single 4-KByte page is allowed only if the indicated bit in the page's SPP vector for each of those sub-pages is set to 1. The following items apply to cases in which an access writes to two 4-KByte pages:

- If a write to either page would be disallowed according to Section 30.3.3.2, the access might be disallowed even if the guest-physical address of that page is eligible for sub-page write permissions. (This behavior is model-specific.)
- The access is allowed only if, for each page, either (1) a write to the page would be allowed following Section 30.3.3.2; or (2) both (a) the guest-physical address of that page is eligible for sub-page write permissions; and (b) the page's sub-page vector allows the write (as described above).

Bit 0 of each entry (SPPL4E, SPPL3E, or SPPL2E) is the entry's **valid** bit. If the process above accesses an entry in which this bit is 0, the process stops and the logical processor incurs an **SPP miss**.

In each entry (SPPL4E, SPPL3E, or SPPL2E), bits 11:1 are reserved, as are bits 63:MAXPHYADDR (see Section 30.3.1). If the process above accesses an entry in which the valid bit is 1 and in which some reserved bit is set, the process stops and the logical processor incurs an **SPP misconfiguration**. Bits in an SPP vector in odd positions are also reserved; an SPP misconfiguration occurs also any of those bits are set in the final SPP vector.

SPP misses and SPP misconfigurations are called **SPP-related events** and cause VM exits.

30.3.5 Accessed and Dirty Flags for EPT

The Intel 64 architecture supports **accessed and dirty flags** in ordinary paging-structure entries (see Section 5.8). Some processors also support corresponding flags in EPT paging-structure entries. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether the processor supports this feature.

Software can enable accessed and dirty flags for EPT using bit 6 of the extended-page-table pointer (EPTP), a VM-execution control field (see Table 26-9 in Section 26.6.11). If this bit is 1, the processor will set the accessed and dirty flags for EPT as described below. In addition, setting this flag causes processor accesses to guest paging-structure entries to be treated as writes (see below and Section 30.3.3.2).

For any EPT paging-structure entry that is used during guest-physical-address translation, bit 8 is the accessed flag. For a EPT paging-structure entry that maps a page (as opposed to referencing another EPT paging structure), bit 9 is the dirty flag.

Whenever the processor uses an EPT paging-structure entry as part of guest-physical-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a guest-physical address, the processor sets the dirty flag (if it is not already set) in the EPT paging-structure entry that identifies the final physical address for the guest-physical address (either an EPT PTE or an EPT paging-structure entry in which bit 7 is 1).

When accessed and dirty flags for EPT are enabled, processor accesses to guest paging-structure entries are treated as writes (see Section 30.3.3.2). Thus, such an access will cause the processor to set the dirty flag in the EPT paging-structure entry that identifies the final physical address of the guest paging-structure entry.

(This does not apply to loads of the PDPTE registers for PAE paging by the MOV to CR instruction; see Section 5.4.1. Those loads of guest PDPTEs are treated as reads and do not cause the processor to set the dirty flag in any EPT paging-structure entry.)

These flags are "sticky," meaning that, once set, the processor does not clear them; only software can clear them.

A processor may cache information from the EPT paging-structure entries in TLBs and paging-structure caches (see Section 30.4). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the processor might not set the corresponding bit in memory on a subsequent access using an affected guest-physical address.

30.3.6 Page-Modification Logging

When accessed and dirty flags for EPT are enabled, software can track writes to guest-physical addresses using a feature called **page-modification logging**.

Software can enable page-modification logging by setting the “enable PML” VM-execution control (see Table 26-7 in Section 26.6.2). When this control is 1, the processor adds entries to the **page-modification log** as described below. The page-modification log is a 4-KByte region of memory located at the physical address in the PML address VM-execution control field. The page-modification log consists of 512 64-bit entries; the PML index VM-execution control field indicates the next entry to use.

Before allowing a guest-physical access, the processor may determine that it first needs to set an accessed or dirty flag for EPT (see Section 30.3.5). When this happens, the processor examines the PML index. If the PML index is not in the range 0–511, there is a **page-modification log-full event** and a VM exit occurs. In this case, the accessed or dirty flag is not set, and the guest-physical access that triggered the event does not occur.

If instead the PML index is in the range 0–511, the processor proceeds to update accessed or dirty flags for EPT as described in Section 30.3.5. If the processor updated a dirty flag for EPT (changing it from 0 to 1), it then operates as follows:

1. The guest-physical address of the access is written to the page-modification log. Specifically, the guest-physical address is written to physical address determined by adding 8 times the PML index to the PML address. Bits 11:0 of the value written are always 0 (the guest-physical address written is thus 4-KByte aligned).
2. The PML index is decremented by 1 (this may cause the value to transition from 0 to FFFFH).

Because the processor decrements the PML index with each log entry, the value may transition from 0 to FFFFH. At that point, no further logging will occur, as the processor will determine that the PML index is not in the range 0–511 and will generate a page-modification log-full event (see above).

30.3.7 EPT and Memory Typing

This section specifies how a logical processor determines the memory type use for a memory access while EPT is in use. (See Chapter 13, “Memory Cache Control,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for details of memory typing in the Intel 64 architecture.) Section 30.3.7.1 explains how the memory type is determined for accesses to the EPT paging structures. Section 30.3.7.2 explains how the memory type is determined for an access using a guest-physical address that is translated using EPT.

30.3.7.1 Memory Type Used for Accessing EPT Paging Structures

This section explains how the memory type is determined for accesses to the EPT paging structures. The determination is based first on the value of bit 30 (cache disable—CD) in control register CR0:

- If CR0.CD = 0, the memory type used for any such reference is the EPT paging-structure memory type, which is specified in bits 2:0 of the extended-page-table pointer (EPTP), a VM-execution control field (see Section 26.6.11). A value of 0 indicates the uncacheable type (UC), while a value of 6 indicates the write-back type (WB). Other values are reserved.
- If CR0.CD = 1, the memory type used for any such reference is uncacheable (UC).

The MTRRs have no effect on the memory type used for an access to an EPT paging structure.

30.3.7.2 Memory Type Used for Translated Guest-Physical Addresses

The **effective memory type** of a memory access using a guest-physical address (an access that is translated using EPT) is the memory type that is used to access memory. The effective memory type is based on the value of bit 30 (cache disable—CD) in control register CR0; the **last** EPT paging-structure entry used to translate the guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE); and the PAT memory type (see below):

- The **PAT memory type** depends on the value of CR0.PG:
 - If CR0.PG = 0, the PAT memory type is WB (writeback).¹

- If CR0.PG = 1, the PAT memory type is the memory type selected from the IA32_PAT MSR as specified in Section 13.12.3, “Selecting a Memory Type from the PAT.”¹
- The **EPT memory type** is specified in bits 5:3 of the last EPT paging-structure entry: 0 = UC; 1 = WC; 4 = WT; 5 = WP; and 6 = WB. Other values are reserved and cause EPT misconfigurations (see Section 30.3.3).
- If CR0.CD = 0, the effective memory type depends upon the value of bit 6 of the last EPT paging-structure entry:
 - If the value is 0, the effective memory type is the combination of the EPT memory type and the PAT memory type specified in Table 13-7 in Section 13.5.2.2, using the EPT memory type in place of the MTRR memory type.
 - If the value is 1, the memory type used for the access is the EPT memory type. The PAT memory type is ignored.
- If CR0.CD = 1, the effective memory type is UC.

The MTRRs have no effect on the memory type used for an access to a guest-physical address.

30.4 CACHING TRANSLATION INFORMATION

Processors supporting Intel® 64 and IA-32 architectures may accelerate the address-translation process by caching on the processor data from the structures in memory that control that process. Such caching is discussed in Section 5.10, “Caching Translation Information,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. The current section describes how this caching interacts with the VMX architecture.

The VPID and EPT features of the architecture for VMX operation augment this caching architecture. EPT defines the guest-physical address space and defines translations to that address space (from the linear-address space) and from that address space (to the physical-address space). Both features control the ways in which a logical processor may create and use information cached from the paging structures.

Section 30.4.1 describes the different kinds of information that may be cached. Section 30.4.2 specifies when such information may be cached and how it may be used. Section 30.4.3 details how software can invalidate cached information.

30.4.1 Information That May Be Cached

Section 5.10, “Caching Translation Information,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, identifies two kinds of translation-related information that may be cached by a logical processor: **translations**, which are mappings from linear page numbers to physical page frames, and **paging-structure caches**, which map the upper bits of a linear page number to information from the paging-structure entries used to translate linear addresses matching those upper bits.

The same kinds of information may be cached when VPIDs and EPT are in use. A logical processor may cache and use such information based on its function. Information with different functionality is identified as follows:

- **Linear mappings.**² There are two kinds:
 - Linear translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

1. Table 13-11 in Section 13.12.3, “Selecting a Memory Type from the PAT,” illustrates how the PAT memory type is selected based on the values of the PAT, PCD, and PWT bits in a page-table entry (or page-directory entry with PS = 1). For accesses to a guest paging-structure entry X, the PAT memory type is selected from the table by using a value of 0 for the PAT bit with the values of PCD and PWT from the paging-structure entry Y that references X (or from CR3 if X is in the root paging structure). With PAE paging, the PAT memory type for accesses to the PDPTes is WB.

2. Earlier versions of this manual used the term “VPID-tagged” to identify linear mappings.

- Linear paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges. For example, bits 47:39 of a linear address would map to the address of the relevant page-directory-pointer table.

Linear mappings do not contain information from any EPT paging structure.

- **Guest-physical mappings.**¹ There are two kinds:

- Guest-physical translations. Each of these is a mapping from a guest-physical page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
- Guest-physical paging-structure-cache entries. Each of these is a mapping from the upper portion of a guest-physical address to the physical address of the EPT paging structure used to translate the corresponding region of the guest-physical address space, along with information about access privileges.

The information in guest-physical mappings about access privileges and memory typing is derived from EPT paging structures.

- **Combined mappings.**² There are two kinds:

- Combined translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
- Combined paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges.

The information in combined mappings about access privileges and memory typing is derived from both guest paging structures and EPT paging structures.

Guest-physical mappings and combined mappings may also include SPP vectors and information about the data structures used to locate SPP vectors (see Section 30.3.4.2).

30.4.2 Creating and Using Cached Translation Information

The following items detail the creation of the mappings described in the previous section:³

- The following items describe the creation of mappings while EPT is not in use (including execution outside VMX non-root operation):
 - Linear mappings may be created. They are derived from the paging structures referenced (directly or indirectly) by the current value of CR3 and are associated with the current VPID and the current PCID.
 - No linear mappings are created with information derived from paging-structure entries that are not present (bit 0 is 0) or that set reserved bits. For example, if a PTE is not present, no linear mapping are created for any linear page number whose translation would use that PTE.
 - No guest-physical or combined mappings are created while EPT is not in use.
- The following items describe the creation of mappings while EPT is in use:
 - Guest-physical mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by bits 51:12 of the current EPTP. These 40 bits contain the address of the EPT root table (a EPT PML4 table with 4-level EPT or a EPT PML5 table with 5-level EPT); the notation **EPTRTA** refers to those 40 bits. Newly created guest-physical mappings are associated with the current EPTRTA.
 - Combined mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by the current EPTRTA. If CR0.PG = 1, they are also derived from the paging structures referenced (directly or indirectly) by the current value of CR3. They are associated with the current VPID,

1. Earlier versions of this manual used the term “EPTP-tagged” to identify guest-physical mappings.

2. Earlier versions of this manual used the term “dual-tagged” to identify combined mappings.

3. This section associated cached information with the current VPID and PCID. If PCIDs are not supported or are not being used (e.g., because CR4.PCIDE = 0), all the information is implicitly associated with PCID 000H; see Section 5.10.1, “Process-Context Identifiers (PCIDs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

the current PCID, and the current EPTRTA.¹ No combined paging-structure-cache entries are created if CR0.PG = 0.²

- No guest-physical mappings or combined mappings are created with information derived from EPT paging-structure entries that are not present (see Section 30.3.2) or that are misconfigured (see Section 30.3.3.1).
- No combined mappings are created with information derived from guest paging-structure entries that are not present or that set reserved bits.
- No linear mappings are created while EPT is in use.

The following items detail the use of the various mappings:

- If EPT is not in use (e.g., when outside VMX non-root operation), a logical processor may use cached mappings as follows:
 - For accesses using linear addresses, it may use linear mappings associated with the current VPID and the current PCID. It may also use global TLB entries (linear mappings) associated with the current VPID and any PCID.
 - No guest-physical or combined mappings are used while EPT is not in use.
- If EPT is in use, a logical processor may use cached mappings as follows:
 - For accesses using linear addresses, it may use combined mappings associated with the current VPID, the current PCID, and the current EPTRTA. It may also use global TLB entries (combined mappings) associated with the current VPID, the current EPTRTA, and any PCID.
 - For accesses using guest-physical addresses, it may use guest-physical mappings associated with the current EPTRTA.
 - No linear mappings are used while EPT is in use.

30.4.3 Invalidating Cached Translation Information

Software modifications of paging structures (including EPT paging structures and the data structures used to locate SPP vectors) may result in inconsistencies between those structures and the mappings cached by a logical processor. Certain operations invalidate information cached by a logical processor and can be used to eliminate such inconsistencies.

30.4.3.1 Operations that Invalidate Cached Mappings

The following operations invalidate cached mappings as indicated:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation (e.g., the INVLPG and INVPCID instructions) invalidate linear mappings and combined mappings.³ They are required to do so only for the current VPID (but, for combined mappings, all EPTRTAs). Linear mappings for the current VPID are invalidated even if EPT is in use.⁴ Combined mappings for the current VPID are invalidated even if EPT is not in use.⁵

1. At any given time, a logical processor may be caching combined mappings for a VPID and a PCID that are associated with different EPTRTAs. Similarly, it may be caching combined mappings for an EPTRTA that are associated with different VPIDs and PCIDs.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
3. See Section 5.10.4, “Invalidation of TLBs and Paging-Structure Caches,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for an enumeration of operations that architecturally invalidate entries in the TLBs and paging-structure caches independent of VMX operation.
4. While no linear mappings are created while EPT is in use, a logical processor may retain, while EPT is in use, linear mappings (for the same VPID as the current one) there were created earlier, when EPT was not in use.
5. While no combined mappings are created while EPT is not in use, a logical processor may retain, while EPT is in not use, combined mappings (for the same VPID as the current one) there were created earlier, when EPT was in use.

- An EPT violation invalidates any guest-physical mappings (associated with the current EPTRTA) that would be used to translate the guest-physical address that caused the EPT violation. If that guest-physical address was the translation of a linear address, the EPT violation also invalidates any combined mappings for that linear address associated with the current PCID, the current VPID and the current EPTRTA.
- If the “enable VPID” VM-execution control is 0, VM entries and VM exits invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs). Combined mappings for VPID 0000H are invalidated for all EPTRTAs.
- Execution of the INVVPID instruction invalidates linear mappings and combined mappings. Invalidation is based on instruction operands, called the INVVPID type and the INVVPID descriptor. Four INVVPID types are currently defined:
 - **Individual-address.** If the INVVPID type is 0, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor and that would be used to translate the linear address specified in of the INVVPID descriptor. Linear mappings and combined mappings for that VPID and linear address are invalidated for all PCIDs and, for combined mappings, all EPTRTAs. (The instruction may also invalidate mappings associated with other VPIDs and for other linear addresses.)
 - **Single-context.** If the INVVPID type is 1, the logical processor invalidates all linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EPTRTAs. (The instruction may also invalidate mappings associated with other VPIDs.)
 - **All-context.** If the INVVPID type is 2, the logical processor invalidates linear mappings and combined mappings associated with all VPIDs except VPID 0000H and with all PCIDs. (The instruction may also invalidate linear mappings with VPID 0000H.) Combined mappings are invalidated for all EPTRTAs.
 - **Single-context-retaining-globals.** If the INVVPID type is 3, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EPTRTAs. The logical processor is **not** required to invalidate information that was used for **global** translations (although it may do so). See Section 5.10, “Caching Translation Information,” for details regarding global translations. (The instruction may also invalidate mappings associated with other VPIDs.)

See Chapter 32 for details of the INVVPID instruction. See Section 30.4.3.3 for guidelines regarding use of this instruction.

- Execution of the INVEPT instruction invalidates guest-physical mappings and combined mappings. Invalidation is based on instruction operands, called the INVEPT type and the INVEPT descriptor. Two INVEPT types are currently defined:
 - **Single-context.** If the INVEPT type is 1, the logical processor invalidates all guest-physical mappings and combined mappings associated with the EPTRTA specified in the INVEPT descriptor. Combined mappings for that EPTRTA are invalidated for all VPIDs and all PCIDs. (The instruction may invalidate mappings associated with other EPTRTAs.)
 - **All-context.** If the INVEPT type is 2, the logical processor invalidates guest-physical mappings and combined mappings associated with all EPTRTAs (and, for combined mappings, for all VPIDs and PCIDs).

See Chapter 32 for details of the INVEPT instruction. See Section 30.4.3.4 for guidelines regarding use of this instruction.

- A power-up or a reset invalidates all linear mappings, guest-physical mappings, and combined mappings.

30.4.3.2 Operations that Need Not Invalidate Cached Mappings

The following items detail cases of operations that are not required to invalidate certain cached mappings:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation are not required to invalidate any guest-physical mappings.
- The INVVPID instruction is not required to invalidate any guest-physical mappings.
- The INVEPT instruction is not required to invalidate any linear mappings.

- VMX transitions are not required to invalidate any guest-physical mappings. If the “enable VPID” VM-execution control is 1, VMX transitions are not required to invalidate any linear mappings or combined mappings.
- The VMXOFF and VMXON instructions are not required to invalidate any linear mappings, guest-physical mappings, or combined mappings.

A logical processor may invalidate any cached mappings at any time. For this reason, the operations identified above may invalidate the indicated mappings despite the fact that doing so is not required.

30.4.3.3 Guidelines for Use of the INVVPID Instruction

The need for VMM software to use the INVVPID instruction depends on how that software is virtualizing memory.

If EPT is not in use, it is likely that the VMM is virtualizing the guest paging structures. Such a VMM may configure the VMCS so that all or some of the operations that invalidate entries the TLBs and the paging-structure caches (e.g., the INVLPG instruction) cause VM exits. If VMM software is emulating these operations, it may be necessary to use the INVVPID instruction to ensure that the logical processor’s TLBs and the paging-structure caches are appropriately invalidated.

Requirements of when software should use the INVVPID instruction depend on the specific algorithm being used for page-table virtualization. The following items provide guidelines for software developers:

- Emulation of the INVLPG instruction may require execution of the INVVPID instruction as follows:
 - The INVVPID type is individual-address (0).
 - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
 - The linear address in the INVVPID descriptor is that of the operand of the INVLPG instruction being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—except for global translations. An example is the MOV to CR3 instruction. (See Section 5.10, “Caching Translation Information,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for details regarding global translations.) Emulation of such an instruction may require execution of the INVVPID instruction as follows:
 - The INVVPID type is single-context-retaining-globals (3).
 - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—including for global translations. An example is the MOV to CR4 instruction if the value of value of bit 4 (page global enable—PGE) is changing. Emulation of such an instruction may require execution of the INVVPID instruction as follows:
 - The INVVPID type is single-context (1).
 - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

If EPT is not in use, the logical processor associates all mappings it creates with the current VPID, and it will use such mappings to translate linear addresses. For that reason, a VMM should not use the same VPID for different non-EPT guests that use different page tables. Doing so may result in one guest using translations that pertain to the other.

If EPT is in use, the instructions enumerated above might not be configured to cause VM exits and the VMM might not be emulating them. In that case, executions of the instructions by guest software properly invalidate the required entries in the TLBs and paging-structure caches (see Section 30.4.3.1); execution of the INVVPID instruction is not required.

If EPT is in use, the logical processor associates all mappings it creates with the value of bits 51:12 of current EPTP. If a VMM uses different EPTP values for different guests, it may use the same VPID for those guests. Doing so cannot result in one guest using translations that pertain to the other.

The following guidelines apply more generally and are appropriate even if EPT is in use:

- As detailed in Section 31.4.5, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the paging structures. If, at one

time, the current VPID on a logical processor was a non-zero value X, it is recommended that software use the INVVPID instruction with the “single-context” INVVPID type and with VPID X in the INVVPID descriptor before a VM entry on the same logical processor that establishes VPID X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.

- Software can use the INVVPID instruction with the “all-context” INVVPID type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from paging structures between separate uses of VMX operation.

30.4.3.4 Guidelines for Use of the INVEPT Instruction

The following items provide guidelines for use of the INVEPT instruction to invalidate information cached from the EPT paging structures.

- Software should use the INVEPT instruction with the “single-context” INVEPT type after making any of the following changes to an EPT paging-structure entry (the INVEPT descriptor should contain an EPTP value that references — directly or indirectly — the modified EPT paging structure):
 - Changing any of the privilege bits 2:0 from 1 to 0.¹
 - Changing the physical address in bits 51:12.
 - Clearing bit 8 (the accessed flag) if accessed and dirty flags for EPT will be enabled.
 - For an EPT PDPTE or an EPT PDE, changing bit 7 (which determines whether the entry maps a page).
 - For the **last** EPT paging-structure entry used to translate a guest-physical address (an EPT PDPTE with bit 7 set to 1, an EPT PDE with bit 7 set to 1, or an EPT PTE), changing either bits 5:3 or bit 6. (These bits determine the effective memory type of accesses using that EPT paging-structure entry; see Section 30.3.7.)
 - For the **last** EPT paging-structure entry used to translate a guest-physical address (an EPT PDPTE with bit 7 set to 1, an EPT PDE with bit 7 set to 1, or an EPT PTE), clearing bit 9 (the dirty flag) if accessed and dirty flags for EPT will be enabled.
- Software should use the INVEPT instruction with the “single-context” INVEPT type before a VM entry with an EPTP value X such that $X[6] = 1$ (accessed and dirty flags for EPT are enabled) if the logical processor had earlier been in VMX non-root operation with an EPTP value Y such that $Y[6] = 0$ (accessed and dirty flags for EPT are not enabled) and $Y[51:12] = X[51:12]$.
- Software should use the INVEPT instruction with the “single-context” INVEPT type before a VM entry with an EPTP value X if the logical processor had earlier been in VMX non-root operation with an EPTP value Y such that $Y[5:3] \neq X[5:3]$ (different EPT page-walk length) and $Y[51:12] = X[51:12]$.
- Software may use the INVEPT instruction after modifying a present EPT paging-structure entry (see Section 30.3.2) to change any of the privilege bits 2:0 from 0 to 1.² Failure to do so may cause an EPT violation that would not otherwise occur. Because an EPT violation invalidates any mappings that would be used by the access that caused the EPT violation (see Section 30.4.3.1), an EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed.
- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present (see Section 30.3.2) or misconfigured (see Section 30.3.3.1), it is not necessary to execute INVEPT following modification of an EPT paging-structure entry that had been not present or misconfigured.
- As detailed in Section 31.4.5, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the EPT paging structures. If EPT was in use on a logical processor at one time with EPTP X, it is recommended that software use the INVEPT instruction with the “single-context” INVEPT type and with EPTP X in the INVEPT descriptor before a VM entry on the same logical processor that enables EPT with EPTP X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.

1. If the “mode-based execute control for EPT” VM-execution control is 1, software should use the INVEPT instruction after changing privilege bit 10 from 1 to 0.

2. If the “mode-based execute control for EPT” VM-execution control is 1, software may use the INVEPT instruction after modifying a present EPT paging-structure entry to change privilege bit 10 from 0 to 1.

- Software can use the INVEPT instruction with the “all-context” INVEPT type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from EPT paging structures between separate uses of VMX operation.

In a system containing more than one logical processor, software must account for the fact that information from an EPT paging-structure entry may be cached on logical processors other than the one that modifies that entry. The process of propagating the changes to a paging-structure entry is commonly referred to as “TLB shutdown.” A discussion of TLB shutdown appears in Section 5.10.5, “Propagation of Paging-Structure Changes to Multiple Processors,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

24. Updates to Chapter 31, Volume 3C

Change bars and violet text show changes to Chapter 31 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- In Section 31.1.6.2, "IPI Virtualization Using Virtual-Interrupt Posting", added footnote for physical address width in the pseudocode of IPI virtualization.
- In Section 31.2.1, "Evaluation of Pending Virtual Interrupts", added content in the pseudocode for SEAM. Information for when virtual interrupts will be delivered in SEAM was included.
- In Section 31.2.2, "Virtual-Interrupt Delivery", added content in the pseudocode for SEAM.

CHAPTER 31

APIC VIRTUALIZATION AND VIRTUAL INTERRUPTS

The VMCS includes controls that enable the virtualization of interrupts and the Advanced Programmable Interrupt Controller (APIC).

When these controls are used, the processor will emulate many accesses to the APIC, track the state of the virtual APIC, and deliver virtual interrupts — all in VMX non-root operation without a VM exit.¹

The processor tracks the state of the virtual APIC using a virtual-APIC page identified by the virtual-machine monitor (VMM). Section 31.1 discusses the virtual-APIC page and how the processor uses it to track the state of the virtual APIC.

The following are the VM-execution controls relevant to APIC virtualization and virtual interrupts (see Section 26.6 for information about the locations of these controls):

- **Virtual-interrupt delivery.** This control enables the evaluation and delivery of pending virtual interrupts (Section 31.2). It also enables the emulation of writes (memory-mapped or MSR-based, as enabled) to the APIC registers that control interrupt prioritization.
- **Use TPR shadow.** This control enables emulation of accesses to the APIC's task-priority register (TPR) via CR8 (Section 31.3) and, if enabled, via the memory-mapped or MSR-based interfaces.
- **Virtualize APIC accesses.** This control enables virtualization of memory-mapped accesses to the APIC (Section 31.4) by causing VM exits on accesses to a VMM-specified APIC-access page. Some of the other controls, if set, may cause some of these accesses to be emulated rather than causing VM exits.
- **Virtualize x2APIC mode.** This control enables virtualization of MSR-based accesses to the APIC (Section 31.5).
- **APIC-register virtualization.** This control allows memory-mapped and MSR-based reads of most APIC registers (as enabled) by satisfying them from the virtual-APIC page. It directs memory-mapped writes to the APIC-access page to the virtual-APIC page, following them by VM exits for VMM emulation.
- **Process posted interrupts.** This control allows software to post virtual interrupts in a data structure and send a notification to another logical processor; upon receipt of the notification, the target processor will process the posted interrupts by copying them into the virtual-APIC page (Section 31.6).
- **IPI virtualization.** This control enables the virtualization of interprocessor interrupts (Section 31.1.6).

"Virtualize APIC accesses", "virtualize x2APIC mode", "virtual-interrupt delivery", and "APIC-register virtualization" are all secondary processor-based VM-execution controls; if bit 31 of the primary processor-based VM-execution controls is 0, the processor operates as if these controls were all 0. "IPI virtualization" is a tertiary processor-based VM-execution control; if bit 17 of the primary processor-based VM-execution controls is 0, the processor operates as if "IPI virtualization" were 0. See Section 26.6.2.

31.1 VIRTUAL APIC STATE

The **virtual-APIC page** is a 4-KByte region of memory that the processor uses to virtualize certain accesses to APIC registers and to manage virtual interrupts. The physical address of the virtual-APIC page is the **virtual-APIC address**, a 64-bit VM-execution control field in the VMCS (see Section 26.6.8).

Depending on the settings of certain VM-execution controls, the processor may virtualize certain fields on the virtual-APIC page with functionality analogous to that performed by the local APIC. Section 31.1.1 identifies and defines these fields. Section 31.1.2, Section 31.1.3, Section 31.1.4, and Section 31.1.5 detail the actions taken to virtualize updates to some of these fields.

With the exception of fields corresponding to virtualized APIC registers (defined in Section 31.1.1), software may modify the virtual-APIC page referenced by the current VMCS of a logical processor in VMX non-root operation. (This is an exception to the general requirement given in Section 26.11.4.)

1. In most cases, it is not necessary for a virtual-machine monitor (VMM) to inject virtual interrupts as part of VM entry.

31.1.1 Virtualized APIC Registers

Depending on the setting of certain VM-execution controls, a logical processor may virtualize certain accesses to APIC registers using the following fields on the virtual-APIC page:

- **Virtual task-priority register (VTPR):** the 32-bit field located at offset 080H on the virtual-APIC page.
- **Virtual processor-priority register (VPPR):** the 32-bit field located at offset 0A0H on the virtual-APIC page.
- **Virtual end-of-interrupt register (VEOI):** the 32-bit field located at offset 0B0H on the virtual-APIC page.
- **Virtual interrupt-service register (VISR):** the 256-bit value comprising eight non-contiguous 32-bit fields at offsets 100H, 110H, 120H, 130H, 140H, 150H, 160H, and 170H on the virtual-APIC page. Bit x of the VISR is at bit position $(x \& 1FH)$ at offset $(100H \mid ((x \& E0H) \gg 1))$. The processor uses only the low 4 bytes of each of the 16-byte fields at offsets 100H, 110H, 120H, 130H, 140H, 150H, 160H, and 170H.
- **Virtual interrupt-request register (VIRR):** the 256-bit value comprising eight non-contiguous 32-bit fields at offsets 200H, 210H, 220H, 230H, 240H, 250H, 260H, and 270H on the virtual-APIC page. Bit x of the VIRR is at bit position $(x \& 1FH)$ at offset $(200H \mid ((x \& E0H) \gg 1))$. The processor uses only the low 4 bytes of each of the 16-Byte fields at offsets 200H, 210H, 220H, 230H, 240H, 250H, 260H, and 270H.
- **Virtual interrupt-command register (VICR_LO):** the 32-bit field located at offset 300H on the virtual-APIC page.
- **Virtual interrupt-command register (VICR_HI):** the 32-bit field located at offset 310H on the virtual-APIC page.

The VTPR field virtualizes the TPR whenever the “use TPR shadow” VM-execution control is 1. The other fields indicated above virtualize the corresponding APIC registers whenever the “virtual-interrupt delivery” VM-execution control is 1. (VICR_LO and VICR_HI also virtualize the ICR when the “IPI virtualization” VM-execution control is 1.)

31.1.2 TPR Virtualization

The processor performs **TPR virtualization** in response to the following operations: (1) virtualization of the MOV to CR8 instruction; (2) virtualization of a write to offset 080H on the APIC-access page; and (3) virtualization of the WRMSR instruction with ECX = 808H. See Section 31.3, Section 31.4.3, and Section 31.5 for details of when TPR virtualization is performed.

The following pseudocode details the behavior of TPR virtualization:

```

IF “virtual-interrupt delivery” is 0
    THEN
        IF VTPR[7:4] < TPR threshold (see Section 26.6.8)
            THEN cause VM exit due to TPR below threshold;
        FI;
    ELSE
        perform PPR virtualization (see Section 31.1.3);
        evaluate pending virtual interrupts (see Section 31.2.1);
    FI;

```

Any VM exit caused by TPR virtualization is trap-like: the instruction causing TPR virtualization completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

31.1.3 PPR Virtualization

The processor performs **PPR virtualization** in response to the following operations: (1) VM entry; (2) TPR virtualization; and (3) EOI virtualization. See Section 28.3.2.5, Section 31.1.2, and Section 31.1.4 for details of when PPR virtualization is performed.

PPR virtualization uses the guest interrupt status (specifically, SVI; see Section 26.4.2) and VTPR. The following pseudocode details the behavior of PPR virtualization:

```

IF VTPR[7:4] ≥ SVI[7:4]

```



```

    THEN VPPR := VTPR & FFH;
    ELSE VPPR := SVI & FOH;
FI;

```

PPR virtualization always clears bytes 3:1 of VPPR.

PPR virtualization is caused only by TPR virtualization, EOI virtualization, and VM entry. Delivery of a virtual interrupt also modifies VPPR, but in a different way (see Section 31.2.2). No other operations modify VPPR, even if they modify SVI, VISR, or VTPR.

31.1.4 EOI Virtualization

The processor performs **EOI virtualization** in response to the following operations: (1) virtualization of a write to offset 0B0H on the APIC-access page; and (2) virtualization of the WRMSR instruction with ECX = 80BH. See Section 31.4.3 and Section 31.5 for details of when EOI virtualization is performed. EOI virtualization occurs only if the “virtual-interrupt delivery” VM-execution control is 1.

EOI virtualization uses and updates the guest interrupt status (specifically, SVI; see Section 26.4.2). The following pseudocode details the behavior of EOI virtualization:

```

Vector := SVI;
VISR[Vector] := 0; (see Section 31.1.1 for definition of VISR)
IF any bits set in VISR
    THEN SVI := highest index of bit set in VISR
    ELSE SVI := 0;
FI;
perform PPR virtualization (see Section 31.1.3);
IF EOI_exit_bitmap[Vector] = 1 (see Section 26.6.8 for definition of EOI_exit_bitmap)
    THEN cause EOI-induced VM exit with Vector as exit qualification;
    ELSE evaluate pending virtual interrupts; (see Section 31.2.1)
FI;

```

Any VM exit caused by EOI virtualization is trap-like: the instruction causing EOI virtualization completes before the VM exit occurs (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

31.1.5 Self-IPI Virtualization

The processor performs **self-IPI virtualization** in response to the following operations: (1) virtualization of a write to offset 300H on the APIC-access page; and (2) virtualization of the WRMSR instruction with ECX = 83FH. See Section 31.4.3 and Section 31.5 for details of when self-IPI virtualization is performed. Self-IPI virtualization occurs only if the “virtual-interrupt delivery” VM-execution control is 1.

Each operation that leads to self-IPI virtualization provides an 8-bit vector (see Section 31.4.3 and Section 31.5). Self-IPI virtualization updates the guest interrupt status (specifically, RVI; see Section 26.4.2). The following pseudocode details the behavior of self-IPI virtualization:

```

VIRR[Vector] := 1; (see Section 31.1.1 for definition of VIRR)
RVI := max{RVI, Vector};
evaluate pending virtual interrupts; (see Section 31.2.1)

```

31.1.6 IPI Virtualization

The processor performs **IPI virtualization** in response to the following operations: (1) virtualization of a write to offset 300H on the APIC-access page (Section 31.4.3); (2) virtualization of the WRMSR instruction with ECX = 830H (Section 31.5); and (3) virtualization of some executions of SENDUIPI (Section 31.7). IPI virtualization occurs only if the “IPI virtualization” VM-execution control is 1.

IPI virtualization uses virtual-interrupt posting, which is described in Section 31.1.6.1. Section 31.1.6.2 gives the details of the operation of IPI virtualization.

31.1.6.1 Virtual-Interrupt Posting

IPI virtualization is based on **virtual-interrupt posting**, a process that can direct virtual interrupts to a specific virtual processor. With virtual-interrupt posting, a hardware or software agent “posts” a virtual interrupt in a data structure (**posted-interrupt descriptor** or **PID**) and then sends an interrupt (notification) to the logical processor on which the target virtual processor is operating. When that logical processor receives the notification, it uses information in the PID to deliver the virtual interrupt to the virtual processor (see Section 31.6).

A PID is a 64-byte data structure. In an expected usage, there is one PID for each virtual processor; the virtual processor’s VMCS contains a pointer to its PID. A PID has the format shown in Table 31-1.

Table 31-1. Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
257	Suppress notify (SN)	Setting this bit directs agents not to send notifications.
271:258	Reserved	Reserved.
279:272	Notify vector (NV)	Notifications will use this vector.
287:280	Reserved	Reserved.
319:288	Notify destination (NDST)	Notifications will be directed to this physical APIC ID.
511:320	Reserved	Reserved.

A hardware or software agent posts a virtual interrupt to a virtual processor with following steps:

1. Read the PIR field in the virtual processor’s PID and write it back atomically, setting the bit that corresponds to the virtual interrupt’s vector.
2. Read the notification-information field in the PID and write it back atomically, setting the ON bit if the ON and SN bits were both 0 in the value read. (Step #2 may be done atomically with step #1.)
3. If step #2 changed the ON bit from 0 to 1, send a notification. The notification is an ordinary interrupt sent to the physical APIC ID NDST with vector NV.

A processor’s response to the delivery of a notification is detailed in Section 31.6.

The use of virtual-interrupt posting for IPI virtualization is explained in Section 31.1.6.2.

31.1.6.2 IPI Virtualization Using Virtual-Interrupt Posting

Each operation that leads to IPI virtualization provides an 8-bit virtual vector V and a 32-bit virtual APIC ID T. IPI virtualization uses those values to initiate the indicated virtual IPI using the **PID-pointer table**.

The PID-pointer table is a data structure referenced by the PID-pointer table address, a field in the VMCS. Each entry in the PID-pointer table contains the following information:

- Bits 63:6 contain bits 63:6 of the 64-bit physical address of a PID (see Section 31.1.6.1).
- Bits 5:1 are reserved and must be 0.
- Bit 0 is a valid bit.

Each such address must be 64-byte aligned. The index of the last entry in the table is also a field in the VMCS.

When virtualizing an IPI, the CPU uses the virtual APIC ID T to select an entry from the PID-pointer table. It uses the address in that entry to locate a posted-interrupt descriptor (PID) and then posts a virtual interrupt with vector V in that PID. The following pseudocode details the behavior of IPI virtualization:

```

IF V < 16
    THEN APIC-write VM exit;           // illegal vector
ELSE IF T ≤ last PID-pointer index    // a field in the VMCS

```

```

THEN
    PID_ADDR := 8 bytes at (PID-pointer table address + (T << 3));
    IF PID_ADDR sets bits beyond the processor's physical-address width1 OR
        PID_ADDR[5:0] ≠ 000001b // PID pointer not valid or reserved bits set
    THEN APIC-write VM exit; // See Section 31.4.3.3
    ELSE
        PID_ADDR[0] := 0; // clear the valid bit before using as an address
        PIR := 32 bytes at PID_ADDR; // under lock
        PIR[V] := 1;
        store PIR at PID_ADDR; // release lock; corresponds to step #1 in Section 31.1.6.1
        NotifyInfo := 8 bytes at PID_ADDR + 32; // under lock
        IF NotifyInfo.ON = 0 AND NotifyInfo.SN = 0
            THEN
                NotifyInfo.ON := 1;
                SendNotify := 1;
            ELSE SendNotify := 0;
        FI;
        store NotifyInfo at PID_ADDR + 32; // release lock; corresponds to step #2 in Section 31.1.6.1
        IF SendNotify = 1
            THEN send an IPI specified by NotifyInfo.NDST and NotifyInfo.NV; // step #3 in Section 31.1.6.1
        FI;
    FI;
ELSE APIC-write VM exit; // virtual APIC ID beyond end of tables
FI;

```

The sending of the notification IPI is indicated by fields in the selected PID: NDST (PID[319:288]) and NV (PID[279:272]):

- If the local APIC is in xAPIC mode, this is the IPI that would be generated by writing NDST[15:8] (PID[303:296]) to ICR_HI[31:24] (offset 310H from IA32_APIC_BASE) and then writing NV to ICR_LO (offset 300H from IA32_APIC_BASE).
- If the local APIC is in x2APIC mode, this is the IPI that would be generated by executing WRMSR with ECX = 830H (ICR), EAX = NV, and EDX = NDST.

If the pseudocode specifies an APIC-write VM exit, this VM exit occurs as if there had been a write access to page offset 300H on the APIC-access page (see Section 31.4.3.3).

31.2 EVALUATION AND DELIVERY OF VIRTUAL INTERRUPTS

If the “virtual-interrupt delivery” VM-execution control is 1, certain actions in VMX non-root operation or during VM entry cause the processor to evaluate and deliver virtual interrupts.

Evaluation of virtual interrupts is triggered by certain actions change the state of the virtual-APIC page and is described in Section 31.2.1. This evaluation may result in recognition of a virtual interrupt. Once a virtual interrupt is recognized, the processor may deliver it within VMX non-root operation without a VM exit. Virtual-interrupt delivery is described in Section 31.2.2.

31.2.1 Evaluation of Pending Virtual Interrupts

If the “virtual-interrupt delivery” VM-execution control is 1, certain actions cause a logical processor to **evaluate pending virtual interrupts**.

The following actions cause the evaluation of pending virtual interrupts: VM entry; TPR virtualization; EOI virtualization; self-IPI virtualization; and posted-interrupt processing. See Section 28.3.2.5, Section 31.1.2, Section 31.1.4, Section 31.1.5, and Section 31.6 for details of when evaluation of pending virtual interrupts is performed. No other operations cause the evaluation of pending virtual interrupts, even if they modify RVI or VPPR.

1. Usually, the processor's physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Evaluation of pending virtual interrupts uses the guest interrupt status (specifically, RVI; see Section 26.4.2). The following pseudocode details the evaluation of pending virtual interrupts:

```

IF logical processor in SEAM and RVI < 31
    THEN RVI := 0;
FI;
IF "interrupt-window exiting" is 0 AND
RVI[7:4] > VPPR[7:4] (see Section 31.1.1 for definition of VPPR)
    THEN recognize a pending virtual interrupt;
ELSE
    do not recognize a pending virtual interrupt;
FI;

```

Once recognized, a virtual interrupt may be delivered in VMX non-root operation; see Section 31.2.2. **Note that, in SEAM, no virtual interrupt will be delivered with a vector lower than 31.**

Evaluation of pending virtual interrupts is caused only by VM entry, TPR virtualization, EOI virtualization, self-IPI virtualization, and posted-interrupt processing. No other operations do so, even if they modify RVI or VPPR. The logical processor ceases recognition of a pending virtual interrupt following the delivery of a virtual interrupt.

31.2.2 Virtual-Interrupt Delivery

If a virtual interrupt has been recognized (see Section 31.2.1), it is delivered at an instruction boundary when the following conditions all hold: (1) RFLAGS.IF = 1; (2) there is no blocking by STI; (3) there is no blocking by MOV SS or by POP SS; and (4) the "interrupt-window exiting" VM-execution control is 0.

Virtual-interrupt delivery has the same priority as that of VM exits due to the 1-setting of the "interrupt-window exiting" VM-execution control.¹ Thus, non-maskable interrupts (NMIs) and higher priority events take priority over delivery of a virtual interrupt; delivery of a virtual interrupt takes priority over external interrupts and lower priority events.

Virtual-interrupt delivery wakes a logical processor from the same inactive activity states as would an external interrupt. Specifically, it wakes a logical processor from the states entered using the HLT and MWAIT instructions. It does not wake a logical processor in the shutdown state or in the wait-for-SIPI state.

Virtual-interrupt delivery updates the guest interrupt status (both RVI and SVI; see Section 26.4.2) and delivers an event within VMX non-root operation without a VM exit. The following pseudocode details the behavior of virtual-interrupt delivery (see Section 31.1.1 for definition of VISR, VIRR, and VPPR):

```

Vector := RVI;
VISR[Vector] := 1;
SVI := Vector;
VPPR := Vector & FOH;
VIRR[Vector] := 0;
IF any bits set in VIRR
    THEN RVI := highest index of bit set in VIRR
    ELSE RVI := 0;
FI;
IF logical processor is in SEAM and RVI < 31
    THEN RVI := 0;
FI;
cease recognition of any pending virtual interrupt;
IF transactional execution is in effect
    THEN abort transactional execution and transition to a non-transactional execution;
FI;
IF logical processor is in enclave mode

```

1. A logical processor never recognizes or delivers a virtual interrupt if the "interrupt-window exiting" VM-execution control is 1. Because of this, the relative priority of virtual-interrupt delivery and VM exits due to the 1-setting of that control is not defined.

```

    THEN cause an Asynchronous Enclave Exit (AEX) (see Chapter 39, "Enclave Exiting Events")
FI;
IF CR4.UINTR = 1 AND IA32_EFER.LMA = 1 AND Vector = UINV
    THEN virtualize user-interrupt notification identification and processing (see Section 31.2.3)
    ELSE deliver interrupt with Vector through IDT;
FI;

```

31.2.3 Virtualizing User-Interrupt Notifications

Section 8.5 describes the process of user-interrupt notification identification and processing. If the "virtual-interrupt delivery" VM-execution control is 1, this process is modified as described in the following paragraphs.

The virtualized form of user-interrupt notification identification begins as described in Section 31.2.2. Following this, instead of writing zero to the EOI register in the local APIC, the logical processor performs the initial steps of EOI virtualization:

```

VISR[V] := 0;
IF any bit is set in VISR
    THEN SVI := highest index of bit set in VISR
    ELSE SVI := 0;
FI;
perform PPR virtualization (Section 31.1.3);

```

Unlike EOI virtualization resulting from a guest write to the EOI register (as defined for virtual-interrupt delivery), the logical processor does not check the EOI-exit bitmap as part of this modified form of user-interrupt notification identification, and the corresponding VM exits cannot occur.

Following this modified form of user-interrupt notification identification, the logical processor then performs user-interrupt notification processing as specified in Section 8.5.2.

A logical processor is not interruptible during this modified form of user-interrupt notification identification or between it and any subsequent user-interrupt notification processing.

If the user-interrupt notification identification that precedes user-interrupt notification processing occurred while the logical processor was in the HLT state, the logical processor returns to the HLT state following user-interrupt notification processing.

31.3 VIRTUALIZING CR8-BASED TPR ACCESSES

In 64-bit mode, software can access the local APIC's task-priority register (TPR) through CR8. Specifically, software uses the MOV from CR8 and MOV to CR8 instructions (see Section 12.8.6, "Task Priority in IA-32e Mode"). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these CR8-based APIC accesses by setting the "CR8-load exiting" and "CR8-store exiting" VM-execution controls, ensuring that the accesses cause VM exits (see Section 27.1.3). Alternatively, there are methods for virtualizing some CR8-based APIC accesses without VM exits.

Normally, an execution of MOV from CR8 or MOV to CR8 that does not fault or cause a VM exit accesses the APIC's TPR. However, such an execution are treated specially if the "use TPR shadow" VM-execution control is 1. The following items provide details:

- **MOV from CR8.** The instruction loads bits 3:0 of its destination operand with bits 7:4 of VTPR (see Section 31.1.1). Bits 63:4 of the destination operand are cleared.
- **MOV to CR8.** The instruction stores bits 3:0 of its source operand into bits 7:4 of VTPR; the remainder of VTPR (bits 3:0 and bits 31:8) are cleared. Following this, the processor performs TPR virtualization (see Section 31.1.2).

31.4 VIRTUALIZING MEMORY-MAPPED APIC ACCESSSES

When the local APIC is in xAPIC mode, software accesses the local APIC's control registers using a memory-mapped interface. Specifically, software uses linear addresses that translate to physical addresses on page frame indicated by the base address in the IA32_APIC_BASE MSR (see Section 12.4.4, "Local APIC Status and Location"). This section describes how these accesses can be virtualized.

A virtual-machine monitor (VMM) can virtualize these memory-mapped APIC accesses by ensuring that any access to a linear address that would access the local APIC instead causes a VM exit. This could be done using paging or the extended page-table mechanism (EPT). Another way is by using the 1-setting of the "virtualize APIC accesses" VM-execution control.

If the "virtualize APIC accesses" VM-execution control is 1, the logical processor treats specially memory accesses using linear addresses that translate to physical addresses in the 4-KByte **APIC-access page**.^{1,2} (The APIC-access page is identified by the **APIC-access address**, a field in the VMCS; see Section 26.6.8.)

In general, an access to the APIC-access page causes an **APIC-access VM exit**. APIC-access VM exits provide a VMM with information about the access causing the VM exit. Section 31.4.1 discusses the priority of APIC-access VM exits.

Certain VM-execution controls enable the processor to virtualize certain accesses to the APIC-access page without a VM exit. In general, this virtualization causes these accesses to be made to the virtual-APIC page instead of the APIC-access page.

NOTES

Unless stated otherwise, this section characterizes only linear accesses to the APIC-access page; an access to the APIC-access page is a linear access if (1) it results from a memory access using a linear address; and (2) the access's physical address is the translation of that linear address. Section 31.4.6 discusses accesses to the APIC-access page that are not linear accesses.

The distinction between the APIC-access page and the virtual-APIC page allows a VMM to share paging structures or EPT paging structures among the virtual processors of a virtual machine (the shared paging structures referencing the same APIC-access address, which appears in the VMCS of all the virtual processors) while giving each virtual processor its own virtual APIC (the VMCS of each virtual processor will have a unique virtual-APIC address).

Section 31.4.2 discusses when and how the processor may virtualize read accesses from the APIC-access page. Section 31.4.3 does the same for write accesses. When virtualizing a write to the APIC-access page, the processor typically takes actions in addition to passing the write through to the virtual-APIC page.

The discussion in those sections uses the concept of an **operation** within which these memory accesses may occur. For those discussions, an "operation" can be an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT.

The 1-setting of the "virtualize APIC accesses" VM-execution control may also affect accesses to the APIC-access page that do not result directly from linear addresses. This is discussed in Section 31.4.6.

Special treatment may apply to Intel SGX instructions or if the logical processor is in enclave mode. See Section 41.5.3 for details.

1. Even when addresses are translated using EPT (see Section 30.3), the determination of whether an APIC-access VM exit occurs depends on an access's physical address, not its guest-physical address. Even when CR0.PG = 0, ordinary memory accesses by software use linear addresses; the fact that CR0.PG = 0 means only that the identity translation is used to convert linear addresses to physical (or guest-physical) addresses.
2. If EPT is enabled and there is write to a guest-physical address that translates to an address on the APIC-access page that is eligible for sub-page write permissions (see Section 30.3.4.1), the processor may treat the write as if the "virtualize APIC accesses" VM-execution control were 0 (and not apply the treatment specified in this section). For that reason, it is recommended that software not configure any guest-physical address that translates to an address on the APIC-access page to be eligible for sub-page write permissions.

31.4.1 Priority of APIC-Access VM Exits

The following items specify the priority of APIC-access VM exits relative to other events.

- The priority of an APIC-access VM exit due to a memory access is below that of any page fault or EPT violation that that access may incur. That is, an access does not cause an APIC-access VM exit if it would cause a page fault or an EPT violation.
- A memory access does not cause an APIC-access VM exit until after the accessed flags are set in the paging structures (including EPT paging structures, if enabled).
- A write access does not cause an APIC-access VM exit until after the dirty flags are set in the appropriate paging structure and EPT paging structure (if enabled).
- With respect to all other events, any APIC-access VM exit due to a memory access has the same priority as any page fault or EPT violation that the access could cause. (This item applies to other events that the access may generate as well as events that may be generated by other accesses by the same operation.)

These principles imply, among other things, that an APIC-access VM exit may occur during the execution of a repeated string instruction (including `INS` and `OUTS`). Suppose, for example, that the first n iterations (n may be 0) of such an instruction do not access the APIC-access page and that the next iteration does access that page. As a result, the first n iterations may complete and be followed by an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers reflect the completion of n iterations.

31.4.2 Virtualizing Reads from the APIC-Access Page

A read access from the APIC-access page causes an APIC-access VM exit if any of the following are true:

- The “use TPR shadow” VM-execution control is 0.
- The access is for an instruction fetch.
- The access is more than 32 bits in size.
- The access is part of an operation for which the processor has already virtualized a write to the APIC-access page.
- The access is not entirely contained within the low 4 bytes of a naturally aligned 16-byte region. That is, bits 3:2 of the access’s address are 0, and the same is true of the address of the highest byte accessed.

If none of the above are true, whether a read access is virtualized depends on the setting of the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls:

- A read access is virtualized if its page offset is 080H (task priority) regardless of the settings of the “APIC-register virtualization” and “virtual-interrupt delivery” VM-execution controls.
- If the “virtual-interrupt delivery” VM-execution control is 1, a read access is virtualized if its page offset is 0B0H (end of interrupt) or 300H (interrupt command — low).
- If “APIC-register virtualization” is 1, a read access is virtualized if it is entirely within one of the following ranges of offsets:
 - 020H–023H (local APIC ID);
 - 030H–033H (local APIC version);
 - 080H–083H (task priority);
 - 0B0H–0B3H (end of interrupt);
 - 0D0H–0D3H (logical destination);
 - 0E0H–0E3H (destination format);
 - 0F0H–0F3H (spurious-interrupt vector);
 - 100H–103H, 110H–113H, 120H–123H, 130H–133H, 140H–143H, 150H–153H, 160H–163H, or 170H–173H (in-service);
 - 180H–183H, 190H–193H, 1A0H–1A3H, 1B0H–1B3H, 1C0H–1C3H, 1D0H–1D3H, 1E0H–1E3H, or 1F0H–1F3H (trigger mode);

- 200H–203H, 210H–213H, 220H–223H, 230H–233H, 240H–243H, 250H–253H, 260H–263H, or 270H–273H (interrupt request);
- 280H–283H (error status);
- 300H–303H or 310H–313H (interrupt command);
- 320H–323H, 330H–333H, 340H–343H, 350H–353H, 360H–363H, or 370H–373H (LVT entries);
- 380H–383H (initial count); or
- 3E0H–3E3H (divide configuration).

In all other cases, the access causes an APIC-access VM exit.

A read access from the APIC-access page that is virtualized returns data from the corresponding page offset on the virtual-APIC page.¹

31.4.3 Virtualizing Writes to the APIC-Access Page

Whether a write access to the APIC-access page is virtualized depends on the settings of the VM-execution controls and the page offset of the access. Section 31.4.3.1 details when APIC-write virtualization occurs.

Unlike reads, writes to the local APIC have side effects; because of this, virtualization of writes to the APIC-access page may require emulation specific to the access's page offset (which identifies the APIC register being accessed). Section 31.4.3.2 describes this **APIC-write emulation**.

For some page offsets, it is necessary for software to complete the virtualization after a write completes. In these cases, the processor causes an **APIC-write VM exit** to invoke VMM software. Section 31.4.3.3 discusses APIC-write VM exits.

31.4.3.1 Determining Whether a Write Access is Virtualized

A write access to the APIC-access page causes an APIC-access VM exit if any of the following are true:

- The "use TPR shadow" VM-execution control is 0.
- The access is more than 32 bits in size.
- The access is part of an operation for which the processor has already virtualized a write (with a different page offset or a different size) to the APIC-access page.
- The access is not entirely contained within the low 4 bytes of a naturally aligned 16-byte region. That is, bits 3:2 of the access's address are 0, and the same is true of the address of the highest byte accessed.

If none of the above are true, whether a write access is virtualized depends on the settings of the "APIC-register virtualization", "virtual-interrupt delivery", and "IPI virtualization" VM-execution controls:

- A write access is virtualized if its page offset is 080H (task priority) regardless of the settings of the "APIC-register virtualization" and "virtual-interrupt delivery" VM-execution controls.
- If the "virtual-interrupt delivery" VM-execution control is 1, a write access is virtualized if its page offset is 0B0H (end of interrupt) or 300H (interrupt command — low).
- If the "IPI virtualization" VM-execution control is 1, a write access is virtualized if its page offset is 300H.
- If the "APIC-register virtualization" VM-execution control is 1, a write access is virtualized if it is entirely within one the following ranges of offsets:
 - 020H–023H (local APIC ID);
 - 080H–083H (task priority);
 - 0B0H–0B3H (end of interrupt);
 - 0D0H–0D3H (logical destination);
 - 0E0H–0E3H (destination format);

1. The memory type used for accesses that read from the virtual-APIC page is reported in bits 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

- 0F0H–0F3H (spurious-interrupt vector);
- 280H–283H (error status);
- 300H–303H or 310H–313H (interrupt command);
- 320H–323H, 330H–333H, 340H–343H, 350H–353H, 360H–363H, or 370H–373H (LVT entries);
- 380H–383H (initial count); or
- 3E0H–3E3H (divide configuration).

In all other cases, the access causes an APIC-access VM exit.

The processor virtualizes a write access to the APIC-access page by writing data to the corresponding page offset on the virtual-APIC page.¹ Following this, the processor performs certain actions after completion of the operation of which the access was a part.² APIC-write emulation is described in Section 31.4.3.2.

31.4.3.2 APIC-Write Emulation

If the processor virtualizes a write access to the APIC-access page, it performs additional actions after completion of an operation of which the access was a part. These actions are called **APIC-write emulation**.

The details of APIC-write emulation depend upon the page offset of the virtualized write access:³

- 080H (task priority). The processor clears bytes 3:1 of VTPR and then causes TPR virtualization (Section 31.1.2).
- 0B0H (end of interrupt). If the “virtual-interrupt delivery” VM-execution control is 1, the processor clears VEOI and then causes EOI virtualization (Section 31.1.4); otherwise, the processor causes an APIC-write VM exit (Section 31.4.3.3).
- 300H (interrupt command — low). If the “virtual-interrupt delivery” VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
 - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
 - Bits 19:18 (destination shorthand) are 01B (self).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bits 10:8 (delivery mode) are 000B (fixed).
 - Bits 7:4 (the upper half of the vector) are **not** 0000B.

If all of the items above are true, the processor performs self-IPI virtualization using the 8-bit vector in byte 0 of VICR_LO (Section 31.1.5).

If the “virtual-interrupt delivery” VM-execution control is 0, or if any of the items above are false, behavior depends on the setting of the “IPI virtualization” VM-execution control:

- If the “IPI virtualization” VM-execution control is 1, the processor checks the value of VICR_LO to determine whether the following are all true:
 - Reserved bits (31:20, 17:16, 13) and bit 12 (delivery status) are all 0.
 - Bits 19:18 (destination shorthand) are 00B (no shorthand).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bit 11 (destination mode) is 0 (physical).
 - Bits 10:8 (delivery mode) are 000B (fixed).

1. The memory type used for accesses that write to the virtual-APIC page is reported in bits 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

2. Recall that, for the purposes of this discussion, an operation is an iteration of a REP-prefixed string instruction, an execution of any other instruction, or delivery of an event through the IDT.

3. For any operation, there can be only one page offset for which a write access was virtualized. This is because a write access is not virtualized if the processor has already virtualized a write access for the same operation with a different page offset.

If all of the items above are true, the processor performs IPI virtualization using the 8-bit vector in byte 0 of VICR_LO and the 8-bit APIC ID in VICR_HI[31:24] (Section 31.1.6); otherwise, the processor causes an APIC-write VM exit.

- If the “IPI virtualization” VM-execution control is 0, the processor causes an APIC-write VM exit.
- 310H–313H (interrupt command — high). The processor clears bytes 2:0 of VICR_HI. No other virtualization or VM exit occurs.
- Any other page offset. The processor causes an APIC-write VM exit.

APIC-write emulation takes priority over system-management interrupts (SMIs), INIT signals, and lower priority events. APIC-write emulation is not blocked if RFLAGS.IF = 0 or by the MOV SS, POP SS, or STI instructions.

If an operation causes a fault after a write access to the APIC-access page and before APIC-write emulation, and that fault is delivered without a VM exit, APIC-write emulation occurs after the fault is delivered and before the fault handler can execute. If an operation causes a VM exit (perhaps due to a fault) after a write access to the APIC-access page and before APIC-write emulation, the APIC-write emulation does not occur.

31.4.3.3 APIC-Write VM Exits

In certain cases, VMM software must be invoked to complete the virtualization of a write access to the APIC-access page. In this case, APIC-write emulation causes an **APIC-write VM exit**. (Section 31.4.3.2 details the cases that causes APIC-write VM exits.)

APIC-write VM exits are invoked by APIC-write emulation, and APIC-write emulation occurs after an operation that performs a write access to the APIC-access page. Because of this, every APIC-write VM exit is trap-like: it occurs after completion of the operation containing the write access that caused the VM exit (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

The basic exit reason for an APIC-write VM exit is “APIC write.” The exit qualification is the page offset of the write access that led to the VM exit.

As noted in Section 31.5, execution of WRMSR with ECX = 83FH (self-IPI MSR) can lead to an APIC-write VM exit if the “virtual-interrupt delivery” VM-execution control is 1; the exit qualification for the APIC-write VM exit is 3F0H. As noted in Section 31.1.6 and in Section 31.7, IPI virtualization and execution of SENDUIPI may lead to APIC-write VM exits; these VM exits produce an exit qualification of 300H.

31.4.4 Instruction-Specific Considerations

Certain instructions that use linear address may cause page faults even though they do not use those addresses to access memory. The APIC-virtualization features may affect these instructions as well:

- **CLFLUSH, CLFLUSHOPT.** With regard to faulting, the processor operates as if each of these instructions reads from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will flush the corresponding cache line on the virtual-APIC page instead of the APIC-access page.
- **ENTER.** With regard to faulting, the processor operates if ENTER writes to the byte referenced by the final value of the stack pointer (even though it does not if its size operand is non-zero). If that value translates to an address on the APIC-access page, the instruction may cause an APIC-access VM exit. If it does not, it will cause the APIC-write emulation appropriate to the address’s page offset.
- **MASKMOVQ and MASKMOVDQU.** Even if the instruction’s mask is zero, the processor may operate with regard to faulting as if MASKMOVQ or MASKMOVDQU writes to memory (the behavior is implementation-specific). In such a situation, an APIC-access VM exit may occur.
- **MONITOR.** With regard to faulting, the processor operates as if MONITOR reads from the effective address in RAX. If the resulting linear address translates to one on the APIC-access page, the instruction may cause an

APIC-access VM exit.¹ If it does not, it will monitor the corresponding address on the virtual-APIC page instead of the APIC-access page.

- **PREFETCH.** An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit. Such an access may prefetch data; if so, it is from the corresponding address on the virtual-APIC page.

Virtualization of accesses to the APIC-access page is principally intended for basic instructions such as AND, MOV, OR, TEST, XCHG, and XOR. Use of an instruction that normally operates on floating-point, SSE, AVX, or AVX-512 registers may cause an APIC-access VM exit unconditionally regardless of the page offset it accesses on the APIC-access page.

31.4.5 Issues Pertaining to Page Size and TLB Management

The 1-setting of the “virtualize APIC accesses” VM-execution is guaranteed to apply only if translations to the APIC-access address use a 4-KByte page. The following items provide details:

- If EPT is not in use, any linear address that translates to an address on the APIC-access page should use a 4-KByte page. Any access to a linear address that translates to the APIC-access page using a larger page may operate as if the “virtualize APIC accesses” VM-execution control were 0.
- If EPT is in use, any guest-physical address that translates to an address on the APIC-access page should use a 4-KByte page. Any access to a linear address that translates to a guest-physical address that in turn translates to the APIC-access page using a larger page may operate as if the “virtualize APIC accesses” VM-execution control were 0. (This is true also for guest-physical accesses to the APIC-access page; see Section 31.4.6.1.)

In addition, software should perform appropriate TLB invalidation when making changes that may affect APIC-virtualization. The specifics depend on whether VPIDs or EPT is being used:

- **VPIDs being used but EPT not being used.** Suppose that there is a VPID that has been used before and that software has since made either of the following changes: (1) set the “virtualize APIC accesses” VM-execution control when it had previously been 0; or (2) changed the paging structures so that some linear address translates to the APIC-access address when it previously did not. In that case, software should execute INVVPID (see “INVVPID— Invalidate Translations Based on VPID” in Section 32.3) before performing on the same logical processor and with the same VPID.²
- **EPT being used.** Suppose that there is an EPTP value that has been used before and that software has since made either of the following changes: (1) set the “virtualize APIC accesses” VM-execution control when it had previously been 0; or (2) changed the EPT paging structures so that some guest-physical address translates to the APIC-access address when it previously did not. In that case, software should execute INVEPT (see “INVEPT— Invalidate Translations Derived from EPT” in Section 32.3) before performing on the same logical processor and with the same EPTP value.³
- **Neither VPIDs nor EPT being used.** No invalidation is required.

Failure to perform the appropriate TLB invalidation may result in the logical processor operating as if the “virtualize APIC accesses” VM-execution control were 0 in responses to accesses to the affected address. (No invalidation is necessary if neither VPIDs nor EPT is being used.)

31.4.6 APIC Accesses Not Directly Resulting From Linear Addresses

Section 31.4 has described the treatment of accesses that use linear addresses that translate to addresses on the APIC-access page. This section considers memory accesses that do not result directly from linear addresses.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. INVVPID should use either (1) the all-contexts INVVPID type; (2) the single-context INVVPID type with the VPID in the INVVPID descriptor; or (3) the individual-address INVVPID type with the linear address and the VPID in the INVVPID descriptor.
3. INVEPT should use either (1) the global INVEPT type; or (2) the single-context INVEPT type with the EPTP value in the INVEPT descriptor.

- An access is called a **guest-physical access** if (1) CR0.PG = 1;¹ (2) the “enable EPT” VM-execution control is 1;² (3) the access’s physical address is the result of an EPT translation; and (4) either (a) the access was not generated by a linear address; or (b) the access’s guest-physical address is not the translation of the access’s linear address. Section 31.4.6.1 discusses the treatment of guest-physical accesses to the APIC-access page.
- An access is called a **physical access** if (1) either (a) the “enable EPT” VM-execution control is 0; or (b) the access’s physical address is not the result of a translation through the EPT paging structures; and (2) either (a) the access is not generated by a linear address; or (b) the access’s physical address is not the translation of its linear address. Section 31.4.6.2 discusses the treatment of physical accesses to the APIC-access page.

31.4.6.1 Guest-Physical Accesses to the APIC-Access Page

Guest-physical accesses include the following when guest-physical addresses are being translated using EPT:

- Reads from the guest paging structures when translating a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging (see Section 5.4).
- Updates to the accessed and dirty flags in the guest paging structures when using a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Memory accesses by Intel Processor Trace when the “Intel PT uses guest physical addresses” VM-execution control is 1 (see Section 27.5.4).

Every guest-physical access using a guest-physical address that translates to an address on the APIC-access page causes an APIC-access VM exit. Such accesses are never virtualized regardless of the page offset.

The following items specify the priority relative to other events of APIC-access VM exits caused by guest-physical accesses to the APIC-access page.

- The priority of an APIC-access VM exit caused by a guest-physical access to memory is below that of any EPT violation that that access may incur. That is, a guest-physical access does not cause an APIC-access VM exit if it would cause an EPT violation.
- With respect to all other events, any APIC-access VM exit caused by a guest-physical access has the same priority as any EPT violation that the guest-physical access could cause.

31.4.6.2 Physical Accesses to the APIC-Access Page

Physical accesses include the following:

- If the “enable EPT” VM-execution control is 0:
 - Reads from the paging structures when translating a linear address.
 - Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging (see Section 5.4).
 - Updates to the accessed and dirty flags in the paging structures.
- If the “enable EPT” VM-execution control is 1, accesses to the EPT paging structures (including updates to the accessed and dirty flags for EPT).
- Any of the following accesses made by the processor to support VMX non-root operation:
 - Accesses to the VMCS region.
 - Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM.³ These include the following:

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
2. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable EPT” VM-execution control were 0. See Section 26.6.2.

- Accesses to SMRAM during SMI delivery and during execution of RSM.
- Accesses during SMM VM exits (including accesses to MSEG) and during VM entries that return from SMM.

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. If it does not cause an APIC-access VM exit, it may access the APIC-access page or the virtual-APIC page. Physical write accesses to the APIC-access page may or may not cause APIC-write emulation or APIC-write VM exits.

The priority of an APIC-access VM exit caused by physical access is not defined relative to other events that the access may cause.

It is recommended that software not set the APIC-access address to any of the addresses used by physical memory accesses (identified above). For example, it should not set the APIC-access address to the physical address of any of the active paging structures if the “enable EPT” VM-execution control is 0.

31.5 VIRTUALIZING MSR-BASED APIC ACCESSES

When the local APIC is in x2APIC mode, software accesses the local APIC’s control registers using the MSR interface. Specifically, software uses the RDMSR and WRMSR instructions, setting ECX (identifying the MSR being accessed) to values in the range 800H–8FFH (see Section 12.12, “Extended XAPIC (x2APIC)”). This section describes how these accesses can be virtualized.

A virtual-machine monitor can virtualize these MSR-based APIC accesses by configuring the MSR bitmaps (see Section 26.6.9) to ensure that the accesses cause VM exits (see Section 27.1.3). Alternatively, there are methods for virtualizing some MSR-based APIC accesses without VM exits.

Normally, an execution of RDMSR or WRMSR that does not fault or cause a VM exit accesses the MSR indicated in ECX. However, such an execution treats some values of ECX in the range 800H–8FFH specially if the “virtualize x2APIC mode” VM-execution control is 1. The following items provide details:

- **RDMSR.** The instruction’s behavior depends on the setting of the “APIC-register virtualization” VM-execution control.
 - If the “APIC-register virtualization” VM-execution control is 0, behavior depends upon the value of ECX.
 - If ECX contains 808H (indicating the TPR MSR), the instruction reads the 8 bytes from offset 080H on the virtual-APIC page (VTPR and the 4 bytes above it) into EDX:EAX. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode).
 - If ECX contains any other value in the range 800H–8FFH, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a readable APIC register, EDX and EAX are loaded with the value of that register. If the local APIC is not in x2APIC mode or ECX does not indicate a readable APIC register, a general-protection fault occurs.
 - If “APIC-register virtualization” is 1 and ECX contains a value in the range 800H–8FFH, the instruction reads the 8 bytes from offset X on the virtual-APIC page into EDX:EAX, where $X = (ECX \& FFH) \ll 4$. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not in x2APIC mode).
- **WRMSR.** The instruction’s behavior depends on the value of ECX and the setting of the “virtual-interrupt delivery” and “IPI virtualization” VM-execution controls.

Special processing applies in the following cases: (1) ECX contains 808H (indicating the TPR MSR); (2) ECX contains 80BH (indicating the EOI MSR) and the “virtual-interrupt delivery” VM-execution control is 1; (3) ECX contains 83FH (indicating the self-IPI MSR) and the “virtual-interrupt delivery” VM-execution control is 1; and (4) ECX contains 830H (indicating the ICR MSR) and the “IPI virtualization” VM-execution control is 1.

If special processing applies, no general-protection exception is produced due to the fact that the local APIC is in xAPIC mode. However, WRMSR does perform the normal reserved-bit checking:

- If ECX contains 808H or 83FH, a general-protection fault occurs if either EDX or EAX[31:8] is non-zero.
- If ECX contains 80BH, a general-protection fault occurs if either EDX or EAX is non-zero.

3. Technically, these accesses do not occur in VMX non-root operation. They are included here for clarity.

- If ECX contains 830H, a general-protection fault occurs if any of bits 31:20, 17:16, or 13 of EAX is non-zero. If there is no fault, WRMSR stores EDX:EAX at offset X on the virtual-APIC page, where $X = (ECX \& FFH) \ll 4$. Following this, the processor performs an operation depending on the value of ECX:
- If ECX contains 808H, the processor performs TPR virtualization (see Section 31.1.2).
- If ECX contains 80BH, the processor performs EOI virtualization (see Section 31.1.4).
- If ECX contains 83FH, the processor then checks the value of EAX[7:4] and proceeds as follows:
 - If the value is non-zero, the logical processor performs self-IPI virtualization with the 8-bit vector in EAX[7:0] (see Section 31.1.5).
 - If the value is zero, the logical processor causes an APIC-write VM exit as if there had been a write access to page offset 3F0H on the APIC-access page (see Section 31.4.3.3).
- If ECX contains 830H, the processor then checks the value of VICR to determine whether the following are all true:
 - Bits 19:18 (destination shorthand) are 00B (no shorthand).
 - Bit 15 (trigger mode) is 0 (edge).
 - Bit 12 (unused) is 0.
 - Bit 11 (destination mode) is 0 (physical).
 - Bits 10:8 (delivery mode) are 000B (fixed).

If all of the items above are true, the processor performs IPI virtualization using the 8-bit vector in byte 0 of VICR and the 32-bit APIC ID in VICR[63:32] (see Section 31.1.6). Otherwise, the logical processor causes an APIC-write VM exit (see Section 31.4.3.3).

If special processing does not apply, the instruction operates normally. If the local APIC is in x2APIC mode and ECX indicates a writable APIC register, the value in EDX:EAX is written to that register. If the local APIC is not in x2APIC mode or ECX does not indicate a writable APIC register, a general-protection fault occurs.

31.6 POSTED-INTERRUPT PROCESSING

Posted-interrupt processing is a feature by which a processor processes the virtual interrupts by recording them as pending on the virtual-APIC page.

Posted-interrupt processing is enabled by setting the “process posted interrupts” VM-execution control. The processing is performed in response to the arrival of an interrupt with the **posted-interrupt notification vector**. In response to such an interrupt, the processor processes virtual interrupts recorded in a data structure called a **posted-interrupt descriptor (PID)**. The posted-interrupt notification vector and the address of the PID are fields in the VMCS; see Section 26.6.8.

If the “process posted interrupts” VM-execution control is 1, a logical processor uses a 64-byte posted-interrupt descriptor located at the posted-interrupt descriptor address. Table 31-2 gives the format of a PID:¹

Table 31-2. Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
511:257	Reserved or used for virtual-interrupt posting	Some of these bits are used by virtual-interrupt posting (Section 31.1.6.1). Posted-interrupt processing does not use or modify these bits.

1. Table 31-1 gives the same format; it is repeated here for the reader, omitting fields that are not used by posted-interrupt processing.

Use of the PID differs from that of other data structures that are referenced by pointers in a VMCS. There is a general requirement that software ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. That requirement does not apply to the posted-interrupt descriptor. There is a requirement, however, that such modifications be done using locked read-modify-write instructions or other atomic operations.

If the “external-interrupt exiting” VM-execution control is 1, any unmasked external interrupt causes a VM exit (see Section 27.2). If the “process posted interrupts” VM-execution control is also 1, this behavior is changed and the processor handles an external interrupt as follows:¹

1. The local APIC is acknowledged; this provides the processor core with an interrupt vector, called here the **physical vector**.
2. If the physical vector equals the posted-interrupt notification vector, the logical processor continues to the next step. Otherwise, a VM exit occurs as it would normally due to an external interrupt; the vector is saved in the VM-exit interruption-information field.
3. The processor clears the outstanding-notification bit in the posted-interrupt descriptor. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation).
4. The processor writes zero to the EOI register in the local APIC; this dismisses the interrupt with the posted-interrupt notification vector from the local APIC.
5. The logical processor performs a logical-OR of PIR into VIRR and clears PIR. No other agent can read or write a PIR bit (or group of bits) between the time it is read (to determine what to OR into VIRR) and when it is cleared.
6. The logical processor sets RVI to be the maximum of the old value of RVI and the highest index of all bits that were set in PIR; if no bit was set in PIR, RVI is left unmodified.
7. The logical processor evaluates pending virtual interrupts as described in Section 31.2.1.

The logical processor performs the steps above in an uninterruptible manner. If step #7 leads to recognition of a virtual interrupt, the processor may deliver that interrupt immediately.

Steps #1 to #7 above occur when the interrupt controller delivers an unmasked external interrupt to the CPU core. The following items consider certain cases of interrupt delivery:

- Interrupt delivery can occur between iterations of a REP-prefixed instruction (after at least one iteration has completed but before all iterations have completed). If this occurs, the following items characterize processor state after posted-interrupt processing completes and before guest execution resumes:
 - RIP references the REP-prefixed instruction;
 - RCX, RSI, and RDI are updated to reflect the iterations completed; and
 - RFLAGS.RF = 1.
- Interrupt delivery can occur when the logical processor is in the active, HLT, or MWAIT states. If the logical processor had been in the active or MWAIT state before the arrival of the interrupt, it is in the active state following completion of step #7; if it had been in the HLT state, it returns to the HLT state after step #7 (if a pending virtual interrupt was recognized, the logical processor may immediately wake from the HLT state).
- Interrupt delivery can occur while the logical processor is in enclave mode. If the logical processor had been in enclave mode before the arrival of the interrupt, an Asynchronous Enclave Exit (AEX) may occur before the steps #1 to #7 (see Chapter 39, “Enclave Exiting Events”). If no AEX occurs before step #1 and a VM exit occurs at step #2, an AEX occurs before the VM exit is delivered.

1. VM entry ensures that the “process posted interrupts” VM-execution control is 1 only if the “external-interrupt exiting” VM-execution control is also 1. See Section 28.2.1.1.

31.7 VIRTUALIZING SENDUIPI

The user-interrupt feature includes the SENDUIPI instruction that software operating with CPL = 3 can use to send user interrupts to another software thread (“user IPIs”). The SENDUIPI instruction has the following high-level operation:

- read selected entry from user-interrupt target table;
- use address in entry to read the referenced user posted-interrupt descriptor (UPID);
- update certain fields in UPID;
- if necessary, send ordinary IPI indicated in UPID’s notification information;

The last step uses two fields in the UPID: an 8-bit notification vector (UPID.NV) and a 32-bit notification destination (an APIC ID, UPID.NDST). Outside of VMX non-root operation, the processor implements the last step as follows:

- If the local APIC is in xAPIC mode, it writes UPID.NDST[15:8] to ICR_HI[31:24] (offset 310H from IA32_APIC_BASE) and then writes UPID.NV to ICR_LO (offset 300H).
- If the local APIC is in x2APIC mode, it performs the control-register write that would be done by an execution of WRMSR with ECX = 310H (ICR), EAX = UPID.NV, and EDX = UPID.NDST.

In VMX non-root operation, implementation of the step depends on the settings of the “use TPR shadow,” “virtualize APIC accesses,” and “IPI virtualization” VM-execution controls:¹

1. If the “use TPR shadow” VM-execution control is 0, the behavior is not modified: the logical processor sends the specified IPI by writing to the local APIC’s ICR as specified above (based on the current mode of the local APIC).
2. If the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” VM-execution control is 0, the logical processor virtualizes the sending of an x2APIC-mode IPI with the following steps:
 - a. The 64-bit value Z is written to offset 300H on the virtual-APIC page (VICR), where Z[7:0] = UPID.NV (the 8-bit virtual vector), Z[63:32] = UPID.NDST (the 32-bit virtual APIC ID) and Z[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
 - b. If the “IPI virtualization” VM-execution control is 1, IPI virtualization (Section 31.1.6) is performed using the vector UPID.NV and the 32-bit virtual APIC ID UPID.NDST.
3. If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1, the logical processor virtualizes the sending of an xAPIC-mode IPI by performing the following steps:
 - a. The 32-bit value X is written to offset 310H on the virtual-APIC page (VICR_HI), where X[31:24] = UPID.NDST[15:8] (the 8-bit virtual APIC ID) and X[23:0] = 000000H.²
 - b. The 32-bit value Y is written to offset 300H on the virtual-APIC page (VICR_LO), where Y[7:0] = UPID.NV (the 8-bit virtual vector) and Y[31:8] = 000000H (indicating a physically addressed fixed-mode IPI).
 - c. If the “IPI virtualization” VM-execution control is 1, IPI virtualization is performed using the vector UPID.NV and the APIC ID UPID.NDST[15:8]. IPI virtualization will use only the 8-bit APIC ID from bits 15:8 of the UPID’s destination field (the 8-bit value written earlier to bits 31:24 of VICR_HI).
4. If the “use TPR shadow” VM-execution control is 1 and the “IPI virtualization” VM-execution control is 0, an APIC-write VM exit occurs as if there had been a write access to page offset 300H on the APIC-access page (see Section 31.4.3.3).

1. The setting of the “virtualize x2APIC mode” VM-execution control does not affect this operation.

2. For xAPIC mode (which is virtualized if the “virtualize APIC accesses” VM-execution control is 1), the destination APIC ID is in byte 1 (**not** byte 0) of the UPID’s 4-byte NDST field.

25. Updates to Chapter 32, Volume 3C

Change bars and violet text show changes to Chapter 32 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Updated the following instructions in Section 32.3, “VMX Instructions” for processors that support SEAM:
 - INVEPT— Invalidate Translations Derived from EPT. Reference to Chapter 28 was added.
 - VMCLEAR—Clear Virtual-Machine Control Structure. Operation section and Protected Mode Exceptions section were updated.
 - VMPTRLD—Load Pointer to Virtual-Machine Control Structure. A footnote for the processor’s physical-address width was added in the Operation section.
 - VMXOFF—Leave VMX Operation. Operation, Protected Mode Exceptions, and 64-Bit Mode Exceptions sections were updated.
 - VMXON—Enter VMX Operation. A footnote for the processor’s physical-address width was added in the Operation section.

32.1 OVERVIEW

This chapter describes the virtual-machine extensions (VMX) for the Intel 64 and IA-32 architectures. VMX is intended to support virtualization of processor hardware and a system software layer acting as a host to multiple guest software environments. The virtual-machine extensions (VMX) includes five instructions that manage the virtual-machine control structure (VMCS), four instructions that manage VMX operation, two TLB-management instructions, and two instructions for use by guest software. Additional details of VMX are described in Chapter 25 through Chapter 31.

The behavior of the VMCS-maintenance instructions is summarized below:

- **VMPTRLD** — This instruction takes a single 64-bit source operand that is in memory. It makes the referenced VMCS active and current, loading the current-VMCS pointer with this operand and establishes the current VMCS based on the contents of VMCS-data area in the referenced VMCS region. Because this makes the referenced VMCS active, a logical processor may start maintaining on the processor some of the VMCS data for the VMCS.
- **VMPTRST** — This instruction takes a single 64-bit destination operand that is in memory. The current-VMCS pointer is stored into the destination operand.
- **VMCLEAR** — This instruction takes a single 64-bit operand that is in memory. The instruction sets the launch state of the VMCS referenced by the operand to “clear”, renders that VMCS inactive, and ensures that data for the VMCS have been written to the VMCS-data area in the referenced VMCS region. If the operand is the same as the current-VMCS pointer, that pointer is made invalid.
- **VMREAD** — This instruction reads a component from a VMCS (the encoding of that field is given in a register operand) and stores it into a destination operand that may be a register or in memory.
- **VMWRITE** — This instruction writes a component to a VMCS (the encoding of that field is given in a register operand) from a source operand that may be a register or in memory.

The behavior of the VMX management instructions is summarized below:

- **VMLAUNCH** — This instruction launches a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
- **VMRESUME** — This instruction resumes a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
- **VMXOFF** — This instruction causes the processor to leave VMX operation.
- **VMXON** — This instruction takes a single 64-bit source operand that is in memory. It causes a logical processor to enter VMX root operation and to use the memory referenced by the operand to support VMX operation.

The behavior of the VMX-specific TLB-management instructions is summarized below:

- **INVEPT** — This instruction invalidates entries in the TLBs and paging-structure caches that were derived from extended page tables (EPT).
- **INVVPID** — This instruction invalidates entries in the TLBs and paging-structure caches based on a Virtual-Processor Identifier (VPID).

None of the instructions above can be executed in compatibility mode; they generate invalid-opcode exceptions if executed in compatibility mode.

The behavior of the guest-available instructions is summarized below:

- **VMCALL** — This instruction allows software in VMX non-root operation to call the VMM for service. A VM exit occurs, transferring control to the VMM.
- **VMFUNC** — This instruction allows software in VMX non-root operation to invoke a VM function (processor functionality enabled and configured by software in VMX root operation) without a VM exit.

32.2 CONVENTIONS

The operation sections for the VMX instructions in Section 32.3 use the pseudo-function VMExit, which indicates that the logical processor performs a VM exit.

The operation sections also use the pseudo-functions VMsucceed, VMfail, VMfailInvalid, and VMfailValid. These pseudo-functions signal instruction success or failure by setting or clearing bits in RFLAGS and, in some cases, by writing the VM-instruction error field. The following pseudocode fragments detail these functions:

VMsucceed:

```
CF := 0;
PF := 0;
AF := 0;
ZF := 0;
SF := 0;
OF := 0;
```

VMfail(ErrorNumber):

```
IF VMCS pointer is valid
    THEN VMfailValid(ErrorNumber);
    ELSE VMfailInvalid;
FI;
```

VMfailInvalid:

```
CF := 1;
PF := 0;
AF := 0;
ZF := 0;
SF := 0;
OF := 0;
```

VMfailValid(ErrorNumber)// executed only if there is a current VMCS

```
CF := 0;
PF := 0;
AF := 0;
ZF := 1;
SF := 0;
OF := 0;
Set the VM-instruction error field to ErrorNumber;
```

The different VM-instruction error numbers are enumerated in Section 32.4, “VM Instruction Error Numbers.”

32.3 VMX INSTRUCTIONS

This section provides detailed descriptions of the VMX instructions.

Processors that support SEAM also support the following SEAM instructions: SEAMCALL, SEAMOPS, SEAMRET, and TDCALL. These instructions are specified in Section 34.5, “SEAM Instruction Reference”.

INVEPT— Invalidate Translations Derived from EPT

Opcode/ Instruction	Op/En	Description
66 0F 38 80 INVEPT r64, m128	RM	Invalidates EPT-derived entries in the TLBs and paging-structure caches (in 64-bit mode).
66 0F 38 80 INVEPT r32, m128	RM	Invalidates EPT-derived entries in the TLBs and paging-structure caches (outside 64-bit mode).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	NA	NA

Description

Invalidates mappings in the translation lookaside buffers (TLBs) and paging-structure caches that were derived from extended page tables (EPT). (See Chapter 30, “VMX Support for Address Translation.”) Invalidation is based on the **INVEPT type** specified in the register operand and the **INVEPT descriptor** specified in the memory operand.

Outside IA-32e mode, the register operand is always 32 bits, regardless of the value of CS.D; in 64-bit mode, the register operand has 64 bits (the instruction cannot be executed in compatibility mode).

The INVEPT types supported by a logical processors are reported in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A, “VMX Capability Reporting Facility”). There are two INVEPT types currently defined:

- Single-context invalidation. If the INVEPT type is 1, the logical processor invalidates all mappings associated with bits 51:12 of the EPT pointer (EPTP) specified in the INVEPT descriptor. It may invalidate other mappings as well.
- Global invalidation: If the INVEPT type is 2, the logical processor invalidates mappings associated with all EPTPs.

If an unsupported INVEPT type is specified, the instruction fails.

INVEPT invalidates all the specified mappings for the indicated EPTP(s) regardless of the VPID and PCID values with which those mappings may be associated.

The INVEPT descriptor comprises 128 bits and contains a 64-bit EPTP value in bits 63:0 (see Figure 32-1).

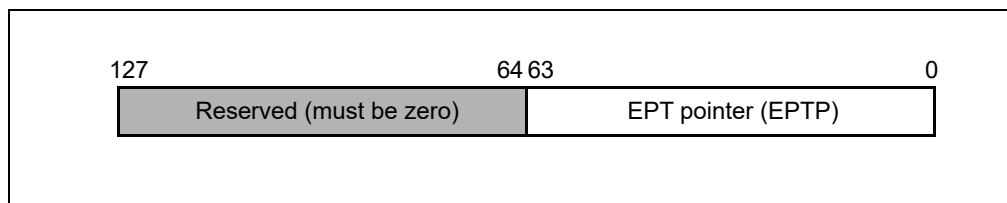


Figure 32-1. INVEPT Descriptor

Operation

```

IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
ELSE

```

```

INVEPT_TYPE := value of register operand;
IF IA32_VMX_EPT_VPID_CAP MSR indicates that processor does not support INVEPT_TYPE
    THEN VMfail(Invalid operand to INVEPT/INVVPID);
    ELSE    // INVEPT_TYPE must be 1 or 2
        INVEPT_DESC := value of memory operand;
        EPTP := INVEPT_DESC[63:0];
        CASE INVEPT_TYPE OF
            1:    // single-context invalidation
                IF VM entry with the "enable EPT" VM execution control set to 1 (see Section 28.2.1.1)
                    would fail due to the EPTP value
                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                    ELSE
                        Invalidate mappings associated with EPTP[51:12];
                        VMSucceed;

                FI;
                BREAK;
            2:    // global invalidation
                Invalidate mappings associated with all EPTPs;
                VMSucceed;
                BREAK;
        ESAC;
    FI;
FI;

```

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If not in VMX operation. If the logical processor does not support EPT (IA32_VMX_PROCBASED_CTLX2[33]=0). If the logical processor supports EPT (IA32_VMX_PROCBASED_CTLX2[33]=1) but does not support the INVEPT instruction (IA32_VMX_EPT_VPID_CAP[20]=0).

Real-Address Mode Exceptions

#UD	The INVEPT instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The INVEPT instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The INVEPT instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation. If the logical processor does not support EPT (IA32_VMX_PROCBASED_CTL2[33]=0). If the logical processor supports EPT (IA32_VMX_PROCBASED_CTL2[33]=1) but does not support the INVEPT instruction (IA32_VMX_EPT_VPID_CAP[20]=0).

INVVPID— Invalidate Translations Based on VPID

Opcode/ Instruction	Op/En	Description
66 0F 38 81 INVVPID r64, m128	RM	Invalidates entries in the TLBs and paging-structure caches based on VPID (in 64-bit mode).
66 0F 38 81 INVVPID r32, m128	RM	Invalidates entries in the TLBs and paging-structure caches based on VPID (outside 64-bit mode).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	NA	NA

Description

Invalidates mappings in the translation lookaside buffers (TLBs) and paging-structure caches based on **virtual-processor identifier** (VPID). (See Chapter 30, “VMX Support for Address Translation.”) Invalidation is based on the **INVVPID type** specified in the register operand and the **INVVPID descriptor** specified in the memory operand.

Outside IA-32e mode, the register operand is always 32 bits, regardless of the value of CS.D; in 64-bit mode, the register operand has 64 bits (the instruction cannot be executed in compatibility mode).

The INVVPID types supported by a logical processors are reported in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A, “VMX Capability Reporting Facility”). There are four INVVPID types currently defined:

- Individual-address invalidation: If the INVVPID type is 0, the logical processor invalidates mappings for the linear address and VPID specified in the INVVPID descriptor. In some cases, it may invalidate mappings for other linear addresses (or other VPIDs) as well.
- Single-context invalidation: If the INVVPID type is 1, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor. In some cases, it may invalidate mappings for other VPIDs as well.
- All-contexts invalidation: If the INVVPID type is 2, the logical processor invalidates all mappings tagged with all VPIDs except VPID 0000H. In some cases, it may invalidate translations with VPID 0000H as well.
- Single-context invalidation, retaining global translations: If the INVVPID type is 3, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor except global translations. In some cases, it may invalidate global translations (and mappings with other VPIDs) as well. See the “Caching Translation Information” section in Chapter 5 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, for information about global translations.

If an unsupported INVVPID type is specified, the instruction fails.

INVVPID invalidates all the specified mappings for the indicated VPID(s) regardless of the EPTP and PCID values with which those mappings may be associated.

The INVVPID descriptor comprises 128 bits and consists of a VPID and a linear address as shown in Figure 32-2.

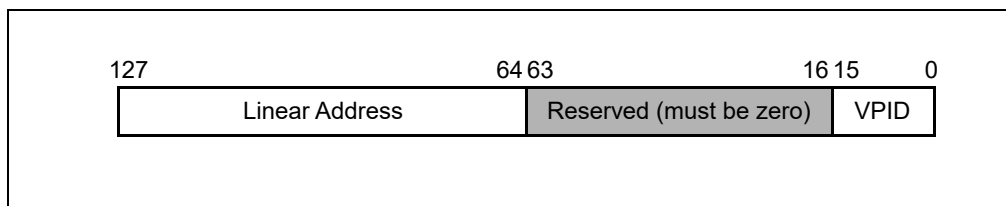


Figure 32-2. INVVPID Descriptor

Operation

```

IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
ELSE
    INVVPID_TYPE := value of register operand;
    IF IA32_VMX_EPT_VPID_CAP MSR indicates that processor does not support
    INVVPID_TYPE
        THEN VMfail(Invalid operand to INVEPT/INVVPID);
    ELSE // INVVPID_TYPE must be in the range 0–3
        INVVPID_DESC := value of memory operand;
        IF INVVPID_DESC[63:16] ≠ 0
            THEN VMfail(Invalid operand to INVEPT/INVVPID);
        ELSE
            CASE INVVPID_TYPE OF
                0: // individual-address invalidation
                    VPID := INVVPID_DESC[15:0];
                    IF VPID = 0
                        THEN VMfail(Invalid operand to INVEPT/INVVPID);
                    ELSE
                        GL_ADDR := INVVPID_DESC[127:64];
                        IF (GL_ADDR is not in a canonical form)
                            THEN
                                VMfail(Invalid operand to INVEPT/INVVPID);
                            ELSE
                                Invalidate mappings for GL_ADDR tagged with VPID;
                                VMSucceed;
                        FI;
                    FI;
                    BREAK;
                1: // single-context invalidation
                    VPID := INVVPID_DESC[15:0];
                    IF VPID = 0
                        THEN VMfail(Invalid operand to INVEPT/INVVPID);
                    ELSE
                        Invalidate all mappings tagged with VPID;
                        VMSucceed;
                    FI;
                    BREAK;
                2: // all-context invalidation
                    Invalidate all mappings tagged with all non-zero VPIDs;
                    VMSucceed;
                    BREAK;
                3: // single-context invalidation retaining globals
                    VPID := INVVPID_DESC[15:0];
                    IF VPID = 0
                        THEN VMfail(Invalid operand to INVEPT/INVVPID);
                    ELSE
                        Invalidate all mappings tagged with VPID except global translations;
                        VMSucceed;
                    FI;
            END CASE;
        END IF;
    END IF;
END IF;

```


FI;
BREAK;
ESAC;
FI;
FI;
FI;

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If not in VMX operation. If the logical processor does not support VPIDs (IA32_VMX_PROCBASED_CTLSS2[37]=0). If the logical processor supports VPIDs (IA32_VMX_PROCBASED_CTLSS2[37]=1) but does not support the INVVPID instruction (IA32_VMX_EPT_VPID_CAP[32]=0).

Real-Address Mode Exceptions

#UD	The INVVPID instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The INVVPID instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The INVVPID instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation. If the logical processor does not support VPIDs (IA32_VMX_PROCBASED_CTLSS2[37]=0). If the logical processor supports VPIDs (IA32_VMX_PROCBASED_CTLSS2[37]=1) but does not support the INVVPID instruction (IA32_VMX_EPT_VPID_CAP[32]=0).

VMCALL—Call to VM Monitor

Opcode/ Instruction	Op/En	Description
OF 01 C1 VMCALL	Z0	Call to VM monitor by causing VM exit.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	NA	NA	NA	NA

Description

This instruction allows guest software can make a call for service into an underlying VM monitor. The details of the programming interface for such calls are VMM-specific; this instruction does nothing more than cause a VM exit, registering the appropriate exit reason.

Use of this instruction in VMX root operation invokes an SMM monitor (see Section 33.15.2). This invocation will activate the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM) if it is not already active (see Section 33.15.6).

Operation

```

IF not in VMX operation
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0 or in SEAM root operation
    THEN #GP(0);
ELSIF in SMM or the logical processor does not support the dual-monitor treatment of SMIs and SMM or the valid bit in the IA32_SMM_MONITOR_CTL MSR is clear
    THEN VMfail (VMCALL executed in VMX root operation);
ELSIF dual-monitor treatment of SMIs and SMM is active
    THEN perform an SMM VM exit (see Section 33.15.2);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
ELSIF launch state of current VMCS is not clear
    THEN VMfailValid(VMCALL with non-clear VMCS);
ELSIF VM-exit control fields are not valid (see Section 33.15.6.1)
    THEN VMfailValid (VMCALL with invalid VM-exit control fields);
ELSE
    enter SMM;
    read revision identifier in MSEG;
    IF revision identifier does not match that supported by processor
        THEN
            leave SMM;
            VMfailValid(VMCALL with incorrect MSEG revision identifier);
        ELSE
            read SMM-monitor features field in MSEG (see Section 33.15.6.1);
            IF features field is invalid
                THEN
                    leave SMM;

```

VMfailValid(VMCALL with invalid SMM-monitor features);
ELSE activate dual-monitor treatment of SMLs and SMM (see Section 33.15.6);
FI;
FI;
FI;

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

- #GP(0) If the current privilege level is not 0 and the logical processor is in VMX root operation.
If in SEAM root operation.
- #UD If executed outside VMX operation.

Real-Address Mode Exceptions

- #UD If executed outside VMX operation.

Virtual-8086 Mode Exceptions

- #UD If executed outside VMX non-root operation.

Compatibility Mode Exceptions

- #UD If executed outside VMX non-root operation.

64-Bit Mode Exceptions

- #UD If executed outside VMX operation.

VMCLEAR—Clear Virtual-Machine Control Structure

Opcode/ Instruction	Op/En	Description
66 0F C7 /6 VMCLEAR m64	M	Copy VMCS data to VMCS region in memory.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	NA	NA	NA

Description

This instruction applies to the VMCS whose VMCS region resides at the physical address contained in the instruction operand. The instruction ensures that VMCS data for that VMCS (some of these data may be currently maintained on the processor) are copied to the VMCS region in memory. It also initializes parts of the VMCS region (for example, it sets the launch state of that VMCS to clear). See Chapter 26, “Virtual Machine Control Structures.”

The operand of this instruction is always 64 bits and is always in memory. If the operand is the current-VMCS pointer, then that pointer is made invalid (set to FFFFFFFF_FFFFFFFFH).

Note that the VMCLEAR instruction might not explicitly write any VMCS data to memory; the data may be already resident in memory before the VMCLEAR is executed.

Operation

IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)

THEN #UD;

ELSIF in VMX non-root operation

THEN VM exit;

ELSIF CPL > 0

THEN #GP(0);

ELSE

addr := contents of 64-bit in-memory operand;

IF addr is not 4KB-aligned OR

addr sets any bits beyond the processor's physical-address width^{1,2}

THEN VMfail(VMCLEAR with invalid physical address);

ELSIF addr = VMXON pointer

THEN VMfail(VMCLEAR with VMXON pointer);

ELSE

ensure that data for VMCS referenced by the operand is in memory;

initialize implementation-specific data in VMCS region;

launch state of VMCS referenced by the operand := “clear”

IF operand addr = current-VMCS pointer

THEN current-VMCS pointer := FFFFFFFF_FFFFFFFFH;

FI;

VMsucceed;

1. If IA32_VMX_BASIC[48] is read as 1, VMfail occurs if addr sets any bits in the range 63:32; see Appendix A.1.

2. Usually, the processor's physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

FI;
FI;

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If operand is a register. If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register. If not in VMX operation.

VMFUNC—Invoke VM function

Opcode/ Instruction	Op/En	Description
NP 0F 01 D4 VMFUNC	Z0	Invoke VM function specified in EAX.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	NA	NA	NA	NA

Description

This instruction allows software in VMX non-root operation to invoke a VM function, which is processor functionality enabled and configured by software in VMX root operation. The value of EAX selects the specific VM function being invoked.

The behavior of each VM function (including any additional fault checking) is specified in Section 27.5.6, “VM Functions.”

Operation

Perform functionality of the VM function specified in EAX;

Flags Affected

Depends on the VM function specified in EAX. See Section 27.5.6, “VM Functions.”

Protected Mode Exceptions (not including those defined by specific VM functions)

#UD If executed outside VMX non-root operation.
 If “enable VM functions” VM-execution control is 0.
 If $EAX \geq 64$.

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

VMLAUNCH/VMRESUME—Launch/Resume Virtual Machine

Opcode/ Instruction	Op/En	Description
OF 01 C2 VMLAUNCH	Z0	Launch virtual machine managed by current VMCS.
OF 01 C3 VMRESUME	Z0	Resume virtual machine managed by current VMCS.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	NA	NA	NA	NA

Description

Effects a VM entry managed by the current VMCS.

- VMLAUNCH fails if the launch state of current VMCS is not “clear”. If the instruction is successful, it sets the launch state to “launched.”
- VMRESUME fails if the launch state of the current VMCS is not “launched.”

If VM entry is attempted, the logical processor performs a series of consistency checks as detailed in Chapter 28, “VM Entries.” Failure to pass checks on the VMX controls or on the host-state area passes control to the instruction following the VMLAUNCH or VMRESUME instruction. If these pass but checks on the guest-state area fail, the logical processor loads state from the host-state area of the VMCS, passing control to the instruction referenced by the RIP field in the host-state area.

VM entry is not allowed when events are blocked by MOV SS or POP SS. Neither VMLAUNCH nor VMRESUME should be used immediately after either MOV to SS or POP to SS.

Operation

```

IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
ELSIF events are being blocked by MOV SS
    THEN VMfailValid(VM entry with events blocked by MOV SS);
ELSIF (VMLAUNCH and launch state of current VMCS is not “clear”)
    THEN VMfailValid(VMLAUNCH with non-clear VMCS);
ELSIF (VMRESUME and launch state of current VMCS is not “launched”)
    THEN VMfailValid(VMRESUME with non-launched VMCS);
ELSE
    Check settings of VMX controls and host-state area;
    IF invalid settings
        THEN VMfailValid(VM entry with invalid VMX-control field(s)) or
            VMfailValid(VM entry with invalid host-state field(s)) or
            VMfailValid(VM entry with invalid executive-VMCS pointer)) or
            VMfailValid(VM entry with non-launched executive VMCS) or
            VMfailValid(VM entry with executive-VMCS pointer not VMXON pointer) or

```

```

        VMfailValid(VM entry with invalid VM-execution control fields in executive
        VMCS)
        as appropriate;
ELSE
    Attempt to load guest state and PDPTs as appropriate;
    clear address-range monitoring;
    IF failure in checking guest state or PDPTs
        THEN VM entry fails (see Section 28.8);
    ELSE
        Attempt to load MSRs from VM-entry MSR-load area;
        IF failure
            THEN VM entry fails
                (see Section 28.8);
            ELSE
                IF VMLAUNCH
                    THEN launch state of VMCS := "launched";
                FI;
                IF in SMM and "entry to SMM" VM-entry control is 0
                    THEN
                        IF "deactivate dual-monitor treatment" VM-entry
                            control is 0
                            THEN SMM-transfer VMCS pointer :=
                                current-VMCS pointer;
                        FI;
                        IF executive-VMCS pointer is VMXON pointer
                            THEN current-VMCS pointer :=
                                VMCS-link pointer;
                            ELSE current-VMCS pointer :=
                                executive-VMCS pointer;
                        FI;
                        leave SMM;
                    FI;
                VM entry succeeds;
            FI;
        FI;
    FI;
FI;

```

Further details of the operation of the VM-entry appear in Chapter 28.

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
#UD	If executed outside VMX operation.

Real-Address Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The VMLAUNCH and VMRESUME instructions are not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0) If the current privilege level is not 0.

#UD If executed outside VMX operation.

VMPTRLD—Load Pointer to Virtual-Machine Control Structure

Opcode/ Instruction	Op/En	Description
NP OF C7 /6 VMPTRLD m64	M	Loads the current VMCS pointer from memory.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	NA	NA	NA

Description

Marks the current-VMCS pointer valid and loads it with the physical address in the instruction operand. The instruction fails if its operand is not properly aligned, sets unsupported physical-address bits, or is equal to the VMXON pointer. In addition, the instruction fails if the 32 bits in memory referenced by the operand do not match the VMCS revision identifier supported by this processor.¹

The operand of this instruction is always 64 bits and is always in memory.

Operation

IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
THEN #UD;

ELSIF in VMX non-root operation

THEN VMexit;

ELSIF CPL > 0

THEN #GP(0);

ELSE

addr := contents of 64-bit in-memory source operand;

IF addr is not 4KB-aligned OR

addr sets any bits beyond the physical-address width^{2,3}

THEN VMfail(VMPTRLD with invalid physical address);

ELSIF addr = VMXON pointer

THEN VMfail(VMPTRLD with VMXON pointer);

ELSE

rev := 32 bits located at physical address addr;

IF rev[30:0] ≠ VMCS revision identifier supported by processor OR

rev[31] = 1 AND processor does not support 1-setting of “VMCS shadowing”

THEN VMfail(VMPTRLD with incorrect VMCS revision identifier);

ELSE

current-VMCS pointer := addr;

VMsucceed;

FI;

FI;

1. Software should consult the VMX capability MSR VMX_BASIC to discover the VMCS revision identifier supported by this processor (see Appendix A, “VMX Capability Reporting Facility”).
2. If IA32_VMX_BASIC[48] is read as 1, VMfail occurs if addr sets any bits in the range 63:32; see Appendix A.1.
3. Usually, the processor’s physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

FI;

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If operand is a register. If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register. If not in VMX operation.

VMPTRST—Store Pointer to Virtual-Machine Control Structure

Opcode/ Instruction	Op/En	Description
NP OF C7 /7 VMPTRST m64	M	Stores the current VMCS pointer into memory.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (w)	NA	NA	NA

Description

Stores the current-VMCS pointer into a specified memory address. The operand of this instruction is always 64 bits and is always in memory.

Operation

```

IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
ELSE
    64-bit in-memory destination operand := current-VMCS pointer;
    VMSucceed;
FI;

```

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0) If the current privilege level is not 0.
If the memory destination operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
If the DS, ES, FS, or GS register contains an unusable segment.

#PF(fault-code) If the destination operand is located in a read-only data segment or any code segment.

#SS(0) If a page fault occurs in accessing the memory destination operand.
If the memory destination operand effective address is outside the SS segment limit.
If the SS register contains an unusable segment.

#UD If operand is a register.
If not in VMX operation.

Real-Address Mode Exceptions

#UD The VMPTRST instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD The VMPTRST instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD The VMPTRST instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0) If the current privilege level is not 0.
 If the destination operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.

#PF(fault-code) If a page fault occurs in accessing the memory destination operand.

#SS(0) If the destination operand is in the SS segment and the memory address is in a non-canonical form.

#UD If operand is a register.
 If not in VMX operation.

VMREAD—Read Field from Virtual-Machine Control Structure

Opcode/ Instruction	Op/En	Description
NP OF 78 VMREAD r/m64, r64	MR	Reads a specified VMCS field (in 64-bit mode).
NP OF 78 VMREAD r/m32, r32	MR	Reads a specified VMCS field (outside 64-bit mode).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA

Description

Reads a specified field from a VMCS and stores it into a specified destination operand (register or memory). In VMX root operation, the instruction reads from the current VMCS. If executed in VMX non-root operation, the instruction reads from the VMCS referenced by the VMCS link pointer field in the current VMCS.

The VMCS field is specified by the VMCS-field encoding contained in the register source operand. Outside IA-32e mode, the source operand has 32 bits, regardless of the value of CS.D. In 64-bit mode, the source operand has 64 bits.

The effective size of the destination operand, which may be a register or in memory, is always 32 bits outside IA-32e mode (the setting of CS.D is ignored with respect to operand size) and 64 bits in 64-bit mode. If the VMCS field specified by the source operand is shorter than this effective operand size, the high bits of the destination operand are cleared to 0. If the VMCS field is longer, then the high bits of the field are not read.

Note that any faults resulting from accessing a memory destination operand can occur only after determining, in the operation section below, that the relevant VMCS pointer is valid and that the specified VMCS field is supported.

Operation

```

IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
  THEN #UD;
ELSIF in VMX non-root operation AND ("VMCS shadowing" is 0 OR source operand sets bits in range 63:15 OR
VMREAD bit corresponding to bits 14:0 of source operand is 1)1
  THEN VMexit;
ELSIF CPL > 0
  THEN #GP(0);
ELSIF (in VMX root operation AND current-VMCS pointer is not valid) OR
(in VMX non-root operation AND VMCS link pointer is not valid)
  THEN VMfailInvalid;
ELSIF source operand does not correspond to any VMCS field
  THEN VMfailValid(VMREAD/VMWRITE from/to unsupported VMCS component);
ELSE
  IF in VMX root operation
    THEN destination operand := contents of field indexed by source operand in current VMCS;
    ELSE destination operand := contents of field indexed by source operand in VMCS referenced by VMCS link pointer;
  FI;
  VMSucceed;
FI;

```

1. The VMREAD bit for a source operand is defined as follows. Let *x* be the value of bits 14:0 of the source operand and let *addr* be the VMREAD-bitmap address. The corresponding VMREAD bit is in bit position *x* & 7 of the byte at physical address *addr* | (*x* » 3).

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If a memory destination operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the destination operand is located in a read-only data segment or any code segment.
#PF(fault-code)	If a page fault occurs in accessing a memory destination operand.
#SS(0)	If a memory destination operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMREAD instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMREAD instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMREAD instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory destination operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing a memory destination operand.
#SS(0)	If the memory destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

VMRESUME—Resume Virtual Machine

See VMLAUNCH/VMRESUME—Launch/Resume Virtual Machine.

VMWRITE—Write Field to Virtual-Machine Control Structure

Opcode/ Instruction	Op/En	Description
NP OF 79 VMWRITE r64, r/m64	RM	Writes a specified VMCS field (in 64-bit mode).
NP OF 79 VMWRITE r32, r/m32	RM	Writes a specified VMCS field (outside 64-bit mode).

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	NA	NA

Description

Writes the contents of a primary source operand (register or memory) to a specified field in a VMCS. In VMX root operation, the instruction writes to the current VMCS. If executed in VMX non-root operation, the instruction writes to the VMCS referenced by the VMCS link pointer field in the current VMCS.

The VMCS field is specified by the VMCS-field encoding contained in the register secondary source operand. Outside IA-32e mode, the secondary source operand is always 32 bits, regardless of the value of CS.D. In 64-bit mode, the secondary source operand has 64 bits.

The effective size of the primary source operand, which may be a register or in memory, is always 32 bits outside IA-32e mode (the setting of CS.D is ignored with respect to operand size) and 64 bits in 64-bit mode. If the VMCS field specified by the secondary source operand is shorter than this effective operand size, the high bits of the primary source operand are ignored. If the VMCS field is longer, then the high bits of the field are cleared to 0.

Note that any faults resulting from accessing a memory source operand occur after determining, in the operation section below, that the relevant VMCS pointer is valid but before determining if the destination VMCS field is supported.

Operation

IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)

THEN #UD;

ELSIF in VMX non-root operation AND (“VMCS shadowing” is 0 OR secondary source operand sets bits in range 63:15 OR VMWRITE bit corresponding to bits 14:0 of secondary source operand is 1)¹

THEN VMexit;

ELSIF CPL > 0

THEN #GP(0);

ELSIF (in VMX root operation AND current-VMCS pointer is not valid) OR

(in VMX non-root operation AND VMCS-link pointer is not valid)

THEN VMfailInvalid;

ELSIF secondary source operand does not correspond to any VMCS field

THEN VMfailValid(VMREAD/VMWRITE from/to unsupported VMCS component);

ELSIF VMCS field indexed by secondary source operand is a VM-exit information field AND processor does not support writing to such fields²

THEN VMfailValid(VMWRITE to read-only VMCS component);

ELSE

1. The VMWRITE bit for a secondary source operand is defined as follows. Let *x* be the value of bits 14:0 of the secondary source operand and let *addr* be the VMWRITE-bitmap address. The corresponding VMWRITE bit is in bit position *x* & 7 of the byte at physical address *addr* | (*x* >> 3).

2. Software can discover whether these fields can be written by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

IF in VMX root operation

THEN field indexed by secondary source operand in current VMCS := primary source operand;

ELSE field indexed by secondary source operand in VMCS referenced by VMCS link pointer := primary source operand;

FI;

VMsucceed;

FI;

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0. If a memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing a memory source operand.
#SS(0)	If a memory source operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If not in VMX operation.

Real-Address Mode Exceptions

#UD	The VMWRITE instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMWRITE instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMWRITE instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0. If the memory source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing a memory source operand.
#SS(0)	If the memory source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

VMXOFF—Leave VMX Operation

Opcode/ Instruction	Op/En	Description
OF 01 C4 VMXOFF	Z0	Leaves VMX operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	NA	NA	NA	NA

Description

Takes the logical processor out of VMX operation, unblocks INIT signals, conditionally re-enables A20M, and clears any address-range monitoring.¹

Operation

```

IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0 or in SEAM root operation
    THEN #GP(0);
ELSIF dual-monitor treatment of SMIs and SMM is active
    THEN VMfail(VMXOFF under dual-monitor treatment of SMIs and SMM);
ELSE
    leave VMX operation;
    unblock INIT;
    IF IA32_SMM_MONITOR_CTL[2] = 02
        THEN unblock SMIs;
    IF outside SMX operation3
        THEN unblock and enable A20M;
    FI;
    IF the processor supports SEAM
        THEN ensure that data for each active SEAM VMCS is in the memory of the corresponding VMCS region;
    FI;
    clear address-range monitoring;
    VMsucceed;
FI;
    
```

Flags Affected

See the operation section and Section 32.2.

1. See the information on MONITOR/MWAIT in Chapter 10, “Multiple-Processor Management,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.
2. Setting IA32_SMM_MONITOR_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register’s value bit (bit 0). Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine whether this is allowed.
3. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 6, “Safer Mode Extensions Reference.”

Protected Mode Exceptions

- #GP(0) If executed in VMX root operation with CPL > 0.
If in SEAM root operation.
- #UD If executed outside VMX operation.

Real-Address Mode Exceptions

- #UD The VMXOFF instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

- #UD The VMXOFF instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

- #UD The VMXOFF instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

- #GP(0) If executed in VMX root operation with CPL > 0.
If in SEAM root operation.
- #UD If executed outside VMX operation.

VMXON—Enter VMX Operation

Opcode/ Instruction	Op/En	Description
F3 0F C7 /6 VMXON m64	M	Enter VMX root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	NA	NA	NA

Description

Puts the logical processor in VMX operation with no current VMCS, blocks INIT signals, disables A20M, and clears any address-range monitoring established by the MONITOR instruction.¹

The operand of this instruction is a 4KB-aligned physical address (the VMXON pointer) that references the VMXON region, which the logical processor may use to support VMX operation. This operand is always 64 bits and is always in memory.

Operation

IF (register operand) or (CR0.PE = 0) or (CR4.VMXE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
THEN #UD;

ELSIF not in VMX operation

THEN

IF (CPL > 0) or (in A20M mode) or

(the values of CR0 and CR4 are not supported in VMX operation; see Section 25.8) or

(bit 0 (lock bit) of IA32_FEATURE_CONTROL MSR is clear) or

(in SMX operation² and bit 1 of IA32_FEATURE_CONTROL MSR is clear) or

(outside SMX operation and bit 2 of IA32_FEATURE_CONTROL MSR is clear)

THEN #GP(0);

ELSE

addr := contents of 64-bit in-memory source operand;

IF addr is not 4KB-aligned or

addr sets any bits beyond the physical-address width^{3,4}

THEN VMfailInvalid;

ELSE

rev := 32 bits located at physical address addr;

IF rev[30:0] ≠ VMCS revision identifier supported by processor OR rev[31] = 1

THEN VMfailInvalid;

ELSE

1. See the information on MONITOR/MWAIT in Chapter 10, “Multiple-Processor Management,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

2. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference.”

3. If IA32_VMX_BASIC[48] is read as 1, VMfailInvalid occurs if addr sets any bits in the range 63:32; see Appendix A.1.

4. Usually, the processor’s physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36]. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

```

        current-VMCS pointer := FFFFFFFF_FFFFFFFFH;
        enter VMX operation;
        block INIT signals;
        block and disable A20M;
        clear address-range monitoring;
        IF the processor supports Intel PT but does not allow it to be used in VMX operation1
            THEN IA32_RTIT_CTL.TraceEn := 0;
        FI;
        VMsucceed;
    FI;
FI;
    FI;
    ELSIF in VMX non-root operation
        THEN VMexit;
    ELSIF CPL > 0
        THEN #GP(0);
        ELSE VMfail("VMXON executed in VMX root operation");
    FI;
FI;

```

Flags Affected

See the operation section and Section 32.2.

Protected Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL>0 or with invalid CR0 or CR4 fixed bits. If executed in A20M mode. If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit. If the DS, ES, FS, or GS register contains an unusable segment. If the source operand is located in an execute-only code segment. If the value of the IA32_FEATURE_CONTROL MSR does not support entry to VMX operation in the current processor mode.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit. If the SS register contains an unusable segment.
#UD	If operand is a register. If executed with CR4.VMXE = 0.

Real-Address Mode Exceptions

#UD	The VMXON instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMXON instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMXON instruction is not recognized in compatibility mode.

1. Software should read the VMX capability MSR IA32_VMX_MISC to determine whether the processor allows Intel PT to be used in VMX operation (see Appendix A.6).

64-Bit Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL > 0 or with invalid CR0 or CR4 fixed bits. If executed in A20M mode. If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form. If the value of the IA32_FEATURE_CONTROL MSR does not support entry to VMX operation in the current processor mode.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register. If executed with CR4.VMXE = 0.

32.4 VM INSTRUCTION ERROR NUMBERS

For certain error conditions, the VM-instruction error field is loaded with an error number to indicate the source of the error. Table 32-1 lists VM-instruction error numbers.

Table 32-1. VM-Instruction Error Numbers

Error Number	Description
1	VMCALL executed in VMX root operation
2	VMCLEAR with invalid physical address
3	VMCLEAR with VMXON pointer
4	VMLAUNCH with non-clear VMCS
5	VMRESUME with non-launched VMCS
6	VMRESUME after VMXOFF (VMXOFF and VMXON between VMLAUNCH and VMRESUME) ^a
7	VM entry with invalid control field(s) ^{b,c}
8	VM entry with invalid host-state field(s) ^b
9	VMPTRLD with invalid physical address
10	VMPTRLD with VMXON pointer
11	VMPTRLD with incorrect VMCS revision identifier
12	VMREAD/VMWRITE from/to unsupported VMCS component
13	VMWRITE to read-only VMCS component
15	VMXON executed in VMX root operation
16	VM entry with invalid executive-VMCS pointer ^b
17	VM entry with non-launched executive VMCS ^b
18	VM entry with executive-VMCS pointer not VMXON pointer (when attempting to deactivate the dual-monitor treatment of SMIs and SMM) ^b
19	VMCALL with non-clear VMCS (when attempting to activate the dual-monitor treatment of SMIs and SMM)
20	VMCALL with invalid VM-exit control fields
22	VMCALL with incorrect MSEG revision identifier (when attempting to activate the dual-monitor treatment of SMIs and SMM)
23	VMXOFF under dual-monitor treatment of SMIs and SMM
24	VMCALL with invalid SMM-monitor features (when attempting to activate the dual-monitor treatment of SMIs and SMM)

Table 32-1. VM-Instruction Error Numbers (Contd.)

Error Number	Description
25	VM entry with invalid VM-execution control fields in executive VMCS (when attempting to return from SMM) ^{b,c}
26	VM entry with events blocked by MOV SS.
28	Invalid operand to INVEPT/INVVPID.

NOTES:

- a. Earlier versions of this manual described this error as “VMRESUME with a corrupted VMCS”.
- b. VM-entry checks on control fields and host-state fields may be performed in any order. Thus, an indication by error number of one cause does not imply that there are not also other errors. Different processors may give different error numbers for the same VMCS.
- c. Error number 7 is not used for VM entries that return from SMM that fail due to invalid VM-execution control fields in the executive VMCS. Error number 25 is used for these cases.

26. Updates to Chapter 33, Volume 3C

Change bars and violet text show changes to Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- In Section 33.15.4.1, "Checks on the Executive-VMCS Pointer Field", added footnote reference for the processor's physical-address width.

This chapter describes aspects of IA-64 and IA-32 architecture used in system management mode (SMM).

SMM provides an alternate operating environment that can be used to monitor and manage various system resources for more efficient energy usage, to control system hardware, and/or to run proprietary code. It was introduced into the IA-32 architecture in the Intel386 SL processor (a mobile specialized version of the Intel386 processor). It is also available in the Pentium M, Pentium 4, Intel Xeon, P6 family, and Pentium and Intel486 processors (beginning with the enhanced versions of the Intel486 SL and Intel486 processors).

33.1 SYSTEM MANAGEMENT MODE OVERVIEW

SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by system firmware, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that operates transparently to the operating system or executive and software applications.

When SMM is invoked through a system management interrupt (SMI), the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment defined by a new address space. The system management software executive (SMI handler) starts execution in that environment, and the critical code and data of the SMI handler reside in a physical memory region (SMRAM) within that address space. While in SMM, the processor executes SMI handler code to perform operations such as powering down unused disk drives or monitors, executing proprietary code, or placing the whole system in a suspended state. When the SMI handler has completed its operations, it executes a resume (RSM) instruction. This instruction causes the processor to reload the saved context of the processor, switch back to protected or real mode, and resume executing the interrupted application or operating-system program or task.

The following SMM mechanisms make it transparent to applications programs and operating systems:

- The only way to enter SMM is by means of an SMI.
- The processor executes SMM code in a separate address space that can be made inaccessible from the other operating modes.
- Upon entering SMM, the processor saves the context of the interrupted program or task.
- All interrupts normally handled by the operating system are disabled upon entry into SMM.
- The RSM instruction can be executed only in SMM.

Section 33.3 describes transitions into and out of SMM. The execution environment after entering SMM is in real-address mode with paging disabled ($CR0.PE = CR0.PG = 0$). In this initial execution environment, the SMI handler can address up to 4 GBytes of memory and can execute all I/O and system instructions. Section 33.5 describes in detail the initial SMM execution environment for an SMI handler and operation within that environment. The SMI handler may subsequently switch to other operating modes while remaining in SMM.

NOTES

Software developers should be aware that, even if a logical processor was using the physical-address extension (PAE) mechanism (introduced in the P6 family processors) or was in IA-32e mode before an SMI, this will not be the case after the SMI is delivered. This is because delivery of an SMI disables paging (see Table 33-4). (This does not apply if the dual-monitor treatment of SMIs and SMM is active; see Section 33.15.)

33.1.1 System Management Mode and VMX Operation

Traditionally, SMM services system management interrupts and then resumes program execution (back to the software stack consisting of executive and application software; see Section 33.2 through Section 33.13).

A virtual machine monitor (VMM) using VMX can act as a host to multiple virtual machines and each virtual machine can support its own software stack of executive and application software. On processors that support VMX, virtual-machine extensions may use system-management interrupts (SMIs) and system-management mode (SMM) in one of two ways:

- **Default treatment.** System firmware handles SMIs. The processor saves architectural states and critical states relevant to VMX operation upon entering SMM. When the firmware completes servicing SMIs, it uses RSM to resume VMX operation.
- **Dual-monitor treatment.** Two VM monitors collaborate to control the servicing of SMIs: one VMM operates outside of SMM to provide basic virtualization in support for guests; the other VMM operates inside SMM (while in VMX operation) to support system-management functions. The former is referred to as **executive monitor**, the latter **SMM-transfer monitor (STM)**.¹

The default treatment is described in Section 33.14, “Default Treatment of SMIs and SMM with VMX Operation and SMX Operation.” Dual-monitor treatment of SMM is described in Section 33.15, “Dual-Monitor Treatment of SMIs and SMM.”

33.2 SYSTEM MANAGEMENT INTERRUPT (SMI)

The only way to enter SMM is by signaling an SMI through the SMI# pin on the processor or through an SMI message received through the APIC bus. The SMI is a nonmaskable external interrupt that operates independently from the processor’s interrupt- and exception-handling mechanism and the local APIC. The SMI takes precedence over an NMI and a maskable interrupt. SMM is non-reentrant; that is, the SMI is disabled while the processor is in SMM.

NOTES

In the Pentium 4, Intel Xeon, and P6 family processors, when a processor that is designated as an application processor during an MP initialization sequence is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However if a SMI is received while an application processor is in the wait for SIPI mode, the SMI will be pended. The processor then responds on receipt of a SIPI by immediately servicing the pended SMI and going into SMM before handling the SIPI.

An SMI may be blocked for one instruction following execution of STI, MOV to SS, or POP into SS.

33.3 SWITCHING BETWEEN SMM AND THE OTHER PROCESSOR OPERATING MODES

Figure 2-3 shows how the processor moves between SMM and the other processor operating modes (protected, real-address, and virtual-8086). Signaling an SMI while the processor is in real-address, protected, or virtual-8086 modes always causes the processor to switch to SMM. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

33.3.1 Entering SMM

The processor always handles an SMI on an architecturally defined “interruptible” point in program execution (which is commonly at an IA-32 architecture instruction boundary). When the processor receives an SMI, it waits for all instructions to retire and for all stores to complete. The processor then saves its current context in SMRAM (see Section 33.4), enters SMM, and begins to execute the SMI handler.

1. The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

Upon entering SMM, the processor signals external hardware that SMI handling has begun. The signaling mechanism used is implementation dependent. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is asserted each time a bus transaction is generated while the processor is in SMM. For the Pentium and Intel486 processors, the SMIACK# pin is asserted.

An SMI has a greater priority than debug exceptions and external interrupts. Thus, if an NMI, maskable hardware interrupt, or a debug exception occurs at an instruction boundary along with an SMI, only the SMI is handled. Subsequent SMI requests are not acknowledged while the processor is in SMM. The first SMI interrupt request that occurs while the processor is in SMM (that is, after SMM has been acknowledged to external hardware) is latched and serviced when the processor exits SMM with the RSM instruction. The processor will latch only one SMI while in SMM.

See Section 33.5 for a detailed description of the execution environment when in SMM.

33.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

NOTES

On processors that support the shadow-stack feature, RSM loads the SSP register from the state save image in SMRAM (see Table 33-3). The value is made canonical by sign-extension before loading it into SSP.

On processors that support Intel® SGX, RSM consults the "incident to enclave mode" bit in SMRAM (see Table 33-3). If it is set, a subsequent VM exit due to an event pending following RSM will indicate that it was incident to enclave mode. See Section 29.2.1 for details.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACK# pin is desasserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 33.4.1). CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- CR4.PCIDE would be set to 1 and IA32_EFER.LMA to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.
- CR4.CET would be set to 1 and CR0.WP to 0.

In the shutdown state, Intel processors stop executing instructions until a RESET#, INIT# or NMI# is asserted. While Pentium family processors recognize the SMI# signal in shutdown state, P6 family and Intel486 processors do not. Intel does not support using SMI# to recover from shutdown states for any processor family; the response of processors in this circumstance is not well defined. On Pentium 4 and later processors, shutdown will inhibit INTR and A20M but will not change any of the other inhibits. On these processors, NMIs will be inhibited if no action is taken in the SMI handler to uninhibit them (see Section 33.8).

If the processor is in the HALT state when the SMI is received, the processor handles the return from SMM slightly differently (see Section 33.10). Also, the SMBASE address can be changed on a return from SMM (see Section 33.11).

33.4 SMRAM

Upon entering SMM, the processor switches to a new address space. Because paging is disabled upon entering SMM, this initial address space maps all memory accesses to the low 4 GBytes of the processor's physical address space. The SMI handler's critical code and data reside in a memory region referred to as system-management RAM (SMRAM). The processor uses a pre-defined region within SMRAM to save the processor's pre-SMI context. SMRAM can also be used to store system management information (such as the system configuration and specific information about powered-down devices) and OEM-specific information.

The default SMRAM size is 64 KBytes beginning at a base physical address in physical memory called the SMBASE (see Figure 33-1). The SMBASE default value following a hardware reset is 30000H. The processor looks for the first instruction of the SMI handler at the address [SMBASE + 8000H]. It stores the processor's state in the area from [SMBASE + FE00H] to [SMBASE + FFFFH]. See Section 33.4.1 for a description of the mapping of the state save area.

The system logic is minimally required to decode the physical address range for the SMRAM from [SMBASE + 8000H] to [SMBASE + FFFFH]. A larger area can be decoded if needed. The size of this SMRAM can be between 32 KBytes and 4 GBytes.

The location of the SMRAM can be changed by changing the SMBASE value (see Section 33.11). It should be noted that all processors in a multiple-processor system are initialized with the same SMBASE value (30000H). Initialization software must sequentially place each processor in SMM and change its SMBASE so that it does not overlap those of other processors.

The actual physical location of the SMRAM can be in system memory or in a separate RAM memory. The processor generates an SMI acknowledge transaction (P6 family processors) or asserts the SMIACK# pin (Pentium and Intel486 processors) when the processor receives an SMI (see Section 33.3.1).

System logic can use the SMI acknowledge transaction or the assertion of the SMIACK# pin to decode accesses to the SMRAM and redirect them (if desired) to specific SMRAM memory. If a separate RAM memory is used for SMRAM, system logic should provide a programmable method of mapping the SMRAM into system memory space when the processor is not in SMM. This mechanism will enable start-up procedures to initialize the SMRAM space (that is, load the SMI handler) before executing the SMI handler during SMM.

33.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 33-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

Some of the registers in the SMRAM state save area (marked YES in column 3) may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). An SMI handler should not rely on any values stored in an area that is marked as reserved.

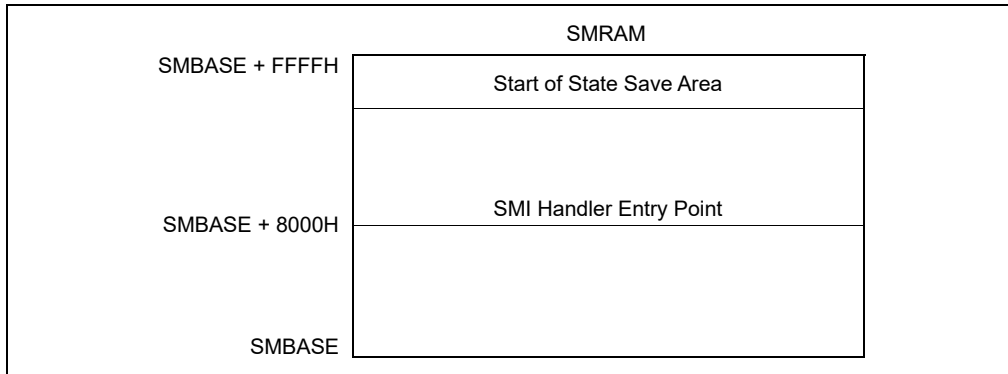


Figure 33-1. SMRAM Usage

Table 33-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR ¹	No
7FC0H	Reserved	No
7FBCH	GS ¹	No
7FB8H	FS ¹	No
7FB4H	DS ¹	No
7FB0H	SS ¹	No
7FACH	CS ¹	No
7FA8H	ES ¹	No
7FA4H	I/O State Field, see Section 33.7	No
7FA0H	I/O Memory Address Field, see Section 33.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes

Table 33-1. SMRAM State Save Map (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s when entering SMM).
- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

If an SMI request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to nonvolatile memory.

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The x87 FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium processors) or test registers TR3 through TR7 (for the Pentium and Intel486 processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The microcode update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor, it must also save and restore them.

NOTES

A small subset of the MSRs (such as, the time-stamp counter and performance-monitoring counters) are not arbitrarily writable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers.

Operating system developers should be aware of this fact and ensure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

33.4.1.1 SMRAM State Save Map and Intel 64 Architecture

When the processor initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area on an Intel 64 processor at [SMBASE + 8000H + 7FFFH] and extends to [SMBASE + 8000H + 7C00H].

Support for Intel 64 architecture is reported by CPUID.80000001:EDX[29] = 1. The layout of the SMRAM state save map is shown in Table 33-3.

Additionally, the SMRAM state save map shown in Table 33-3 also applies to processors with the following CPUID signatures listed in Table 33-2, irrespective of the value in CPUID.80000001H:EDX[29].

Table 33-2. Processor Signatures and 64-bit SMRAM State Save Map Format

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9xxx, Intel Core 2 Duo processors E8000, T9000,
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_1CH	45 nm Intel Atom® processors

Table 33-3. SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CR0	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL ¹	No
7FC0H	LDTR SEL ¹	No
7FBCH	GS SEL ¹	No
7FB8H	FS SEL ¹	No
7FB4H	DS SEL ¹	No
7FB0H	SS SEL ¹	No
7FACH	CS SEL ¹	No
7FA8H	ES SEL ¹	No
7FA4H	IO_MISC	No
7F9CH	IO_MEM_ADDR	No
7F94H	RDI	Yes
7F8CH	RSI	Yes
7F84H	RBP	Yes
7F7CH	RSP	Yes
7F74H	RBX	Yes
7F6CH	RDX	Yes
7F64H	RCX	Yes
7F5CH	RAX	Yes
7F54H	R8	Yes
7F4CH	R9	Yes

Table 33-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F44H	R10	Yes
7F3CH	R11	Yes
7F34H	R12	Yes
7F2CH	R13	Yes
7F24H	R14	Yes
7F1CH	R15	Yes
7F1BH-7F04H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7EE4H	Reserved	No
7EE0H	Bit 0 is set if the SMI was incident to VMX non-root operation and the “enable EPT” VM-execution control was 1. Bit 1 is set if the SMI was incident to enclave mode.	No
7ED8H	Value of EPTP VM-execution control field	No
7ED7H - 7ECC0H	Reserved	No
7EC8H	SSP	Yes
7EC7H - 7EA0H	Reserved	No
7E9CH	LDT Base (lower 32 bits)	No
7E98H	Reserved	No
7E94H	IDT Base (lower 32 bits)	No
7E90H	Reserved	No
7E8CH	GDT Base (lower 32 bits)	No
7E8BH - 7E48H	Reserved	No
7E40H	CR4 (64 bits)	No
7E3FH - 7DF0H	Reserved	No
7DE8H	IO_RIP	Yes
7DE7H - 7DDCH	Reserved	No
7DD8H	IDT Base (Upper 32 bits)	No
7DD4H	LDT Base (Upper 32 bits)	No
7DD0H	GDT Base (Upper 32 bits)	No
7DCFH - 7C00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

33.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and

in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Any of the following three methods of locating the SMRAM in system memory will guarantee cache coherency.

- Place the SMRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SMRAM can be designated as cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.
- Place the SMRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SMRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method maintains cache coherency, but incurs the overhead of two complete cache flushes.

For Pentium 4, Intel Xeon, and P6 family processors, a combination of the first two methods of locating the SMRAM is recommended. Here the SMRAM is split between an overlapping and a dedicated region of memory. Upon entering SMM, the SMRAM space that is accessed overlaps video memory (typically located in low memory). This SMRAM section is designated as UC memory. The initial SMM code then jumps to a second SMRAM section that is located in a dedicated region of system memory (typically in high memory). This SMRAM section can be cached for optimum processor performance.

For systems that explicitly flush the caches upon entering SMM (the third method described above), the cache flush can be accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM (generally initiated by asserting the SMI# pin). The priorities of the FLUSH# and SMI# pins are such that the FLUSH# is serviced first. To guarantee this behavior, the processor requires that the following constraints on the interaction of FLUSH# and SMI# be met. In a system where the FLUSH# and SMI# pins are synchronous and the set up and hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first.

Upon leaving SMM (for systems that explicitly flush the caches), the WBINVD instruction should be executed prior to leaving SMM to flush the caches.

NOTES

In systems based on the Pentium processor that use the FLUSH# pin to write back and invalidate cache contents before entering SMM, the processor will prefetch at least one cache line in between when the Flush Acknowledge cycle is run and the subsequent recognition of SMI# and the assertion of SMIACK#.

It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive to the Pentium processor.

33.4.2.1 System Management Range Registers (SMRR)

SMI handler code and data stored by SMM code resides in SMRAM. The SMRR interface is an enhancement in Intel 64 architecture to limit cacheable reference of addresses in SMRAM to code running in SMM. The SMRR interface can be configured only by code running in SMM. Details of SMRR is described in Section 13.11.2.4.

33.5 SMI HANDLER EXECUTION ENVIRONMENT

Section 33.5.1 describes the initial execution environment for an SMI handler. An SMI handler may re-configure its execution environment to other supported operating modes. Section 33.5.2 discusses modifications an SMI handler can make to its execution environment. Section 33.5.3 discusses Control-flow Enforcement Technology (CET) interactions in the environment.

33.5.1 Initial SMM Execution Environment

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 33-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable address space ranges from 0 to FFFFFFFFH (4 GBytes).
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.
- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

Table 33-4. Processor Register Initialization in SMM

Register	Contents
General-purpose registers	Undefined
EFLAGS	00000002H
EIP	00008000H
CS selector	SMM Base shifted right 4 bits (default 3000H)
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	00000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFH
CR0	PE, EM, TS, and PG flags set to 0; others unmodified
CR4	Cleared to zero
DR6	Undefined
DR7	00000400H

- Near jumps and calls can be made to anywhere in the 4-GByte address space if a 32-bit operand-size override prefix is used. Due to the real-address-mode style of base-address formation, a far call or jump cannot transfer control to a segment with a base address of more than 20 bits (1 MByte). However, since the segment limit in SMM is 4 GBytes, offsets into a segment that go beyond the 1-MByte limit are allowed when using 32-bit operand-size override prefixes. Any program control transfer that does not have a 32-bit operand-size override prefix truncates the EIP value to the 16 low-order bits.
- Data and the stack can be located anywhere in the 4-GByte address space, but can be accessed only with a 32-bit address-size override if they are located above 1 MByte. As with the code segment, the base address for a data or stack segment cannot be more than 20 bits.

The value in segment register CS is automatically set to the default of 30000H for the SMBASE shifted 4 bits to the right; that is, 3000H. The EIP register is set to 8000H. When the EIP value is added to shifted CS value (the SMBASE), the resulting linear address points to the first instruction of the SMI handler.

The other segment registers (DS, SS, ES, FS, and GS) are cleared to 0 and their segment limits are set to 4 GBytes. In this state, the SMRAM address space may be treated as a single flat 4-GByte linear address space. If a segment register is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base (hidden part of the segment register). The limits and attributes are not modified.

Maskable hardware interrupts, exceptions, NMI interrupts, SMI interrupts, A20M interrupts, single-step traps, breakpoint traps, and INIT operations are inhibited when the processor enters SMM. Maskable hardware interrupts, exceptions, single-step traps, and breakpoint traps can be enabled in SMM if the SMM execution environment provides and initializes an interrupt table and the necessary interrupt and exception handlers (see Section 33.6).

33.5.2 SMI Handler Operating Mode Switching

Within SMM, an SMI handler may change the processor's operating mode (e.g., to enable PAE paging, enter 64-bit mode, etc.) after it has made proper preparation and initialization to do so. For example, if switching to 32-bit protected mode, the SMI handler should follow the guidelines provided in Chapter 11, "Processor Management and Initialization." If the SMI handler does wish to change operating mode, it is responsible for executing the appropriate mode-transition code after each SMI.

It is recommended that the SMI handler make use of all means available to protect the integrity of its critical code and data. In particular, it should use the system-management range register (SMRR) interface if it is available (see Section 11.11.2.4). The SMRR interface can protect only the first 4 GBytes of the physical address space. The SMI handler should take that fact into account if it uses operating modes that allow access to physical addresses beyond that 4-GByte limit (e.g., PAE paging or 64-bit mode).

Execution of the RSM instruction restores the pre-SMI processor state from the SMRAM state-state map (see Section 33.4.1) into which it was stored when the processor entered SMM. (The SMBASE field in the SMRAM state-save map does not determine the state following RSM but rather the initial environment following the next entry to SMM.) Any required change to operating mode is performed by the RSM instruction; there is no need for the SMI handler to change modes explicitly prior to executing RSM.

33.5.3 Control-flow Enforcement Technology Interactions

On processors that support CET shadow stacks, when the processor enters SMM, the processor saves the SSP register to the SMRAM state save area (see Table 33-3) and clears CR4.CET to 0. Thus, the initial execution environment of the SMI handler has CET disabled and all of the CET state of the interrupted program is still in the machine. An SMM that uses CET is required to save the interrupted program's CET state and restore the CET state prior to exiting SMM.

33.6 EXCEPTIONS AND INTERRUPTS WITHIN SMM

When the processor enters SMM, all hardware interrupts are disabled in the following manner:

- The IF flag in the EFLAGS register is cleared, which inhibits maskable hardware interrupts from being generated.
- The TF flag in the EFLAGS register is cleared, which disables single-step traps.
- Debug register DR7 is cleared, which disables breakpoint traps. (This action prevents a debugger from accidentally breaking into an SMI handler if a debug breakpoint is set in normal address space that overlays code or data in SMRAM.)
- NMI, SMI, and A20M interrupts are blocked by internal SMM logic. (See Section 33.8 for more information about how NMIs are handled in SMM.)

Software-invoked interrupts and exceptions can still occur, and maskable hardware interrupts can be enabled by setting the IF flag. Intel recommends that SMM code be written in so that it does not invoke software interrupts (with the INT *n*, INTO, INT1, INT3, or BOUND instructions) or generate exceptions.

If the SMI handler requires interrupt and exception handling, an SMM interrupt table and the necessary exception and interrupt handlers must be created and initialized from within SMM. Until the interrupt table is correctly initialized (using the LIDT instruction), exceptions and software interrupts will result in unpredictable processor behavior.

The following restrictions apply when designing SMM interrupt and exception-handling facilities:

- The interrupt table should be located at linear address 0 and must contain real-address mode style interrupt vectors (4 bytes containing CS and IP).
- Due to the real-address mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more than 20 bits.
- An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).

- When an exception or interrupt occurs, only the 16 least-significant bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One solution to this problem is for a handler to adjust the return address on the stack.)
- The SMBASE relocation feature affects the way the processor will return from an interrupt or exception generated while the SMI handler is executing. For example, if the SMBASE is relocated to above 1 MByte, but the exception handlers are below 1 MByte, a normal return to the SMI handler is not possible. One solution is to provide the exception handler with a mechanism for calculating a return address above 1 MByte from the 16-bit return address on the stack, then use a 32-bit far call to return to the interrupted procedure.
- If an SMI handler needs access to the debug trap facilities, it must ensure that an SMM accessible debug handler is available and save the current contents of debug registers DR0 through DR3 (for later restoration). Debug registers DR0 through DR3 and DR7 must then be initialized with the appropriate values.
- If an SMI handler needs access to the single-step mechanism, it must ensure that an SMM accessible single-step handler is available, and then set the TF flag in the EFLAGS register.
- If the SMI design requires the processor to respond to maskable hardware interrupts or software-generated interrupts while in SMM, it must ensure that SMM accessible interrupt handlers are available and then set the IF flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, so they do not need to be enabled.

33.7 MANAGING SYNCHRONOUS AND ASYNCHRONOUS SYSTEM MANAGEMENT INTERRUPTS

When coding for a multiprocessor system or a system with Intel HT Technology, it was not always possible for an SMI handler to distinguish between a synchronous SMI (triggered during an I/O instruction) and an asynchronous SMI. To facilitate the discrimination of these two events, incremental state information has been added to the SMM state save map.

Processors that have an SMM revision ID of 30004H or higher have the incremental state information described below.

33.7.1 I/O State Implementation

Within the extended SMM state save map, a bit (IO_SMI) is provided that is set only when an SMI is either taken immediately after a *successful* I/O instruction or is taken after a *successful* iteration of a REP I/O instruction (the *successful* notion pertains to the processor point of view; not necessarily to the corresponding platform function). When set, the IO_SMI bit provides a strong indication that the corresponding SMI was synchronous. In this case, the SMM State Save Map also supplies the port address of the I/O operation. The IO_SMI bit and the I/O Port Address may be used in conjunction with the information logged by the platform to confirm that the SMI was indeed synchronous.

The IO_SMI bit by itself is a strong indication, not a guarantee, that the SMI is synchronous. This is because an asynchronous SMI might coincidentally be taken after an I/O instruction. In such a case, the IO_SMI bit would still be set in the SMM state save map.

Information characterizing the I/O instruction is saved in two locations in the SMM State Save Map (Table 33-5). The IO_SMI bit also serves as a valid bit for the rest of the I/O information fields. The contents of these I/O information fields are not defined when the IO_SMI bit is not set.

Table 33-5. I/O Instruction Information in the SMM State Save Map

State (SMM Rev. ID: 30004H or higher)	Format									
	31	16	15	8	7	4	3	1	0	

Table 33-5. I/O Instruction Information in the SMM State Save Map

I/O State Field SMRAM offset 7FA4	I/O Port	Reserved	I/O Type	I/O Length	IO_SMI
	31				0
I/O Memory Address Field SMRAM offset 7FA0	I/O Memory Address				

When IO_SMI is set, the other fields may be interpreted as follows:

- I/O length:
 - 001 – Byte
 - 010 – Word
 - 100 – Dword
- I/O instruction type (Table 33-6)

Table 33-6. I/O Instruction Type Encodings

Instruction	Encoding
IN Immediate	1001
IN DX	0001
OUT Immediate	1000
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

33.8 NMI HANDLING WHILE IN SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMI handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same “real mode” manner in which they are handled outside of SMM.

Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI interrupts from inside of SMM. This behavior is implementation specific for the Pentium processor and is not part of the IA-32 architecture.

33.9 SMM REVISION IDENTIFIER

The SMM revision identifier field is used to indicate the version of SMM and the SMM extensions that are supported by the processor (see Figure 33-2). The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at offset 7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture.

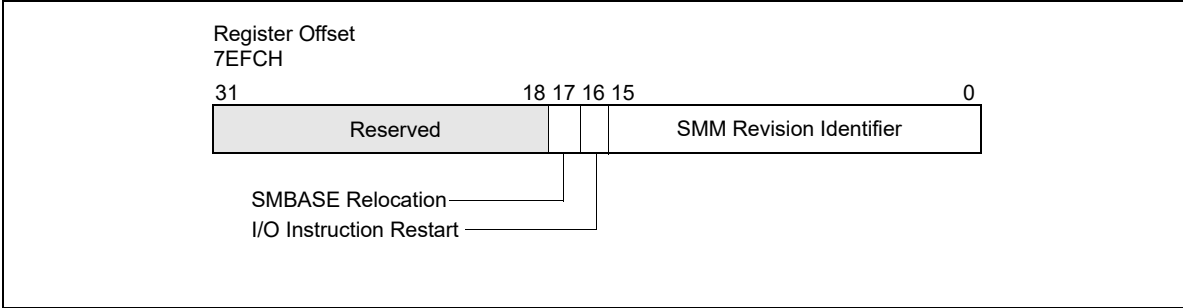


Figure 33-2. SMM Revision Identifier

The upper word of the SMM revision identifier refers to the extensions available. If the I/O instruction restart flag (bit 16) is set, the processor supports the I/O instruction restart (see Section 33.12); if the SMBASE relocation flag (bit 17) is set, SMRAM base address relocation is supported (see Section 33.11).

33.10 AUTO HALT RESTART

If the processor is in a HALT state (due to the prior execution of a HLT instruction) when it receives an SMI, the processor records the fact in the auto HALT restart flag in the saved processor state (see Figure 33-3). (This flag is located at offset 7F02H and bit 0 in the state save area of the SMRAM.)

If the processor sets the auto HALT restart flag upon entering SMM (indicating that the SMI occurred when the processor was in the HALT state), the SMI handler has two options:

- It can leave the auto HALT restart flag set, which instructs the RSM instruction to return program control to the HLT instruction. This option in effect causes the processor to re-enter the HALT state after handling the SMI. (This is the default operation.)
- It can clear the auto HALT restart flag, which instructs the RSM instruction to return program control to the instruction following the HLT instruction.

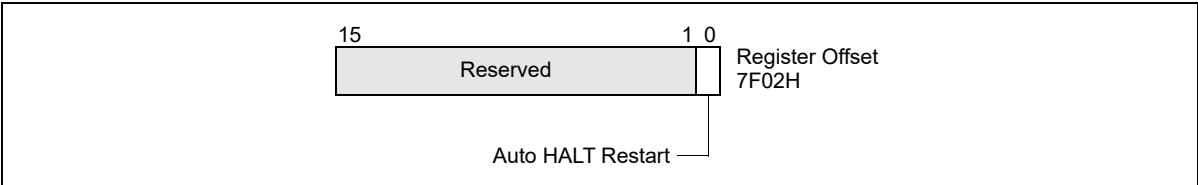


Figure 33-3. Auto HALT Restart Field

These options are summarized in Table 33-7. If the processor was not in a HALT state when the SMI was received (the auto HALT restart flag is cleared), setting the flag to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 33-7. Auto HALT Restart Flag Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
0	0	Returns to next instruction in interrupted program or task.
0	1	Unpredictable.
1	0	Returns to next instruction after HLT instruction.
1	1	Returns to HALT state.

If the HLT instruction is restarted, the processor will generate a memory access to fetch the HLT instruction (if it is

not in the internal cache), and execute a HLT bus transaction. This behavior results in multiple HLT bus transactions for the same HLT instruction.

33.10.1 Executing the HLT Instruction in SMM

The HLT instruction should not be executed during SMM, unless interrupts have been enabled by setting the IF flag in the EFLAGS register. If the processor is halted in SMM, the only event that can remove the processor from this state is a maskable hardware interrupt or a hardware reset.

33.11 SMBASE RELOCATION

The default base address for the SMRAM is 30000H. This value is contained in an internal processor register called the SMBASE register. Software can relocate the SMRAM by setting the SMBASE field in the saved state map (at offset 7EF8H) to a new value (see Figure 33-4). The RSM instruction reloads the internal SMBASE register with the value in the SMBASE field each time it exits SMM. All subsequent SMI requests will use the new SMBASE value to find the starting address for the SMI handler (at SMBASE + 8000H) and the SMRAM state save area (from SMBASE + FE00H to SMBASE + FFFFH). (The processor resets the value in its internal SMBASE register to 30000H on a RESET, but does not change it on an INIT.)

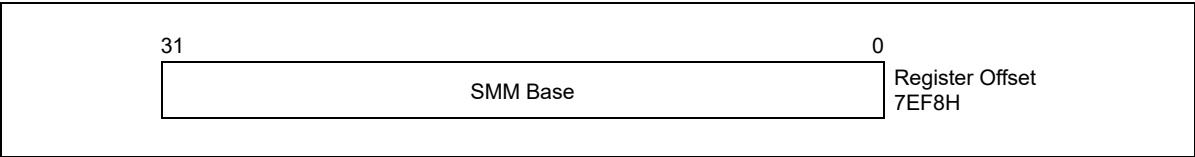


Figure 33-4. SMBASE Relocation Field

In multiple-processor systems, initialization software must adjust the SMBASE value for each processor so that the SMRAM state save areas for each processor do not overlap. (For Pentium and Intel486 processors, the SMBASE values must be aligned on a 32-KByte boundary or the processor will enter shutdown state during the execution of a RSM instruction.)

If the SMBASE relocation flag in the SMM revision identifier field is set, it indicates the ability to relocate the SMBASE (see Section 33.9).

33.12 I/O INSTRUCTION RESTART

If the I/O instruction restart flag in the SMM revision identifier field is set (see Section 33.9), the I/O instruction restart mechanism is present on the processor. This mechanism allows an interrupted I/O instruction to be re-executed upon returning from SMM mode. For example, if an I/O instruction is used to access a powered-down I/O device, a chipset supporting this device can intercept the access and respond by asserting SMI#. This action invokes the SMI handler to power-up the device. Upon returning from the SMI handler, the I/O instruction restart mechanism can be used to re-execute the I/O instruction that caused the SMI.

The I/O instruction restart field (at offset 7F00H in the SMM state-save area, see Figure 33-5) controls I/O instruction restart. When an RSM instruction is executed, if this field contains the value FFH, then the EIP register is modified to point to the I/O instruction that received the SMI request. The processor will then automatically re-execute the I/O instruction that the SMI trapped. (The processor saves the necessary machine state to ensure that re-execution of the instruction is handled coherently.)



Figure 33-5. I/O Instruction Restart Field

If the I/O instruction restart field contains the value 00H when the RSM instruction is executed, then the processor begins program execution with the instruction following the I/O instruction. (When a repeat prefix is being used, the next instruction may be the next I/O instruction in the repeat loop.) Not re-executing the interrupted I/O instruction is the default behavior; the processor automatically initializes the I/O instruction restart field to 00H upon entering SMM. Table 33-8 summarizes the states of the I/O instruction restart field.

Table 33-8. I/O Instruction Restart Field Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

The I/O instruction restart mechanism does not indicate the cause of the SMI. It is the responsibility of the SMI handler to examine the state of the processor to determine the cause of the SMI and to determine if an I/O instruction was interrupted and should be restarted upon exiting SMM. If an SMI interrupt is signaled on a non-I/O instruction boundary, setting the I/O instruction restart field to FFH prior to executing the RSM instruction will likely result in a program error.

33.12.1 Back-to-Back SMI Interrupts When I/O Instruction Restart Is Being Used

If an SMI interrupt is signaled while the processor is servicing an SMI interrupt that occurred on an I/O instruction boundary, the processor will service the new SMI request before restarting the originally interrupted I/O instruction. If the I/O instruction restart field is set to FFH prior to returning from the second SMI handler, the EIP will point to an address different from the originally interrupted I/O instruction, which will likely lead to a program error. To avoid this situation, the SMI handler must be able to recognize the occurrence of back-to-back SMI interrupts when I/O instruction restart is being used and ensure that the handler sets the I/O instruction restart field to 00H prior to returning from the second invocation of the SMI handler.

33.13 SMM MULTIPLE-PROCESSOR CONSIDERATIONS

The following should be noted when designing multiple-processor systems:

- Any processor in a multiprocessor system can respond to an SMI.
- Each processor needs its own SMRAM space. This space can be in system memory or in a separate RAM.
- The SMRAMs for different processors can be overlapped in the same memory space. The only stipulation is that each processor needs its own state save area and its own dynamic data storage area. (Also, for the Pentium and Intel486 processors, the SMBASE address must be located on a 32-KByte boundary.) Code and static data can be shared among processors. Overlapping SMRAM spaces can be done more efficiently with the P6 family processors because they do not require that the SMBASE address be on a 32-KByte boundary.
- The SMI handler will need to initialize the SMBASE for each processor.
- Processors can respond to local SMIs through their SMI# pins or to SMIs received through the APIC interface. The APIC interface can distribute SMIs to different processors.
- Two or more processors can be executing in SMM at the same time.

- When operating Pentium processors in dual processing (DP) mode, the SMI $\overline{ACT}\#$ pin is driven only by the MRM processor and should be sampled with ADS#. For additional details, see Chapter 14 of the Pentium Processor Family User's Manual, Volume 1.

SMM is not re-entrant, because the SMRAM State Save Map is fixed relative to the SMBASE. If there is a need to support two or more processors in SMM mode at the same time then each processor should have dedicated SMRAM spaces. This can be done by using the SMBASE Relocation feature (see Section 33.11).

33.14 DEFAULT TREATMENT OF SMIS AND SMM WITH VMX OPERATION AND SMX OPERATION

Under the default treatment, the interactions of SMIs and SMM with VMX operation are few. This section details those interactions. It also explains how this treatment affects SMX operation.

33.14.1 Default Treatment of SMI Delivery

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on architectural definitions. Under the default treatment, processors that support VMX operation perform SMI delivery as follows:

```

enter SMM;
save the following internal to the processor:
    CR4.VMXE
    an indication of whether the logical processor was in VMX operation (root or non-root)
IF the logical processor is in VMX operation
    THEN
        save current VMCS pointer internal to the processor;
        leave VMX operation;
        save VMX-critical state defined below;
FI;
IF the logical processor supports SMX operation
    THEN
        save internal to the logical processor an indication of whether the Intel® TXT private space is locked;
        IF the TXT private space is unlocked
            THEN lock the TXT private space;
        FI;
FI;
CR4.VMXE := 0;
perform ordinary SMI delivery:
    save processor state in SMRAM;
    set processor state to standard SMM values;1
    invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H
are invalidated for all EPT $\overline{RTA}$  values (EPT $\overline{RTA}$  is the value of bits 51:12 of EPTP; see Section 30.4);
  
```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state consists of the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM²; (3) the state of blocking by STI and by MOV SS (see Table 26-3 in Section 26.4.2); (4) the state of virtual-NMI blocking (only if the processor is in VMX non-root operation and the "virtual NMIs" VM-execution control is 1); and (5) an indication of whether an MTF VM exit is pending (see Section 27.5.2). These data may be saved internal to the processor or in the VMCS region of the current

-
1. This causes the logical processor to block INIT signals, NMIs, and SMIs.
 2. Section 33.14 and Section 33.15 use the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of these registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to the lower 32 bits of the register.

VMCS. Processors that do not support SMI recognition while there is blocking by STI or by MOV SS need not save the state of such blocking.

If the logical processor supports the 1-setting of the “enable EPT” VM-execution control and the logical processor was in VMX non-root operation at the time of an SMI, it saves the value of that control into bit 0 of the 32-bit field at offset SMBASE + 8000H + 7EE0H (SMBASE + FEE0H; see Table 33-3).¹ If the logical processor was not in VMX non-root operation at the time of the SMI, it saves 0 into that bit. If the logical processor saves 1 into that bit (it was in VMX non-root operation and the “enable EPT” VM-execution control was 1), it saves the value of the EPT pointer (EPTP) into the 64-bit field at offset SMBASE + 8000H + 7ED8H (SMBASE + FED8H).

Because SMI delivery causes a logical processor to leave VMX operation, all the controls associated with VMX non-root operation are disabled in SMM and thus cannot cause VM exits while the logical processor in SMM.

33.14.2 Default Treatment of RSM

Ordinary execution of RSM restores processor state from SMRAM. Under the default treatment, processors that support VMX operation perform RSM as follows:

```

IF VMXE = 1 in CR4 image in SMRAM
    THEN fail and enter shutdown state;
ELSE
    restore state normally from SMRAM;
    invalidate linear mappings and combined mappings associated with all VPIDs and all PCIDs; combined mappings are invalidated
for all EPTPTA values (EPTPTA is the value of bits 51:12 of EPTP; see Section 30.4);
    IF the logical processor supports SMX operation and the Intel® TXT private space was unlocked at the time of the last SMI (as
saved)
        THEN unlock the TXT private space;
    FI;
    CR4.VMXE := value stored internally;
    IF internal storage indicates that the logical processor
    had been in VMX operation (root or non-root)
        THEN
            enter VMX operation (root or non-root);
            restore VMX-critical state as defined in Section 33.14.1;
            set to their fixed values any bits in CR0 and CR4 whose values must be fixed in VMX operation (see Section 25.8);2
            IF RFLAGS.VM = 0 AND (in VMX root operation OR the “unrestricted guest” VM-execution control is 0)3
                THEN
                    CS.RPL := SS.DPL;
                    SS.RPL := SS.DPL;
            FI;
            restore current VMCS pointer;
        FI;
    leave SMM;
    IF logical processor will be in VMX operation or in SMX operation after RSM
        THEN block A20M and leave A20M mode;
    FI;
FI;

```

-
1. “Enable EPT” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, SMI functions as the “enable EPT” VM-execution control were 0. See Section 26.6.2.
 2. If the RSM is to VMX non-root operation and both the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls will be 1, CR0.PE and CR0.PG retain the values that were loaded from SMRAM regardless of what is reported in the capability MSR IA32_VMX_CR0_FIXED0.
 3. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “unrestricted guest” VM-execution control were 0. See Section 26.6.2.

RSM unblocks SMIs. It restores the state of blocking by NMI (see Table 26-3 in Section 26.4.2) as follows:

- If the RSM is not to VMX non-root operation or if the “virtual NMIs” VM-execution control will be 0, the state of NMI blocking is restored normally.
- If the RSM is to VMX non-root operation and the “virtual NMIs” VM-execution control will be 1, NMIs are not blocked after RSM. The state of virtual-NMI blocking is restored as part of VMX-critical state.

INIT signals are blocked after RSM if and only if the logical processor will be in VMX root operation.

If RSM returns a logical processor to VMX non-root operation, it re-establishes the controls associated with the current VMCS. If the “interrupt-window exiting” VM-execution control is 1, a VM exit occurs immediately after RSM if the enabling conditions apply. The same is true for the “NMI-window exiting” VM-execution control. Such VM exits occur with their normal priority. See Section 27.2.

If an MTF VM exit was pending at the time of the previous SMI, an MTF VM exit is pending on the instruction boundary following execution of RSM. The following items detail the treatment of MTF VM exits that may be pending following RSM:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these MTF VM exits. These MTF VM exits take priority over debug-trap exceptions and lower priority events.
- These MTF VM exits wake the logical processor if RSM caused the logical processor to enter the HLT state (see Section 33.10). They do not occur if the logical processor just entered the shutdown state.

33.14.3 Protection of CR4.VMXE in SMM

Under the default treatment, CR4.VMXE is treated as a reserved bit while a logical processor is in SMM. Any attempt by software running in SMM to set this bit causes a general-protection exception. In addition, software cannot use VMX instructions or enter VMX operation while in SMM.

33.14.4 VMXOFF and SMI Unblocking

The VMXOFF instruction can be executed only with the default treatment (see Section 33.15.1) and only outside SMM. If SMIs are blocked when VMXOFF is executed, VMXOFF unblocks them unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 33.15.5 for details regarding this MSR).¹ Section 33.15.7 identifies a case in which SMIs may be blocked when VMXOFF is executed.

Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine whether this is allowed.

33.15 DUAL-MONITOR TREATMENT OF SMIs AND SMM

Dual-monitor treatment is activated through the cooperation of the **executive monitor** (the VMM that operates outside of SMM to provide basic virtualization) and the **SMM-transfer monitor (STM)**; the VMM that operates inside SMM—while in VMX operation—to support system-management functions). Control is transferred to the STM through VM exits; VM entries are used to return from SMM.

The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

33.15.1 Dual-Monitor Treatment Overview

The dual-monitor treatment uses an executive monitor and an SMM-transfer monitor (STM). Transitions from the executive monitor or its guests to the STM are called **SMM VM exits** and are discussed in Section 33.15.2. SMM

1. Setting IA32_SMM_MONITOR_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register’s valid bit (bit 0).

VM exits are caused by SMIs as well as executions of VMCALL in VMX root operation. The latter allow the executive monitor to call the STM for service.

The STM runs in VMX root operation and uses VMX instructions to establish a VMCS and perform VM entries to its own guests. This is done all inside SMM (see Section 33.15.3). The STM returns from SMM, not by using the RSM instruction, but by using a VM entry that returns from SMM. Such VM entries are described in Section 33.15.4.

Initially, there is no STM and the default treatment (Section 33.14) is used. The dual-monitor treatment is not used until it is enabled and activated. The steps to do this are described in Section 33.15.5 and Section 33.15.6.

It is not possible to leave VMX operation under the dual-monitor treatment; VMXOFF will fail if executed. The dual-monitor treatment must be deactivated first. The STM deactivates dual-monitor treatment using a VM entry that returns from SMM with the “deactivate dual-monitor treatment” VM-entry control set to 1 (see Section 33.15.7).

The executive monitor configures any VMCS that it uses for VM exits to the executive monitor. SMM VM exits, which transfer control to the STM, use a different VMCS. Under the dual-monitor treatment, each logical processor uses a separate VMCS called the **SMM-transfer VMCS**. When the dual-monitor treatment is active, the logical processor maintains another VMCS pointer called the **SMM-transfer VMCS pointer**. The SMM-transfer VMCS pointer is established when the dual-monitor treatment is activated.

33.15.2 SMM VM Exits

An SMM VM exit is a VM exit that begins outside SMM and that ends in SMM.

Unlike other VM exits, SMM VM exits can begin in VMX root operation. SMM VM exits result from the arrival of an SMI outside SMM or from execution of VMCALL in VMX root operation outside SMM. Execution of VMCALL in VMX root operation causes an SMM VM exit only if the valid bit is set in the IA32_SMM_MONITOR_CTL MSR (see Section 33.15.5).

Execution of VMCALL in VMX root operation causes an SMM VM exit even under the default treatment. This SMM VM exit activates the dual-monitor treatment (see Section 33.15.6).

Differences between SMM VM exits and other VM exits are detailed in Sections 33.15.2.1 through 33.15.2.5. Differences between SMM VM exits that activate the dual-monitor treatment and other SMM VM exits are described in Section 33.15.6.

33.15.2.1 Architectural State Before a VM Exit

System-management interrupts (SMIs) that cause SMM VM exits always do so directly. They do not save state to SMRAM as they do under the default treatment.

33.15.2.2 Updating the Current-VMCS and Executive-VMCS Pointers

SMM VM exits begin by performing the following steps:

1. The executive-VMCS pointer field in the SMM-transfer VMCS is loaded as follows:
 - If the SMM VM exit commenced in VMX non-root operation, it receives the current-VMCS pointer.
 - If the SMM VM exit commenced in VMX root operation, it receives the VMXON pointer.
2. The current-VMCS pointer is loaded with the value of the SMM-transfer VMCS pointer.

The last step ensures that the current VMCS is the SMM-transfer VMCS. VM-exit information is recorded in that VMCS, and VM-entry control fields in that VMCS are updated. State is saved into the guest-state area of that VMCS. The VM-exit controls and host-state area of that VMCS determine how the VM exit operates.

33.15.2.3 Recording VM-Exit Information

SMM VM exits differ from other VM exit with regard to the way they record VM-exit information. The differences follow.

- **Exit reason.**

- Bits 15:0 of this field contain the basic exit reason. The field is loaded with the reason for the SMM VM exit: I/O SMI (an SMI arrived immediately after retirement of an I/O instruction), other SMI, or VMCALL. See Appendix C, “VMX Basic Exit Reasons.”
- SMM VM exits are the only VM exits that may occur in VMX root operation. Because the SMM-transfer monitor may need to know whether it was invoked from VMX root or VMX non-root operation, this information is stored in bit 29 of the exit-reason field (see Table 26-19 in Section 26.9.1). The bit is set by SMM VM exits from VMX root operation.
- If the SMM VM exit occurred in VMX non-root operation and an MTF VM exit was pending, bit 28 of the exit-reason field is set; otherwise, it is cleared.
- Bits 27:16 and bits 31:30 are cleared.
- **Exit qualification.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, the exit qualification contains information about the I/O instruction that retired immediately before the SMI. It has the format given in Table 33-9.

Table 33-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used.
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- **Guest linear address.** This field is used for VM exits due to SMIs that arrive immediately after the retirement of an INS or OUTS instruction for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override prefix) at the time the instruction started. If the relevant segment is not usable, the value is undefined. On processors that support Intel 64 architecture, bits 63:32 are clear if the logical processor was not in 64-bit mode before the VM exit.
- **I/O RCX, I/O RSI, I/O RDI, and I/O RIP.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, these fields receive the values that were in RCX, RSI, RDI, and RIP, respectively, before the I/O instruction executed. Thus, the value saved for I/O RIP addresses the I/O instruction.

33.15.2.4 Saving Guest State

SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area.

The value of the VMX-preemption timer is saved into the corresponding field in the guest-state area if the “save VMX-preemption timer value” VM-exit control is 1. That field becomes undefined if, in addition, either the SMM VM exit is from VMX root operation or the SMM VM exit is from VMX non-root operation and the “activate VMX-preemption timer” VM-execution control is 0.

33.15.2.5 Updating State

If an SMM VM exit is from VMX non-root operation and the “Intel PT uses guest physical addresses” VM-execution control is 1, the IA32_RTIT_CTL MSR is cleared to 00000000_00000000H.¹ This is done even if the “clear IA32_RTIT_CTL” VM-exit control is 0.

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the “virtual NMIs” VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 33.15.4).

SMM VM exits invalidate linear mappings and combined mappings associated with VPID 0000H for all PCIDs. Combined mappings for VPID 0000H are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP; see Section 30.4). (Ordinary VM exits are not required to perform such invalidation if the “enable VPID” VM-execution control is 1; see Section 29.5.5.)

33.15.3 Operation of the SMM-Transfer Monitor

Once invoked, the SMM-transfer monitor (STM) is in VMX root operation and can use VMX instructions to configure VMCSs and to cause VM entries to virtual machines supported by those structures. As noted in Section 33.15.1, the VMXOFF instruction cannot be used under the dual-monitor treatment and thus cannot be used by the STM.

The RSM instruction also cannot be used under the dual-monitor treatment. As noted in Section 27.1.3, it causes a VM exit if executed in SMM in VMX non-root operation. If executed in VMX root operation, it causes an invalid-opcode exception. The STM uses VM entries to return from SMM (see Section 33.15.4).

33.15.4 VM Entries that Return from SMM

The SMM-transfer monitor (STM) returns from SMM using a VM entry with the “entry to SMM” VM-entry control clear. VM entries that return from SMM reverse the effects of an SMM VM exit (see Section 33.15.2).

VM entries that return from SMM may differ from other VM entries in that they do not necessarily enter VMX non-root operation. If the executive-VMCS pointer field in the current VMCS contains the VMXON pointer, the logical processor remains in VMX root operation after VM entry.

For differences between VM entries that return from SMM and other VM entries see Sections 33.15.4.1 through 33.15.4.10.

33.15.4.1 Checks on the Executive-VMCS Pointer Field

VM entries that return from SMM perform the following checks on the executive-VMCS pointer field in the current VMCS:

- Bits 11:0 must be 0.
- The pointer must not set any bits beyond the processor’s physical-address width.^{2,3}

1. In this situation, the value of this MSR was saved earlier into the guest-state area. All VM exits save this MSR if the 1-setting of the “load IA32_RTIT_CTL” VM-entry control is supported (see Section 29.3.1), which must be the case if the “Intel PT uses guest physical addresses” VM-execution control is 1 (see Section 28.2.1.1).

2. Usually, the processor’s physical-address width is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), the width is reduced by the value of IA32_TME_ACTIVATE[39:36]. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

3. If IA32_VMX_BASIC[48] is read as 1, this pointer must not set any bits in the range 63:32; see Appendix A.1.

- The 32 bits located in memory referenced by the physical address in the pointer must contain the processor's VMCS revision identifier (see Section 26.2).

The checks above are performed before the checks described in Section 33.15.4.2 and before any of the following checks:

- If the "deactivate dual-monitor treatment" VM-entry control is 0 and the executive-VMCS pointer field does not contain the VMXON pointer, the launch state of the executive VMCS (the VMCS referenced by the executive-VMCS pointer field) must be launched (see Section 26.11.3).
- If the "deactivate dual-monitor treatment" VM-entry control is 1, the executive-VMCS pointer field must contain the VMXON pointer (see Section 33.15.7).¹

33.15.4.2 Checks on VM-Execution Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-execution control fields specified in Section 28.2.1.1. They do not apply the checks to the current VMCS. Instead, VM-entry behavior depends on whether the executive-VMCS pointer field contains the VMXON pointer:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are not performed at all.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the checks are performed on the VM-execution control fields in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS). These checks are performed after checking the executive-VMCS pointer field itself (for proper alignment).

Other VM entries ensure that, if "activate VMX-preemption timer" VM-execution control is 0, the "save VMX-preemption timer value" VM-exit control is also 0. This check is not performed by VM entries that return from SMM.

33.15.4.3 Checks on VM-Entry Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-entry control fields specified in Section 28.2.1.3.

Specifically, if the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the VM-entry interruption-information field must not indicate injection of a pending MTF VM exit (see Section 28.6.2). Specifically, the following cannot all be true for that field:

- the valid bit (bit 31) is 1
- the interruption type (bits 10:8) is 7 (other event); and
- the vector (bits 7:0) is 0 (pending MTF VM exit).

33.15.4.4 Checks on the Guest State Area

Section 28.3.1 specifies checks performed on fields in the guest-state area of the VMCS. Some of these checks are conditioned on the settings of certain VM-execution controls (e.g., "virtual NMIs" or "unrestricted guest").

VM entries that return from SMM modify these checks based on whether the executive-VMCS pointer field contains the VMXON pointer:²

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are performed as all relevant VM-execution controls were 0. (As a result, some checks may not be performed at all.)
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), this check is performed based on the settings of the VM-execution controls in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS).

1. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

2. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

For VM entries that return from SMM, the activity-state field must not indicate the wait-for-SIPI state if the executive-VMCS pointer field contains the VMXON pointer (the VM entry is to VMX root operation).

33.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate linear mappings and combined mappings associated with all VPIDs. Combined mappings are invalidated for all EPTRTA values (EPTRTA is the value of bits 51:12 of EPTP; see Section 30.4). (Ordinary VM entries are required to perform such invalidation only for VPID 0000H and are not required to do even that if the “enable VPID” VM-execution control is 1; see Section 28.3.2.5.)

33.15.4.6 VMX-Preemption Timer

A VM entry that returns from SMM activates the VMX-preemption timer only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation) and the “activate VMX-preemption timer” VM-execution control is 1 in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field). In this case, VM entry starts the VMX-preemption timer with the value in the VMX-preemption timer-value field in the current VMCS.

33.15.4.7 Updating the Current-VMCS and SMM-Transfer VMCS Pointers

Successful VM entries (returning from SMM) load the SMM-transfer VMCS pointer with the current-VMCS pointer. Following this, they load the current-VMCS pointer from a field in the current VMCS:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the current-VMCS pointer is loaded from the VMCS-link pointer field.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the current-VMCS pointer is loaded with the value of the executive-VMCS pointer field.

If the VM entry successfully enters VMX non-root operation, the VM-execution controls in effect after the VM entry are those from the new current VMCS. This includes any structures external to the VMCS referenced by VM-execution control fields.

The updating of these VMCS pointers occurs before event injection. Event injection is determined, however, by the VM-entry control fields in the VMCS that was current when the VM entry commenced.

33.15.4.8 VM Exits Induced by VM Entry

Section 28.6.1.2 describes how the event-delivery process invoked by event injection may lead to a VM exit. Section 28.7.3 to Section 28.7.7 describe other situations that may cause a VM exit to occur immediately after a VM entry.

Whether these VM exits occur is determined by the VM-execution control fields in the current VMCS. For VM entries that return from SMM, they can occur only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation).

In this case, determination is based on the VM-execution control fields in the VMCS that is current after the VM entry. This is the VMCS referenced by the value of the executive-VMCS pointer field at the time of the VM entry (see Section 33.15.4.7). This VMCS also controls the delivery of such VM exits. Thus, VM exits induced by a VM entry returning from SMM are to the executive monitor and not to the STM.

33.15.4.9 SMI Blocking

VM entries that return from SMM determine the blocking of system-management interrupts (SMIs) as follows:

- If the “deactivate dual-monitor treatment” VM-entry control is 0, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the blocking of SMIs depends on whether the logical processor is in SMX operation:¹
 - If the logical processor is in SMX operation, SMIs are blocked after VM entry.

- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

VM entries that return from SMM and that do not deactivate the dual-monitor treatment may leave SMIs blocked. This feature exists to allow the STM to invoke functionality outside of SMM without unblocking SMIs.

33.15.4.10 Failures of VM Entries That Return from SMM

Section 28.8 describes the treatment of VM entries that fail during or after loading guest state. Such failures record information in the VM-exit information fields and load processor state as would be done on a VM exit. The VMCS used is the one that was current before the VM entry commenced. Control is thus transferred to the STM and the logical processor remains in SMM.

33.15.5 Enabling the Dual-Monitor Treatment

Code and data for the SMM-transfer monitor (STM) reside in a region of SMRAM called the **monitor segment** (MSEG). Code running in SMM determines the location of MSEG and establishes its content. This code is also responsible for enabling the dual-monitor treatment.

SMM code enables the dual-monitor treatment and specifies the location of MSEG by writing to the IA32_SMM_MONITOR_CTL MSR (index 9BH). The MSR has the following format:

- Bit 0 is the register's valid bit. The STM may be invoked using VMCALL only if this bit is 1. Because VMCALL is used to activate the dual-monitor treatment (see Section 33.15.6), the dual-monitor treatment cannot be activated if the bit is 0. This bit is cleared when the logical processor is reset.
- Bit 1 is reserved.
- Bit 2 determines whether executions of VMXOFF unblock SMIs under the default treatment of SMIs and SMM. Executions of VMXOFF unblock SMIs unless bit 2 is 1 (the value of bit 0 is irrelevant). See Section 33.14.4. Certain leaf functions of the GETSEC instruction clear this bit (see Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D).
- Bits 11:3 are reserved.
- Bits 31:12 contain a value that, when shifted left 12 bits, is the physical address of MSEG (the MSEG base address).
- Bits 63:32 are reserved.

The following items detail use of this MSR:

- The IA32_SMM_MONITOR_CTL MSR is supported only on processors that support the dual-monitor treatment.¹ On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32_SMM_MONITOR_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to the IA32_SMM_MONITOR_CTL MSR fails if made as part of a VM exit that does not end in SMM or part of a VM entry that does not begin in SMM.
- Reads from the IA32_SMM_MONITOR_CTL MSR using RDMSR are allowed any time RDMSR is allowed. The MSR may be read as part of any VM exit.
- The dual-monitor treatment can be activated only if the valid bit in the MSR is set to 1.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENDER]. A logical processor is outside SMX operation if GETSEC[SENDER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENDER]. See Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.

1. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

The 32 bytes located at the MSEG base address are called the **MSEG header**. The format of the MSEG header is given in Table 33-10 (each field is 32 bits).

Table 33-10. Format of MSEG Header

Byte Offset	Field
0	MSEG-header revision identifier
4	SMM-transfer monitor features
8	GDTR limit
12	GDTR base offset
16	CS selector
20	EIP offset
24	ESP offset
28	CR3 offset

To ensure proper behavior in VMX operation, software should maintain the MSEG header in writeback cacheable memory. Future implementations may allow or require a different memory type.¹ Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

SMM code should enable the dual-monitor treatment (by setting the valid bit in IA32_SMM_MONITOR_CTL MSR) only after establishing the content of the MSEG header as follows:

- Bytes 3:0 contain the **MSEG revision identifier**. Different processors may use different MSEG revision identifiers. These identifiers enable software to avoid using an MSEG header formatted for one processor on a processor that uses a different format. Software can discover the MSEG revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).
- Bytes 7:4 contain the **SMM-transfer monitor features** field. Bits 31:1 of this field are reserved and must be zero. Bit 0 of the field is the **IA-32e mode SMM feature bit**. It indicates whether the logical processor will be in IA-32e mode after the STM is activated (see Section 33.15.6).
- Bytes 31:8 contain fields that determine how processor state is loaded when the STM is activated (see Section 33.15.6.5). SMM code should establish these fields so that activating of the STM invokes the STM's initialization code.

33.15.6 Activating the Dual-Monitor Treatment

The dual-monitor treatment may be enabled by SMM code as described in Section 33.15.5. The dual-monitor treatment is activated only if it is enabled and only by the executive monitor. The executive monitor activates the dual-monitor treatment by executing VMCALL in VMX root operation.

When VMCALL activates the dual-monitor treatment, it causes an SMM VM exit. Differences between this SMM VM exit and other SMM VM exits are discussed in Sections 33.15.6.1 through 33.15.6.6. See also “VMCALL—Call to VM Monitor” in Chapter 32.

33.15.6.1 Initial Checks

An execution of VMCALL attempts to activate the dual-monitor treatment if (1) the processor supports the dual-monitor treatment;² (2) the logical processor is in VMX root operation; (3) the logical processor is outside SMM and

1. Alternatively, software may map the MSEG header with the UC memory type; this may be necessary, depending on how memory is organized. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

the valid bit is set in the IA32_SMM_MONITOR_CTL MSR; (4) the logical processor is not in virtual-8086 mode and not in compatibility mode; (5) CPL = 0; and (6) the dual-monitor treatment is not active.

Such an execution of VMCALL begins with some initial checks. These checks are performed before updating the current-VMCS pointer and the executive-VMCS pointer field (see Section 33.15.2.2).

The VMCS that manages SMM VM exit caused by this VMCALL is the current VMCS established by the executive monitor. The VMCALL performs the following checks on the current VMCS in the order indicated:

1. There must be a current VMCS pointer.
2. The launch state of the current VMCS must be clear.
3. The VM-exit controls in the current VMCS must be set properly:
 - Reserved bits in the primary VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper setting (see Appendix A.4.1).
 - If the “activate secondary controls” primary VM-exit control is 1, reserved bits in the secondary VM-exit controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.4.2).
 - If the “activate secondary controls” primary VM-exit control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary VM-exit controls. The logical processor operates as if all the secondary VM-exit controls were 0.

If any of these checks fail, subsequent checks are skipped and VMCALL fails. If all these checks succeed, the logical processor uses the IA32_SMM_MONITOR_CTL MSR to determine the base address of MSEG. The following checks are performed in the order indicated:

1. The logical processor reads the 32 bits at the base of MSEG and compares them to the processor’s MSEG revision identifier.
2. The logical processor reads the SMM-transfer monitor features field:
 - Bit 0 of the field is the IA-32e mode SMM feature bit, and it indicates whether the logical processor will be in IA-32e mode after the SMM-transfer monitor (STM) is activated.
 - If the VMCALL is executed on a processor that does not support Intel 64 architecture, the IA-32e mode SMM feature bit must be 0.
 - If the VMCALL is executed in 64-bit mode, the IA-32e mode SMM feature bit must be 1.
 - Bits 31:1 of this field are currently reserved and must be zero.

If any of these checks fail, subsequent checks are skipped and the VMCALL fails.

33.15.6.2 Updating the Current-VMCS and Executive-VMCS Pointers

Before performing the steps in Section 33.15.2.2, SMM VM exits that activate the dual-monitor treatment begin by loading the SMM-transfer VMCS pointer with the value of the current-VMCS pointer.

33.15.6.3 Saving Guest State

As noted in Section 33.15.2.4, SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area. While this is true also for SMM VM exits that activate the dual-monitor treatment, the VMCS used for those VM exits exists outside SMRAM.

The SMM-transfer monitor (STM) can also discover the current value of the SMBASE register by using the RDMSR instruction to read the IA32_SMBASE MSR (MSR address 9EH). The following items detail use of this MSR:

- The MSR is supported only if IA32_VMX_MISC[15] = 1 (see Appendix A.6).
- A write to the IA32_SMBASE MSR using WRMSR generates a general-protection fault (#GP(0)). An attempt to write to the IA32_SMBASE MSR fails if made as part of a VM exit or part of a VM entry.

-
2. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

- A read from the IA32_SMBASE MSR using RDMSR generates a general-protection fault (#GP(0)) if executed outside of SMM. An attempt to read from the IA32_SMBASE MSR fails if made as part of a VM exit that does not end in SMM.

33.15.6.4 Saving MSRs

The VM-exit MSR-store area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are saved into that area.

33.15.6.5 Loading Host State

The VMCS that is current during an SMM VM exit that activates the dual-monitor treatment was established by the executive monitor. It does not contain the VM-exit controls and host state required to initialize the STM. For this reason, such SMM VM exits do not load processor state as described in Section 29.5. Instead, state is set to fixed values or loaded based on the content of the MSEG header (see Table 33-10):

- CR0 is set to as follows:
 - PG, NE, ET, MP, and PE are all set to 1.
 - CD and NW are left unchanged.
 - All other bits are cleared to 0.
- CR3 is set as follows:
 - Bits 63:32 are cleared on processors that support IA-32e mode.
 - Bits 31:12 are set to bits 31:12 of the sum of the MSEG base address and the CR3-offset field in the MSEG header.
 - Bits 11:5 and bits 2:0 are cleared (the corresponding bits in the CR3-offset field in the MSEG header are ignored).
 - Bits 4:3 are set to bits 4:3 of the CR3-offset field in the MSEG header.
- CR4 is set as follows:
 - MCE, PGE, CET, PCIDE, and LA57 are cleared.
 - PAE is set to the value of the IA-32e mode SMM feature bit.
 - If the IA-32e mode SMM feature bit is clear, PSE is set to 1 if supported by the processor; if the bit is set, PSE is cleared.
 - All other bits are unchanged.
- DR7 is set to 400H.
- The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
- The registers CS, SS, DS, ES, FS, and GS are loaded as follows:
 - All registers are usable.
 - CS.selector is loaded from the corresponding field in the MSEG header (the high 16 bits are ignored), with bits 2:0 cleared to 0. If the result is 0000H, CS.selector is set to 0008H.
 - The selectors for SS, DS, ES, FS, and GS are set to CS.selector+0008H. If the result is 0000H (if the CS selector was FFF8H), these selectors are instead set to 0008H.
 - The base addresses of all registers are cleared to zero.
 - The segment limits for all registers are set to FFFFFFFFH.
 - The AR bytes for the registers are set as follows:
 - CS.Type is set to 11 (execute/read, accessed, non-conforming code segment).
 - For SS, DS, ES, FS, and GS, the Type is set to 3 (read/write, accessed, expand-up data segment).
 - The S bits for all registers are set to 1.
 - The DPL for each register is set to 0.

- The P bits for all registers are set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the value of the IA-32e mode SMM feature bit.
- CS.D is loaded with the inverse of the value of the IA-32e mode SMM feature bit.
- For each of SS, DS, ES, FS, and GS, the D/B bit is set to 1.
- The G bits for all registers are set to 1.
- LDTR is unusable. The LDTR selector is cleared to 0000H, and the register is otherwise undefined (although the base address is always canonical)
- GDTR.base is set to the sum of the MSEG base address and the GDTR base-offset field in the MSEG header (bits 63:32 are always cleared on processors that support IA-32e mode). GDTR.limit is set to the corresponding field in the MSEG header (the high 16 bits are ignored).
- IDTR.base is unchanged. IDTR.limit is cleared to 0000H.
- RIP is set to the sum of the MSEG base address and the value of the RIP-offset field in the MSEG header (bits 63:32 are always cleared on logical processors that support IA-32e mode).
- RSP is set to the sum of the MSEG base address and the value of the RSP-offset field in the MSEG header (bits 63:32 are always cleared on logical processor that supports IA-32e mode).
- RFLAGS is cleared, except bit 1, which is always set.
- The logical processor is left in the active state.
- Event blocking after the SMM VM exit is as follows:
 - There is no blocking by STI or by MOV SS.
 - There is blocking by non-maskable interrupts (NMIs) and by SMIs.
- There are no pending debug exceptions after the SMM VM exit.
- For processors that support IA-32e mode, the IA32_EFER MSR is modified so that LME and LMA both contain the value of the IA-32e mode SMM feature bit.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was 1 before and after the transition).

33.15.6.6 Loading MSRs

The VM-exit MSR-load area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are loaded from that area.

33.15.7 Deactivating the Dual-Monitor Treatment

The SMM-transfer monitor may deactivate the dual-monitor treatment and return the processor to default treatment of SMIs and SMM (see Section 33.14). It does this by executing a VM entry with the “deactivate dual-monitor treatment” VM-entry control set to 1.

As noted in Section 28.2.1.3 and Section 33.15.4.1, an attempt to deactivate the dual-monitor treatment fails in the following situations: (1) the processor is not in SMM; (2) the “entry to SMM” VM-entry control is 1; or (3) the executive-VMCS pointer does not contain the VMXON pointer (the VM entry is to VMX non-root operation).

As noted in Section 33.15.4.9, VM entries that deactivate the dual-monitor treatment ignore the SMI bit in the interruptibility-state field of the guest-state area. Instead, the blocking of SMIs following such a VM entry depends on whether the logical processor is in SMX operation:¹

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D.

- If the logical processor is in SMX operation, SMIs are blocked after VM entry. SMIs may later be unblocked by the VMXOFF instruction (see Section 33.14.4) or by certain leaf functions of the GETSEC instruction (see Chapter 7, “Safer Mode Extensions Reference,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2D).
- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

33.16 SMI AND PROCESSOR EXTENDED STATE MANAGEMENT

On processors that support processor extended states using XSAVE/XRSTOR (see Chapter 13, “Managing State Using the XSAVE Feature Set,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1), the processor does not save any XSAVE/XRSTOR related state on an SMI. It is the responsibility of the SMI handler code to properly preserve the state information (including CR4.OSXSAVE, XCR0, and possibly processor extended states using XSAVE/XRSTOR). Therefore, the SMI handler must follow the rules described in Chapter 13, “Managing State Using the XSAVE Feature Set,” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

33.17 MODEL-SPECIFIC SYSTEM MANAGEMENT ENHANCEMENT

This section describes enhancement of system management features that apply only to the 4th generation Intel Core processors. These features are model-specific. BIOS and SMM handler must use CPUID to enumerate Display-Family_DisplayModel signature when programming with these interfaces.

33.17.1 SMM Handler Code Access Control

The BIOS may choose to restrict the address ranges of code that SMM handler executes. When SMM handler code execution check is enabled, an attempt by the SMM handler to execute outside the ranges specified by SMRR (see Section 33.4.2.1) will cause the assertion of an unrecoverable machine check exception (MCE).

The interface to enable SMM handler code access check resides in a per-package scope model-specific register MSR_SMM_FEATURE_CONTROL at address 4E0H. An attempt to access MSR_SMM_FEATURE_CONTROL outside of SMM will cause a #GP. Writes to MSR_SMM_FEATURE_CONTROL is further protected by configuration interface of MSR_SMM_MCA_CAP at address 17DH.

Details of the interface of MSR_SMM_FEATURE_CONTROL and MSR_SMM_MCA_CAP are described in Table 2-29 in Chapter 2, “Model-Specific Registers (MSRs),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

33.17.2 SMI Delivery Delay Reporting

Entry into the system management mode occurs at instruction boundary. In situations where a logical processor is executing an instruction involving a long flow of internal operations, servicing an SMI by that logical processor will be delayed. Delayed servicing of SMI of each logical processor due to executing long flows of internal operation in a physical processor can be queried via a package-scope register MSR_SMM_DELAYED at address 4E2H.

The interface to enable reporting of SMI delivery delay due to long internal flows resides in a per-package scope model-specific register MSR_SMM_DELAYED. An attempt to access MSR_SMM_DELAYED outside of SMM will cause a #GP. Availability to MSR_SMM_DELAYED is protected by configuration interface of MSR_SMM_MCA_CAP at address 17DH.

Details of the interface of MSR_SMM_DELAYED and MSR_SMM_MCA_CAP are described in Table 2-29 in Chapter 2, “Model-Specific Registers (MSRs),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

33.17.3 Blocked SMI Reporting

A logical processor may have entered into a state and blocked from servicing other interrupts (including SMI). Logical processors in a physical processor that are blocked in serving SMI can be queried in a package-scope register MSR_SMM_BLOCKED at address 4E3H. An attempt to access MSR_SMM_BLOCKED outside of SMM will cause a #GP.

Details of the interface of MSR_SMM_BLOCKED is described in Table 2-29 in Chapter 2, “Model-Specific Registers (MSRs),” of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4.

27. Updates to Chapter 34, Volume 3C

Change bars and violet text show changes to Chapter 34 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Inserted new Chapter 34, "Secure Arbitration Mode (SEAM)" and re-numbered subsequent chapters.

CHAPTER 34

SECURE ARBITRATION MODE (SEAM)

This chapter specifies secure arbitration mode (SEAM), an extension of the virtual machines extensions architecture (VMX) that allows running secure virtual machines.

The SEAM architecture is used as a foundation for Trusted Domains Extensions (TDX) architecture. For details about TDX, see *Intel® Trust Domain Extensions (Intel® TDX) — An overview of the Intel TDX technology*.

34.1 MODES OF OPERATION

The SEAM architecture defines a protected mode of operation called SEAM, which is an extension of VMX operation. Like VMX operation, SEAM includes root and non-root operations:

- **SEAM root operation** is the SEAM extension of VMX root operation. The software running in SEAM root operation is called a **SEAM module**. There are two sub-modes to SEAM root operation:
 - **Loader mode**: At most one logical processor can execute in this sub-mode at any time.
 - **Module mode**: Multiple logical processors can execute in this sub-mode in parallel.
- **SEAM non-root operation** is the SEAM extension of VMX non-root operation. The software running in this SEAM non-root operation is called **SEAM guest**. Multiple logical processors can execute in this mode concurrently.

Certain platform operations may be required to enable SEAM; until they are performed, SEAM may be globally disabled. Once SEAM is enabled, it may or may not be ready for execution in loader mode or in module mode (again, possibly based on certain platform operations). Details regarding this enabling are provided in the *Intel® Trust Domain CPU Architectural Extensions*.

The following diagram shows the VMX and SEAM software components, the modes in which they operate, and the transitions between these execution modes.

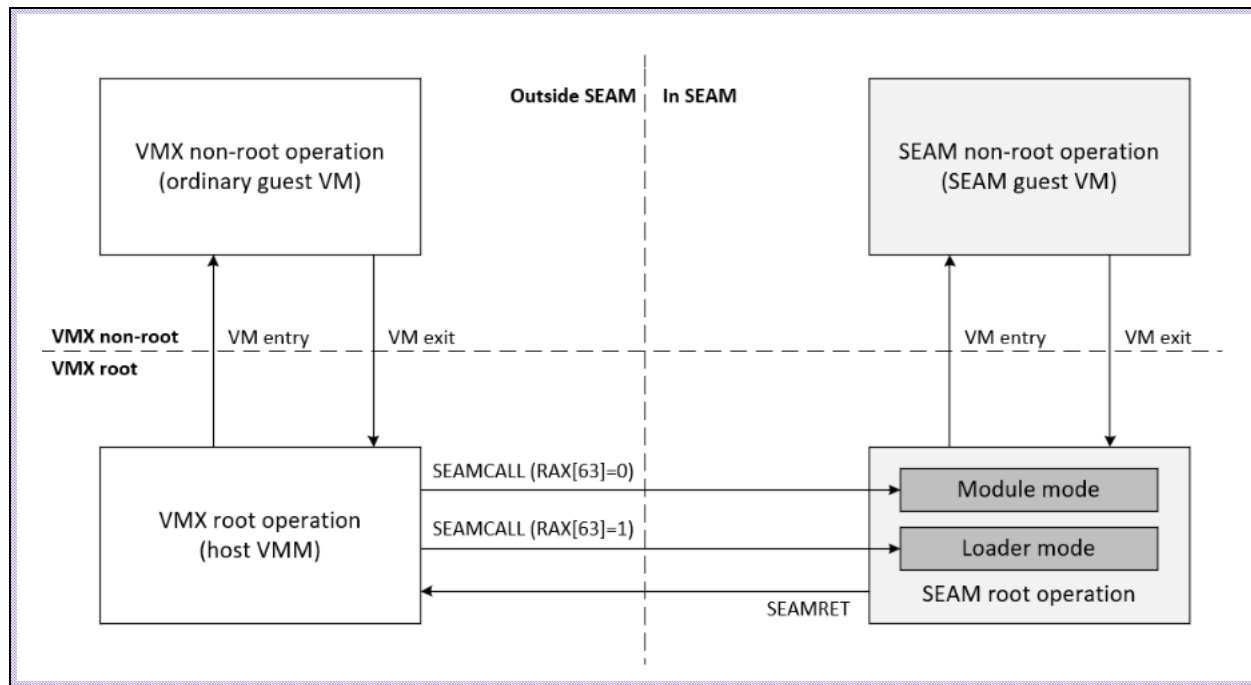


Figure 34-1. SEAM Modes of Operation

34.1.1 SEAM Calls

Processor transitions from VMX root operation to SEAM root operation are called **SEAM calls**. SEAM calls are initiated by executing the SEAMCALL instruction. A SEAM call operates in a manner similar to that of SMM VM exits (see Section 33.15.2) in response to execution of the VMCALL instruction in VMX root operation. A SEAM call uses a **SEAM-transfer VMCS** that resides within the SEAM range of the physical-address space (see Section 34.4.1).

SEAM calls use RAX[63] to determine the SEAM root operation sub-mode to be entered. When RAX[63] is 0, the transition is into module-mode SEAM root operation; otherwise, it is into loader-mode SEAM root operation. An attempt to enter loader-mode SEAM root operation fails if another logical processor is in SEAM having entered in loader mode.

34.1.2 SEAM Returns

Processor transitions from SEAM root operation back to VMX root operation are called **SEAM returns**. SEAM returns are initiated by executing the SEAMRET instruction. A SEAM return operates in a manner similar to that of VM entries that return from SMM (see Section 33.15.4). Like SEAM calls, SEAM returns use a SEAM-transfer VMCS.

A SEAM return from loader-mode SEAM root operation ensures that all data for any VMCS that the logical processor used in SEAM (including the SEAM-transfer VMCS) are in memory; in addition, there is no current VMCS after such a SEAM return.

34.1.3 SEAM Host-Guest Transitions

Processor transitions between SEAM root operation and SEAM non-root operation are similar to processor transition between VMX root operation and VMX non-root operation. A VMCS used to transition between SEAM root and non-root operation, and to control SEAM non-root operation, is called **SEAM-guest VMCS**.

A SEAM-guest VMCS supports SEAM-specific fields and controls. In general, those fields and controls are ignored outside SEAM.

The SEAM-specific VMCS fields are the following:

- **SEAM-guest KeyID**. This is a VM-execution control field that contains the key identifier (KeyID) assigned for the key with which the SEAM guest's private pages are encrypted in memory.
- **SEAM shared EPT pointer**. This is a VM-execution control field that contains the physical address of the base of shared EPT PML4 table or PML5 table (depending on how EPT is configured).

The SEAM-specific VMX controls are the following:

- **SEAM guest-physical address width**. This is a VM-execution control that determines, for SEAM non-root operation, the width of guest-physical addresses translated by EPT and also the position of the SHARED bit in guest-physical addresses. See Section 34.4.
- **Allow SEAM-guest telemetry**. This is a VM-entry control that allows collection of core telemetry in SEAM non-root operation.

Any processor that supports SEAM also supports the SEAM-specific VMCS fields and controls identified above. Support for the SEAM-specific VMX controls above is enumerated normally by the IA32_VMX_PROCBASED_CTLSS3 MSR (for "SEAM guest-physical address width") and by IA32_VMX_ENTRY_CTLSS MSR (for "allow SEAM-guest telemetry").

SEAM guests must run with EPT enabled. VM entry in SEAM root operation will fail if the "enable EPT" VM-execution control is not set in the SEAM-guest VMCS.

34.2 SEAM AND TME-MK

SEAM interoperates with multi-key total memory encryption (TME-MK). This section explains the specifics of TME-MK related to that interoperation.

The current configuration of TME-MK can be inferred from the contents of the IA32_TME_ACTIVATE MSR (MSR address 982H). The following fields are relevant:

- If `IA32_TME_ACTIVATE[1:0] = 11B`, total-memory encryption (TME) is active.
- `IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32]` specifies the number of bits in a physical address that are interpreted as a key identifier (KeyID). If TME is active and this value is greater than 0, TME-MK is active.
- `IA32_TME_ACTIVATE.TDX_RESERVED_KEYID_BITS[39:36]` specifies the number of KeyID bits that can be used only in SEAM-private KeyIDs (see Section 34.3.2). This value never exceeds `IA32_TME_ACTIVATE.MK_TME_KEYID_BITS`.

Let m be the value enumerated by `CPUID.80000008H:EAX[7:0]`; $k = \text{IA32\_TME\_ACTIVATE.MK\_TME\_KEYID\_BITS}$; and $p = \text{IA32\_TME\_ACTIVATE.TDX\_RESERVED\_KEYID\_BITS}$. Then in any physical address, bits $m-1:m-k$ form a key identifier that is used to select an encryption key (only bits $(m-k)-1:0$ are used to address memory). The `PCONFIG` instruction can be used to specify the encryption key associated with each key identifier. A k -bit key identifier is a **SEAM-private KeyID** if it sets any of bits $k-1:k-p$; thus, a physical address contains a SEAM-private KeyID if it sets any of bits $m-1:m-p$.

The `IA32_MKTME_KEYID_PARTITIONING` MSR (MSR address 87H) enumerates the number of key identifiers available for use:

- `IA32_MKTME_KEYID_PARTITIONING.NUM_MKTME_KIDS[31:0]` enumerates the number of TME-MK key identifiers available for general use. These are numbered 1 to `NUM_MKTME_KIDS` (0 is not considered a TME-MK key identifier).
- `IA32_MKTME_KEYID_PARTITIONING.NUM_TDX_PRIV_KIDS[63:32]` enumerates the number of SEAM-private key identifiers. These are numbered `NUM_MKTME_KIDS + 1` to `NUM_MKTME_KIDS + NUM_TDX_PRIV_KIDS`.

34.3 OPERATION IN SEAM

This section provides details of both SEAM root operation and SEAM non-root operation.

34.3.1 SEAM Root Operation

Processor behavior in SEAM root operation is similar to processor behavior in VMX root operation, with the following differences:

- Software can execute `SEAMRET` and `SEAMOPS`. (Execution of these instructions outside SEAM root operation results in a `#UD` exception.) As noted in Section 34.1.2, execution of `SEAMRET` causes a SEAM return to VMX root operation. `SEAMOPS` is described in *Intel® Trust Domain CPU Architectural Extensions*.
- Software can access a special set of SEAM-only MSRs. An attempt to access a SEAM-only MSR outside SEAM results in a `#GP(0)` exception. (Some may be accessible only in SEAM root operation.) The full set of SEAM-only MSRs is specified in *Intel® Trust Domain CPU Architectural Extensions*.
- An attempt to execute the `VMXOFF` instruction or the `VMCALL` instruction in SEAM root operation results in a `#GP(0)` exception. Software in SEAM cannot leave VMX operation, nor can it invoke an SMM-transfer monitor using an SMM VM exit.
- Entry to SEAM root operation (by either a SEAM call or a VM exit from SEAM non-root operation) blocks NMIs. Execution of `IRET` in SEAM root operation unblocks NMIs normally.
- System-management interrupts (SMIs), uncore machine checks and machine-check SMIs (MSMIs) are masked in SEAM root operation. If pending, they are delivered after leaving SEAM root operation.
- Triple faults and machine checks caused by poisoned cache-line consumption in SEAM root operation cause unbreakable shutdown.

34.3.2 SEAM Non-Root Operation

Processor behavior in SEAM non-root operation is largely the same as that in VMX non-root operation.

System-management interrupts (SMIs) and machine-check SMIs (MSMIs) that occur in SEAM non-root operation cause a VM exit instead of being delivered normally. The SMI or MSMI remains pending and will be delivered following the next SEAM return.

The remainder of this section explains the changes in SEAM non-root operation with regard to the use of guest-physical addresses.

As noted in Section 34.1.3, VM entry to SEAM non-root operation requires EPT to be enabled. Thus, all addresses used by guest software as physical addresses (e.g., addresses in the guest's paging structures) are treated as guest-physical addresses and are thus translated by EPT. Translation by EPT in SEAM non-root operation differs somewhat from that in ordinary VMX non-root operation.

In VMX non-root operation, the width of guest-physical addresses is either 48 bits (if 4-level EPT is in use) or 52 bits (for 5-level EPT). In SEAM non-root operation, the guest-physical address width is 48 bits if the "SEAM guest-physical address width" VM-execution control is 0 (even if 5-level EPT is in use).

If the guest-physical address width is 48, an access to a guest-physical address that sets bits in the range 51:48 causes an EPT violation. (This is not the case with 5-level EPT outside SEAM. In SEAM, this occurs with 5-level paging as long as the "SEAM guest-physical address width" VM-execution control is 0.)

The uppermost bit in a guest-physical address (bit 47 or bit 51 as appropriate) is defined to be the address's "SHARED" bit. Guest-physical addresses that clear this bit are **SEAM-guest private addresses**; those that set this bit are **SEAM-guest shared addresses**.

Certain situations in SEAM non-root operation require SEAM-guest private addresses:

- Instruction fetches are limited to SEAM-guest private addresses. An attempt to fetch an instruction from a SEAM-guest shared address results in a general-protection exception (#GP(0)).
- CR3 must contain a SEAM-guest private address. An attempt to load CR3 with a SEAM-guest shared address results in #GP(0).
- With PAE paging, each of the PDPTs registers must contain a SEAM-guest private address. An attempt to load any PDPT register with a SEAM-guest shared address results in #GP(0).
- The guest paging structures must be located at SEAM-guest private addresses. If linear-address translation encounters a paging-structure entry that references a paging structure with a SEAM-guest shared address, a page fault (#PF) occurs. The error code will set bit 3, indicating that a reserved bit was set.

Translation of guest-physical addresses differs for private and shared addresses:

- Translation of a SEAM-guest private address uses EPT paging structures located using the EPTP field in the VMCS.
 - Any physical address used to access an EPT paging structure is modified as follows. Let k be the value of `IA32_TME_ACTIVATE.MK_TME_KEYID_BITS[35:32]` and M be the value enumerated by `CPUID.80000008H:EAX[7:0]`. Before being used to access memory, bits $M-1:M-k$ of the physical address are replaced by bits $k-1:0$ of the SEAM-guest KeyID field in the VMCS.
 - The final physical address (to which the original SEAM-guest private address is being translated) is also modified in the manner described in the previous paragraph.
- Translation of a SEAM-guest shared address uses EPT paging structures located using the SEAM shared EPT pointer in the VMCS.
 - Bits 7:0 of the EPTP field (not the SEAM shared EPT pointer) control the translation of the guest-physical address.
 - Any physical address used to access an EPT paging structure should not use a SEAM-private KeyID (see Section 34.2).

If translation of a SEAM-guest shared address would access an EPT paging structure with a SEAM-private KeyID, an EPT misconfiguration occurs.

 - The final physical address (to which the original SEAM-guest shared address is being translated) also must not contain a SEAM-private KeyID; otherwise, an EPT misconfiguration occurs.

34.4 MEMORY PROTECTION

This section specifies access controls and cryptographic measures applied to protect software running in SEAM root operation (SEAM modules) and SEAM non-root operation (SEAM guests).

34.4.1 SEAM Range

The SEAM range is a physical memory range sequestered for SEAM modules. The range is allocated using range-register MSRs `IA32_SEAMRR_BASE` (index 1400H) and `IA32_SEAMRR_MASK` (index 1401H). The existence of these MSRs is enumerated by bit 15 of `IA32_MTRRCAP` (MSR address FEH). Processors that support SEAM architecture return `IA32_MTRRCAP` bit 15 as 1. For more information about these MSRs, see Chapter 2 in Volume 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual*.

The SEAM range is divided into a **module sub-range** and a **loader sub-range**.

Bit 11 of `IA32_SEAMRR_MASK` MSR indicates whether the SEAM range is valid and protected. When this bit is 1, the processor provides the SEAM range with cryptographic and access-control protections as follows:

- Outside SEAM root operation, accesses to the physical addresses in the SEAM range are aborted, meaning that writes are ignored and reads return values with all bits set to 1.
- The following items apply in SEAM root operation:
 - Accesses to the module sub-range of the SEAM range are allowed.
 - Accesses to the loader sub-range of the SEAM range are allowed only in loader-mode SEAM root operation; otherwise they are aborted (see above).
 - Instruction fetches from physical addresses outside the SEAM range are prevented by the processor (the response to such attempts is implementation-specific).
- The contents of the SEAM range are encrypted by the TME-MK engine. The encryption key is machine-specific. It's typically an "ephemeral" key that is generated each time the platform is restarted.

Accesses to SEAM range when `CR0.CD = 0` use the writeback (WB) memory type regardless of the MTRR and PAT registers. If `CR0.CD = 1`, accesses use the uncacheable (UC) memory type.

34.4.2 Memory Protections

Outside SEAM, registers holding physical addresses cannot be loaded with physical addresses with SEAM-private KeyIDs (see Section 34.3.2):

- Attempts to load control registers and MSRs with a physical address with a SEAM-private KeyID results in a general-protection exception. This does not apply to loads of the `IA32_RTIT_OUTPUT_BASE` MSR (using `WRMSR`, `WRMSRLIST`, `WRMSRNS`, or `XRSTORS`), as that MSR is checked at another time (see below).
- Linear-address translation without EPT that encounters a paging-structure entry with a physical address with a SEAM-private KeyID results in a page fault.
- Guest-physical-address translation with EPT that encounters an EPT paging-structure entry with a physical address with a SEAM-private KeyID results in an EPT misconfiguration.
- VM entry fails if it would use or load a physical address (from a VMCS field other than the SEAM shared EPT pointer) that contains a SEAM-private KeyID.
- The value of the physical address loaded into `IA32_RTIT_OUTPUT_BASE` MSR is checked when processor trace is enabled (when `IA32_RTIT_CTL.TraceEn` is set to 1), as is the physical address of a new output buffer when it is loaded from the table of physical addresses (ToPA).

If, despite the limitations identified above, there is an access using a physical address with a SEAM-private KeyID (perhaps because an MSR had been loaded before TME was activated), the access is aborted, meaning that writes are ignored and reads return values with all bits set to 1.

A cacheline that was written with a physical address with a SEAM-private KeyID cannot be read using a physical address with a different KeyID. The memory-protection mode supported by a particular processor is machine-dependent. For information about memory protection modes, see *Intel® Architecture Memory Protections for Confidential Compute Usages*.

34.5 SEAM INSTRUCTION REFERENCE

The SEAM architecture includes the following instructions:

- **SEAMCALL.** Execution causes a SEAM call (see Section 34.1.1) to enter SEAM root operation. It can be used by virtual-machine monitors (operating in VMX root operation) to call SEAM module services.
- **TDCALL.** Execution causes a VM exit from SEAM non-root operation to SEAM root operation. It can be used by SEAM guests to request SEAM-module services.
- **SEAMOPS.** Execution invokes SEAM-specific operation. It is used by SEAM modules to execute such operations.
- **SEAMRET.** Execution causes a SEAM return (Section 34.1.2) that leaves SEAM root operation. It is used by SEAM modules to return to the calling virtual-machine monitor.

SEAMCALL and TDCALL are described later in this section. SEAMOPS and SEAMRET can be executed only in SEAM root operation. Details are given in *Intel® Trust Domain CPU Architectural Extensions*, while the following items provide high-level information:

- **SEAMOPS.** The SEAMOPS instruction allows software running in SEAM root operation to invoke SEAM-specific operations. Some of these operations can be invoked only in loader mode.
- **SEAMRET.**
 - SEAMRET is a privileged instruction that allows software running in SEAM root operation to return to VMX root operation.
 - The instruction causes a SEAM return (see Section 34.1.2) using the SEAM-transfer VMCS. This transition is similar to VM entries that return from SMM (see Section 33.15.4).

An attempt to execute a SEAM instruction on a processor that does not support SEAM results in an invalid-opcode exception (#UD).

SEAMCALL—Enter SEAM Root Operation

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CF	SEAMCALL	Z0	Invalid	Valid	Transitions to SEAM root operation.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

SEAMCALL is a privileged instruction that passes control to software running in SEAM root operation. The instruction can be executed only in 64-bit mode in VMX root operation outside SEAM and outside SMM.

Execution when the SEAM range is not valid (IA32_SEAMRR_MASK MSR[11] = 0) or immediately following execution of MOV to SS or POP SS instruction results in a general-protection exception (#GP).

The instruction uses RAX[63] to select a sub-mode of SEAM root operation: if this value is 1, the instruction transitions into loader-mode SEAM root operation; otherwise, it transitions into module-mode SEAM root operation. The instruction fails with VMFailInvalid indication (see Section 34.2) if SEAM is globally disabled; or if a loader-mode SEAM root operation is called when another logical processor is in SEAM having entered in loader mode.

If there is no failure, the SEAMCALL uses the SEAM-transfer VMCS to effect a SEAM call (see Section 34.1.1). The details of such transitions are specified in the *Intel® Trust Domain CPU Architectural Extensions*.

Operation

```

IF not in VMX operation or in SMM or in SEAM or IA32_EFER.LMA = 0 or CS.L = 0
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF IA32_SEAMRR_MASK.VALID = 0 or events blocked by MOV SS
    THEN #GP(0);
ELSIF SEAM is globally disabled
    THEN VMFailInvalid;
ELSIF RAX[63] = 0
    THEN
        IF module-mode SEAM root operation is not ready for execution
            THEN VMFailInvalid;
        ELSE
            Enter module-mode SEAM root operation;
        FI;
    ELSE
        IF loader-mode SEAM root operation is not ready for execution or
           another logical processor is executing in SEAM having entered in loader mode
            THEN VMFailInvalid;
        ELSE
            Enter loader-mode SEAM root operation;
        FI;
    FI;

```

Flags Affected

See the Operation section above and Section 34.2.

Protected Mode Exceptions

#UD SEAMCALL is not recognized in protected mode.

Real-Address Mode Exceptions

#UD SEAMCALL is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD SEAMCALL is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD SEAMCALL is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0) If the current privilege level is not 0.
If the SEAM range is not valid.
If events blocked by MOV SS.
#UD If executed outside VMX operation.
If executed in system-management mode (SMM).
If executed in SEAM.

TDCALL—Call SEAM Module

Op/En	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
66 0F 01 CC	TDCALL	Z0	Invalid	Valid	Call SEAM monitor from SEAM guest.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
Z0	N/A	N/A	N/A	N/A

Description

TDCALL is a privileged instruction that allows software running in SEAM non-root operation to call software running in SEAM root operation. The instruction can be executed only in SEAM non-root operation or VMX non-root operation.

In SEAM non-root operation or VMX non-root operation, execution causes a VM exit with exit reason 77 (TDCALL).

Operation

```
IF not in SEAM non-root operation and not in VMX non-root operation
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSE VM exit;
FI;
```

Flags Affected

See the Operation section above and Section 34.2.

Protected Mode Exceptions

#UD If executed outside VMX non-root operation and outside SEAM non-root operation.
 #GP(0) If the current privilege level is not 0.

Real-Address Mode Exceptions

#UD If executed outside VMX non-root operation and outside SEAM non-root operation.

Virtual-8086 Mode Exceptions

#GP(0) TDCALL is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

Same as protected mode exceptions.

64-Bit Mode Exceptions

Same as protected mode exceptions.

28. Updates to Chapter 35, Volume 3C

Change bars and violet text show changes to Chapter 35 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C: System Programming Guide, Part 3*.

Changes to this chapter:

- Chapter 35, "Intel® Processor Trace" was previously numbered as 34. With the insertion of the new SEAM chapter (chapter 34), this was subsequently re-numbered.
- In Section 35.2.7.2, "Table of Physical Addresses (ToPA)", added footnote to ToPA Table Entry Format for MAXPHYADDR and updated the ToPA entry base address physical address bits for alignment violations described in ToPA Errors.
- Added PRMRR and SEAMRR protected regions to Table 35-5 in Section 35.2.7.4, "Restricted Memory Access".
- In Section 35.2.8.7, "IA32_RTIT_OUTPUT_BASE MSR" added description for processor's physical-address width, MAXPHYADDR. In addition, Table 35-8 was updated to include bit field M-1:MAXPHYADDR.

CHAPTER 35

INTEL® PROCESSOR TRACE

35.1 OVERVIEW

Intel® Processor Trace (**Intel PT**) is an extension of Intel® Architecture that captures information about software execution using dedicated hardware facilities that cause only minimal performance perturbation to the software being traced. This information is collected in **data packets**. The initial implementations of Intel PT offer **control flow tracing**, which generates a variety of packets to be processed by a software decoder. The packets include timing, program flow information (e.g., branch targets, branch taken/not taken indications) and program-induced mode related information (e.g., Intel TSX state transitions, CR3 changes). These packets may be buffered internally before being sent to the memory subsystem or other output mechanism available in the platform. Debug software can process the trace data and reconstruct the program flow.

Intel Processor Trace was first introduced in Intel® processors based on Broadwell microarchitecture and Intel Atom® processors based on Goldmont microarchitecture. Later generations include additional trace sources, including software trace instrumentation using PTWRITE, and Power Event tracing.

35.1.1 Features and Capabilities

Intel PT's control flow trace generates a variety of packets that, when combined with the binaries of a program by a post-processing tool, can be used to produce an exact execution trace. The packets record flow information such as instruction pointers (IP), indirect branch targets, and directions of conditional branches within contiguous code regions (basic blocks).

Intel PT can also be configured to log software-generated packets using PTWRITE, and packets describing processor power management events. Further, Precise Event-Based Sampling (PEBS) can be configured to log PEBS records in the Intel PT trace; see Section 21.5.5.2.

In addition, the packets record other contextual, timing, and bookkeeping information that enables both functional and performance debugging of applications. Intel PT has several control and filtering capabilities available to customize the tracing information collected and to append other processor state and timing information to enable debugging. For example, there are modes that allow packets to be filtered based on the current privilege level (CPL) or the value of CR3.

Configuration of the packet generation and filtering capabilities are programmed via a set of MSRs. The MSRs generally follow the naming convention of IA32_RTIT_*. The capability provided by these configuration MSRs are enumerated by CPUID, see Section 35.3. Details of the MSRs for configuring Intel PT are described in Section 35.2.8.

35.1.1.1 Packet Summary

After a tracing tool has enabled and configured the appropriate MSRs, the processor will collect and generate trace information in the following categories of packets (for more details on the packets, see Section 35.4):

- Packets about basic information on program execution; these include:
 - Packet Stream Boundary (PSB) packets: PSB packets act as 'heartbeats' that are generated at regular intervals (e.g., every 4K trace packet bytes). These packets allow the packet decoder to find the packet boundaries within the output data stream; a PSB packet should be the first packet that a decoder looks for when beginning to decode a trace.
 - Paging Information Packet (PIP): PIPs record modifications made to the CR3 register. This information, along with information from the operating system on the CR3 value of each process, allows the debugger to attribute linear addresses to their correct application source.
 - Time-Stamp Counter (TSC) packets: TSC packets aid in tracking wall-clock time, and contain some portion of the software-visible time-stamp counter.
 - Core Bus Ratio (CBR) packets: CBR packets contain the core:bus clock ratio.

- Mini Time Counter (MTC) packets: MTC packets provide periodic indication of the passing of wall-clock time.
- Cycle Count (CYC) packets: CYC packets provide indication of the number of processor core clock cycles that pass between packets.
- Overflow (OVF) packets: OVF packets are sent when the processor experiences an internal buffer overflow, resulting in packets being dropped. This packet notifies the decoder of the loss and can help the decoder to respond to this situation.
- Packets about control flow information:
 - Taken Not-Taken (TNT) packets: TNT packets track the “direction” of direct conditional branches (taken or not taken).
 - Target IP (TIP) packets: TIP packets record the target IP of indirect branches, exceptions, interrupts, and other branches or events. These packets can contain the IP, although that IP value may be compressed by eliminating upper bytes that match the last IP. There are various types of TIP packets; they are covered in more detail in Section 35.4.2.2.
 - Flow Update Packets (FUP): FUPs provide the source IP addresses for asynchronous events (interrupt and exceptions), as well as other cases where the source address cannot be determined from the binary.
 - MODE packets: These packets provide the decoder with important processor execution information so that it can properly interpret the dis-assembled binary and trace log. MODE packets have a variety of formats that indicate details such as the execution mode (16-bit, 32-bit, or 64-bit).
- Packets inserted by software:
 - PTWRITE (PTW) packets: includes the value of the operand passed to the PTWRITE instruction (see “PTWRITE—Write Data to a Processor Trace Packet” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B).
- Packets about processor power management events:
 - MWAIT packets: Indicate successful completion of an MWAIT operation to a C-state deeper than C0.0.
 - Power State Entry (PWRE) packets: Indicate entry to a C-state deeper than C0.0.
 - Power State Exit (PWRX) packets: Indicate exit from a C-state deeper than C0.0, returning to C0.
 - Execution Stopped (EXSTOP) packets: Indicate that software execution has stopped, due to events such as P-state change, C-state change, or thermal throttling.
- Packets containing groups of processor state values:
 - Block Begin Packets (BBP): Indicate the type of state held in the following group.
 - Block Item Packets (BIP): Indicate the state values held in the group.
 - Block End Packets (BEP): Indicate the end of the current group.

35.2 INTEL® PROCESSOR TRACE OPERATIONAL MODEL

This section describes the overall Intel Processor Trace mechanism and the essential concepts relevant to how it operates.

35.2.1 Change of Flow Instruction (COFI) Tracing

A basic program block is a section of code where no jumps or branches occur. The instruction pointers (IPs) in this block of code need not be traced, as the processor will execute them from start to end without redirecting code flow. Instructions such as branches, and events such as exceptions or interrupts, can change the program flow. These instructions and events that change program flow are called Change of Flow Instructions (COFI). There are three categories of COFI:

- Direct transfer COFI.
- Indirect transfer COFI.
- Far transfer COFI.

The following subsections describe the COFI events that result in trace packet generation. Table 35-1 lists branch instruction by COFI types. For detailed description of specific instructions, see the Intel® 64 and IA-32 Architectures Software Developer's Manual.

Table 35-1. COFI Type for Branch Instructions

COFI Type	Instructions
Conditional Branch	JA, JAE, JB, JBE, JC, JCXZ, JECXZ, JRCXZ, JE, JG, JGE, JL, JLE, JNA, JNAE, JNB, JNBE, JNC, JNE, JNG, JNGE, JNL, JNLE, JNO, JNP, JNS, JNZ, JO, JP, JPE, JPO, JS, JZ, LOOP, LOOPE, LOOPNE, LOOPNZ, LOOPZ
Unconditional Direct Branch	JMP (E9 xx, EB xx), CALL (E8 xx)
Indirect Branch	JMP (FF /4), CALL (FF /2), RET (C3, C2 xx)
Far Transfers	INT1, INT3, INT <i>n</i> , INTO, IRET, IRETD, IRETQ, JMP (EA xx, FF /5), CALL (9A xx, FF /3), RET (CB, CA xx), SYSCALL, SYSRET, SYSENTER, SYSEXIT, VMLAUNCH, VMRESUME

35.2.1.1 Direct Transfer COFI

Direct Transfer COFI are relative branches. This means that their target is an IP whose offset from the current IP is embedded in the instruction bytes. It is not necessary to indicate target of these instructions in the trace output since it can be obtained through the source disassembly. Conditional branches need to indicate only whether the branch is taken or not. Unconditional branches do not need any recording in the trace output. There are two sub-categories:

- **Conditional Branch (Jcc, J*CXZ) and LOOP**

To track this type of instruction, the processor encodes a single bit (taken or not taken — TNT) to indicate the program flow after the instruction.

Jcc, J*CXZ, and LOOP can be traced with TNT bits. To improve the trace packet output efficiency, the processor will compact several TNT bits into a single packet.

- **Unconditional Direct Jumps**

There is no trace output required for direct unconditional jumps (like JMP near relative or CALL near relative) since they can be directly inferred from the application assembly. Direct unconditional jumps do not generate a TNT bit or a Target IP packet, though TIP.PGD and TIP.PGE packets can be generated by unconditional direct jumps that toggle Intel PT enables (see Section 35.2.6).

35.2.1.2 Indirect Transfer COFI

Indirect transfer instructions involve updating the IP from a register or memory location. Since the register or memory contents can vary at any time during execution, there is no way to know the target of the indirect transfer until the register or memory contents are read. As a result, the disassembled code is not sufficient to determine the target of this type of COFI. Therefore, tracing hardware must send out the destination IP in the trace packet for debug software to determine the target address of the COFI. Note that this IP may be a linear or effective address (see Section 35.3.1.1).

An indirect transfer instruction generates a Target IP Packet (TIP) that contains the target address of the branch. There are two sub-categories:

- **Near JMP Indirect and Near Call Indirect**

As previously mentioned, the target of an indirect COFI resides in the contents of either a register or memory location. Therefore, the processor must generate a packet that includes this target address to allow the decoder to determine the program flow.

- **Near RET**

When a CALL instruction executes, it pushes onto the stack the address of the next instruction following the CALL. Upon completion of the call procedure, the RET instruction is often used to pop the return address off of the call stack and redirect code flow back to the instruction following the CALL.

A RET instruction simply transfers program flow to the address it popped off the stack. Because a called procedure may change the return address on the stack before executing the RET instruction, debug software

can be misled if it assumes that code flow will return to the instruction following the last CALL. Therefore, even for near RET, a Target IP Packet may be sent.

— RET Compression

A special case is applied if the target of the RET is consistent with what would be expected from tracking the CALL stack. If it is assured that the decoder has seen the corresponding CALL (with “corresponding” defined as the CALL with matching stack depth), and the RET target is the instruction after that CALL, the RET target may be “compressed”. In this case, only a single TNT bit of “taken” is generated instead of a Target IP Packet. To ensure that the decoder will not be confused in cases of RET compression, only RETs that correspond to CALLs which have been seen since the last PSB packet may be compressed in a given logical processor. For details, see “Indirect Transfer Compression for Returns (RET)” in Section 35.4.2.2.

35.2.1.3 Far Transfer COFI

All operations that change the instruction pointer and are not near jumps are “far transfers”. This includes exceptions, interrupts, traps, TSX aborts, and instructions that do far transfers.

All far transfers will produce a Target IP (TIP) packet, which provides the destination IP address. For those far transfers that cannot be inferred from the binary source (e.g., asynchronous events such as exceptions and interrupts), the TIP will be preceded by a Flow Update packet (FUP), which provides the source IP address at which the event was taken. Table 35-23 indicates exactly which IP will be included in the FUP generated by a far transfer.

35.2.2 Software Trace Instrumentation with PTWRITE

PTWRITE provides a mechanism by which software can instrument the Intel PT trace. PTWRITE is a ring3-accessible instruction that can be passed to a register or memory variable, see “PTWRITE—Write Data to a Processor Trace Packet” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B, for details. The contents of that variable will be used as the payload for the PTW packet (see Table 35-40 “PTW Packet Definition”), inserted at the time of PTWRITE retirement, assuming PTWRITE is enabled and all other filtering conditions are met. Decode and analysis software will then be able to determine the meaning of the PTWRITE packet based on the IP of the associated PTWRITE instruction.

PTWRITE is enabled via IA32_RTIT_CTL.PTWEn[12] (see Table 35-6). Optionally, the user can use IA32_RTIT_CTL.FUPonPTW[5] to enable PTW packets to be followed by FUP packets containing the IP of the associated PTWRITE instruction. Support for PTWRITE is introduced in Intel Atom processors based on the Goldmont Plus microarchitecture.

35.2.3 Power Event Tracing

Power Event Trace is a capability that exposes core- and thread-level sleep state and power down transition information. When this capability is enabled, the trace will expose information about:

- Scenarios where software execution stops.
 - Due to sleep state entry, frequency change, or other powerdown.
 - Includes the IP, when in the tracing context.
- The requested and resolved hardware thread C-state.
 - Including indication of hardware autonomous C-state entry.
- The last and deepest core C-state achieved during a sleep session.
- The reason for C-state wake.

This information is in addition to the bus ratio (CBR) information provided by default after any powerdown, and the timing information (TSC, TMA, MTC, CYC) provided during or after a powerdown state.

Power Event Trace is enabled via IA32_RTIT_CTL.PwrEvtEn[4]. Support for Power Event Tracing is introduced in Intel Atom processors based on the Goldmont Plus microarchitecture.

35.2.4 Event Tracing

Event Trace is a capability that exposes details about the asynchronous events, when they are generated, and when their corresponding software event handler completes execution. These include:

- Interrupts, including NMI and SMI, including the interrupt vector when defined.
- Faults, exceptions including the fault vector.
 - Page faults additionally include the page fault address, when in context.
- Event handler returns, including IRET and RSM.
- VM exits and VM entries.¹
 - VM exits include the values written to the “exit reason” and “exit qualification” VMCS fields.
- INIT and SIPI events.
- TSX aborts, including the abort status returned for the RTM instructions.
- Shutdown.

Additionally, it provides indication of the status of the Interrupt Flag (IF), to indicate when interrupts are masked.

Event Trace is enabled via `IA32_RTIT_CTL.EventEn[31]`. Event Trace information is conveyed in Control Flow Event (CFE) and Event Data (EVD) packets, as well as the legacy `MODE.Exec` packet. See Section 35.4.2 for packet details. Support for Event Trace is introduced in Intel® processors based on Gracemont microarchitecture.

35.2.5 Trace Filtering

Intel Processor Trace provides filtering capabilities, by which the debug/profile tool can control what code is traced.

35.2.5.1 Filtering by Current Privilege Level (CPL)

Intel PT provides the ability to configure a logical processor to generate trace packets only when `CPL = 0`, when `CPL > 0`, or regardless of CPL.

CPL filtering ensures that no IPs or other architectural state information associated with the filtered CPL can be seen in the log. For example, if the processor is configured to trace only when `CPL > 0`, and software executes `SYSCALL` (changing the CPL to 0), the destination IP of the `SYSCALL` will be suppressed from the generated packet (see the discussion of `TIP.PGD` in Section 35.4.2.5).

It should be noted that CPL is always 0 in real-address mode and that CPL is always 3 in virtual-8086 mode. To trace code in these modes, filtering should be configured accordingly.

When software is executing in a non-enabled CPL, `ContextEn` is cleared. See Section 35.2.6.1 for details.

35.2.5.2 Filtering by CR3

Intel PT supports a CR3-filtering mechanism by which the generation of packets containing architectural states can be enabled or disabled based on the value of CR3. A debugger can use CR3 filtering to trace only a single application without context switching the state of the RTIT MSRs. For the reconstruction of traces from software with multiple threads, debug software may wish to context-switch for the state of the RTIT MSRs (if the operating system does not provide context-switch support) to separate the output for the different threads (see Section 35.3.5, “Context Switch Consideration”).

To trace for only a single CR3 value, software can write that value to the `IA32_RTIT_CR3_MATCH` MSR, and set `IA32_RTIT_CTL.CR3Filter`. When CR3 value does not match `IA32_RTIT_CR3_MATCH` and `IA32_RTIT_CTL.CR3Filter` is 1, `ContextEn` is forced to 0, and packets containing architectural states will not be generated. Some other packets can be generated when `ContextEn` is 0; see Section 35.2.6.3 for details. When CR3 does match `IA32_RTIT_CR3_MATCH` (or when `IA32_RTIT_CTL.CR3Filter` is 0), CR3 filtering does not force `ContextEn` to 0 (although it could be 0 due to other filters or modes).

1. Logging of VMX transitions depends on VMCS configuration, see Section 35.5.1.

CR3 matches IA32_RTIT_CR3_MATCH if the two registers are identical for bits 63:12, or 63:5 when in PAE paging mode; the lower 5 bits of CR3 and IA32_RTIT_CR3_MATCH are ignored. CR3 filtering is independent of the value of CR0.PG.

When CR3 filtering is in use, PIP packets may still be seen in the log if the processor is configured to trace when CPL = 0 (IA32_RTIT_CTL.OS = 1). If not, no PIP packets will be seen.

35.2.5.3 Filtering by IP

Trace packet generation with configurable filtering by IP is supported if CPUID.14H.00H:EBX[2] = 1. Intel PT can be configured to enable the generation of packets containing architectural states only when the processor is executing code within certain IP ranges. If the IP is outside of these ranges, generation of some packets is blocked.

IP filtering is enabled using the ADDRn_CFG fields in the IA32_RTIT_CTL MSR (Section 35.2.8.2), where the digit 'n' is a zero-based number that selects which address range is being configured. Each ADDRn_CFG field configures the use of the register pair IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B (Section 35.2.8.5). IA32_RTIT_ADDRn_A defines the base and IA32_RTIT_ADDRn_B specifies the limit of the range in which tracing is enabled. Thus each range, referred to as the ADDRn range, is defined by [IA32_RTIT_ADDRn_A, IA32_RTIT_ADDRn_B]. There can be multiple such ranges, software can query CPUID (Section 35.3.1) for the number of ranges supported on a processor.

Default behavior (ADDRn_CFG=0) defines no IP filter range, meaning FilterEn is always set. In this case code at any IP can be traced, though other filters, such as CR3 or CPL, could limit tracing. When ADDRn_CFG is set to enable IP filtering (see Section 35.3.1), tracing will commence when a taken branch or event is seen whose target address is in the ADDRn range.

While inside a tracing region and with FilterEn is set, leaving the tracing region may only be detected once a taken branch or event with a target outside the range is retired. If an ADDRn range is entered or exited by executing the next sequential instruction, rather than by a control flow transfer, FilterEn may not toggle immediately. See Section 35.2.6.5 for more details on FilterEn.

Note that these address range base and limit values are inclusive, such that the range includes the first and last instruction whose first instruction byte is in the ADDRn range.

Depending upon processor implementation, IP filtering may be based on linear or effective address. This can cause different behavior between implementations if CSbase is not equal to zero or in real mode. See Section 35.3.1.1 for details. Software can query CPUID to determine filters are based on linear or effective address (Section 35.3.1).

Note that some packets, such as MTC (Section 35.3.7) and other timing packets, do not depend on FilterEn. For details on which packets depend on FilterEn, and hence are impacted by IP filtering, see Section 35.4.1.

TraceStop

The ADDRn ranges can also be configured to cause tracing to be disabled upon entry to the specified region. This is intended for cases where unexpected code is executed, and the user wishes to immediately stop generating packets in order to avoid overwriting previously written packets.

The TraceStop mechanism works much the same way that IP filtering does, and uses the same address comparison logic. The TraceStop region base and limit values are programmed into one or more ADDRn ranges, but IA32_RTIT_CTL.ADDRn_CFG is configured with the TraceStop encoding. Like FilterEn, TraceStop is detected when a taken branch or event lands in a TraceStop region.

Further, TraceStop requires that TriggerEn=1 at the beginning of the branch/event, and ContextEn=1 upon completion of the branch/event. When this happens, the CPU will set IA32_RTIT_STATUS.Stopped, thereby clearing TriggerEn and hence disabling packet generation. This may generate a TIP.PGD packet with the target IP of the branch or event that entered the TraceStop region. Finally, a TraceStop packet will be inserted, to indicate that the condition was hit.

If a TraceStop condition is encountered during buffer overflow (Section 35.3.8), it will not be dropped, but will instead be signaled once the overflow has resolved.

Note that a TraceStop event does not guarantee that all internally buffered packets are flushed out of internal buffers. To ensure that this has occurred, the user should clear TraceEn.

To resume tracing after a TraceStop event, the user must first disable Intel PT by clearing IA32_RTIT_CTL.TraceEn before the IA32_RTIT_STATUS.Stopped bit can be cleared. At this point Intel PT can be reconfigured, and tracing resumed.

Note that the IA32_RTIT_STATUS.Stopped bit can also be set using the ToPA STOP bit. See Section 35.2.7.2.

IP Filtering Example

The following table gives an example of IP filtering behavior. Assume that IA32_RTIT_ADDRn_A = the IP of RangeBase, and that IA32_RTIT_ADDRn_B = the IP of RangeLimit, while IA32_RTIT_CTL.ADDRn_CFG = 0x1 (enable ADDRn range as a FilterEn range).

Table 35-2. IP Filtering Packet Example

Code Flow	Packets
<pre> Bar: jmp RangeBase // jump into filter range RangeBase: jcc Foo // not taken add eax, 1 Foo: jmp RangeLimit+1 // jump out of filter range RangeLimit: nop jcc Bar </pre>	<pre> TIP.PGE(RangeBase) TNT(0) TIP.PGD(RangeLimit+1) </pre>

IP Filtering and TraceStop

It is possible for the user to configure IP filter range(s) and TraceStop range(s) that overlap. In this case, code executing in the non-overlapping portion of either range will behave as would be expected from that range. Code executing in the overlapping range will get TraceStop behavior.

35.2.6 Packet Generation Enable Controls

Intel Processor Trace includes a variety of controls that determine whether a packet is generated. In general, most packets are sent only if Packet Enable (**PacketEn**) is set. PacketEn is an internal state maintained in hardware in response to software configurable enable controls, PacketEn is not visible to software directly. The relationship of PacketEn to the software-visible controls in the configuration MSRs is described in this section.

35.2.6.1 Packet Enable (PacketEn)

When PacketEn is set, the processor is in the mode that Intel PT is monitoring. PacketEn is composed of other states according to this relationship:

```
PacketEn := TriggerEn AND ContextEn AND FilterEn AND BranchEn
```

These constituent controls are detailed in the following subsections.

PacketEn ultimately determines when the processor is tracing. When PacketEn is set, all control flow packets are enabled. When PacketEn is clear, no control flow packets are generated, though other packets (timing and book-keeping packets) may still be sent. See Section 35.2.7 for details of PacketEn and packet generation.

Note that, on processors that do not support IP filtering (i.e., CPUID.14H.00H:EBX[2] = 0), FilterEn is treated as always set.

35.2.6.2 Trigger Enable (TriggerEn)

Trigger Enable (**TriggerEn**) is the primary indicator that trace packet generation is active. TriggerEn is set when IA32_RTIT_CTL.TraceEn is set, and cleared by any of the following conditions:

- TraceEn is cleared by software.

- A TraceStop condition is encountered and IA32_RTIT_STATUS.Stopped is set.
- IA32_RTIT_STATUS.Error is set due to an operational error (see Section 35.3.10).

Software can discover the current TriggerEn value by reading the IA32_RTIT_STATUS.TriggerEn bit. When TriggerEn is clear, tracing is inactive and no packets are generated.

35.2.6.3 Context Enable (ContextEn)

Context Enable (**ContextEn**) indicates whether the processor is in the state or mode that software configured hardware to trace. For example, if execution with CPL = 0 code is not being traced (IA32_RTIT_CTL.OS = 0), then ContextEn will be 0 when the processor is in CPL0.

Software can discover the current ContextEn value by reading the IA32_RTIT_STATUS.ContextEn bit. ContextEn is defined as follows:

```
ContextEn = !((IA32_RTIT_CTL.OS = 0 AND CPL = 0) OR
(IA32_RTIT_CTL.USER = 0 AND CPL > 0) OR (IS_IN_A_PRODUCTION_ENCLAVE1) OR
(IA32_RTIT_CTL.CR3Filter = 1 AND IA32_RTIT_CR3_MATCH does not match CR3))
```

If the clearing of ContextEn causes PacketEn to be cleared, a Packet Generation Disable (TIP.PGD) packet is generated, but its IP payload is suppressed. If the setting of ContextEn causes PacketEn to be set, a Packet Generation Enable (TIP.PGE) packet is generated.

When ContextEn is 0, control flow packets (TNT, FUP, TIP.*, MODE.*) are not generated, and no Linear Instruction Pointers (LIPs) are exposed. However, some packets, such as MTC and PSB (see Section 35.4.2.16 and Section 35.4.2.17), may still be generated while ContextEn is 0. For details of which packets are generated only when ContextEn is set, see Section 35.4.1.

The processor does not update ContextEn when TriggerEn = 0.

The value of ContextEn will toggle only when TriggerEn = 1.

35.2.6.4 Branch Enable (BranchEn)

This value is based purely on the IA32_RTIT_CTL.BranchEn value. If **BranchEn** is not set, then relevant COFI packets (TNT, TIP*, FUP, MODE.*) are suppressed. Other packets related to timing (TSC, TMA, MTC, CYC), as well as PSB, will be generated normally regardless. Further, PIP and VMCS continue to be generated, as indicators of what software is running.

35.2.6.5 Filter Enable (FilterEn)

Filter Enable indicates that the Instruction Pointer (IP) is within the range of IPs that Intel PT is configured to watch. Software can get the state of Filter Enable by a RDMSR of IA32_RTIT_STATUS.FilterEn. For details on configuration and use of IP filtering, see Section 35.2.5.3.

On clearing of FilterEn that also clears PacketEn, a Packet Generation Disable (TIP.PGD) will be generated, but unlike the ContextEn case, the IP payload may not be suppressed. For direct, unconditional branches, as well as for indirect branches (including RETs), the PGD generated by leaving the tracing region and clearing FilterEn will contain the target IP. This means that IPs from outside the configured range can be exposed in the trace, as long as they are within context.

When FilterEn is 0, control flow packets are not generated (e.g., TNT, TIP). However, some packets, such as PIP, MTC, and PSB, may still be generated while FilterEn is clear. For details on packet enable dependencies, see Section 35.4.1.

After TraceEn is set, FilterEn is set to 1 at all times if there is no IP filter range configured by software (IA32_RTIT_CTL.ADDRn_CFG != 1, for all n), or if the processor does not support IP filtering (i.e., CPUID.14H.00H:EBX[2] = 0). FilterEn will toggle only when TraceEn=1 and ContextEn=1, and when at least one range is configured for IP filtering.

1. Trace packets generation is disabled in a production enclave, see Section 35.2.9.5. See the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D, about differences between a production enclave and a debug enclave.

35.2.7 Trace Output

Intel PT output should be viewed independently from trace content and filtering mechanisms. The options available for trace output can vary across processor generations and platforms.

Trace output is written out using one of the following output schemes, as configured by the ToPA and FabricEn bit fields of IA32_RTIT_CTL (see Section 35.2.8.2):

- A single, contiguous region of physical address space.
- A collection of variable-sized regions of physical memory. These regions are linked together by tables of pointers to those regions, referred to as Table of Physical Addresses (**ToPA**). The trace output stores bypass the caches and the TLBs, but are not serializing. This is intended to minimize the performance impact of the output.
- A platform-specific trace transport subsystem.

Regardless of the output scheme chosen, Intel PT stores bypass the processor caches by default. This ensures that they don't consume precious cache space, but they do not have the serializing aspects associated with un-cacheable (UC) stores. Software should avoid using MTRRs to mark any portion of the Intel PT output region as UC, as this may override the behavior described above and force Intel PT stores to UC, thereby incurring severe performance impact.

There is no guarantee that a packet will be written to memory or other trace endpoint after some fixed number of cycles after a packet-producing instruction executes. The only way to assure that all packets generated have reached their endpoint is to clear TraceEn and follow that with a store, fence, or serializing instruction; doing so ensures that all buffered packets are flushed out of the processor.

35.2.7.1 Single Range Output

When IA32_RTIT_CTL.ToPA and IA32_RTIT_CTL.FabricEn bits are clear, trace packet output is sent to a single, contiguous memory (or MMIO if DRAM is not available) range defined by a base address in IA32_RTIT_OUTPUT_BASE (Section 35.2.8.7) and mask value in IA32_RTIT_OUTPUT_MASK_PTRS (Section 35.2.8.8). The current write pointer in this range is also stored in IA32_RTIT_OUTPUT_MASK_PTRS. This output range is circular, meaning that when the writes wrap around the end of the buffer they begin again at the base address.

This output method is best suited for cases where Intel PT output is either:

- Configured to be directed to a sufficiently large contiguous region of DRAM.
- Configured to go to an MMIO debug port, in order to route Intel PT output to a platform-specific trace endpoint (e.g., JTAG). In this scenario, a specific range of addresses is written in a circular manner, and SoC will intercept these writes and direct them to the proper device. Repeated writes to the same address do not overwrite each other, but are accumulated by the debugger, and hence no data is lost by the circular nature of the buffer.

The processor will determine the address to which to write the next trace packet output byte as follows:

```
OutputBase[63:0] := IA32_RTIT_OUTPUT_BASE[63:0]
OutputMask[63:0] := ZeroExtend64(IA32_RTIT_OUTPUT_MASK_PTRS[31:0])
OutputOffset[63:0] := ZeroExtend64(IA32_RTIT_OUTPUT_MASK_PTRS[63:32])
trace_store_phys_addr := (OutputBase & ~OutputMask) + (OutputOffset & OutputMask)
```


Single-Range Output Errors

If the output base and mask are not properly configured by software, an operational error (see Section 35.3.10) will be signaled, and tracing disabled. Error scenarios with single-range output are:

- Mask value is non-contiguous.
IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTablePointer value has a 0 in a less significant bit position than the most significant bit containing a 1.
- Base address and Mask are mis-aligned, and have overlapping bits set.
IA32_RTIT_OUTPUT_BASE && IA32_RTIT_OUTPUT_MASK_PTRS[31:0] > 0.
- Illegal Output Offset
IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than the mask value IA32_RTIT_OUTPUT_MASK_PTRS[31:0].

Also note that errors can be signaled due to trace packet output overlapping with restricted memory, see Section 35.2.7.4.

35.2.7.2 Table of Physical Addresses (ToPA)

When IA32_RTIT_CTL.ToPA is set and IA32_RTIT_CTL.FabricEn is clear, the ToPA output mechanism is utilized. The ToPA mechanism uses a linked list of tables; see Figure 35-1 for an illustrative example. Each entry in the table contains some attribute bits, a pointer to an output region, and the size of the region. The last entry in the table may hold a pointer to the next table. This pointer can either point to the top of the current table (for circular array) or to the base of another table. The table size is not fixed, since the link to the next table can exist at any entry.

The processor treats the various output regions referenced by the ToPA table(s) as a unified buffer. This means that a single packet may span the boundary between one output region and the next.

The ToPA mechanism is controlled by three values maintained by the processor:

- **proc_trace_table_base.**
This is the physical address of the base of the current ToPA table. When tracing is enabled, the processor loads this value from the IA32_RTIT_OUTPUT_BASE MSR. While tracing is enabled, the processor updates the IA32_RTIT_OUTPUT_BASE MSR with changes to proc_trace_table_base, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_base.
- **proc_trace_table_offset.**
This indicates the entry of the current table that is currently in use. (This entry contains the address of the current output region.) When tracing is enabled, the processor loads the value from bits 31:7 (MaskOrTableOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS into bits 27:3 of proc_trace_table_offset. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset with changes to proc_trace_table_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_table_offset.
- **proc_trace_output_offset.**
This is a pointer into the current output region and indicates the location of the next write. When tracing is enabled, the processor loads this value from bits 63:32 (OutputOffset) of the IA32_RTIT_OUTPUT_MASK_PTRS. While tracing is enabled, the processor updates IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset with changes to proc_trace_output_offset, but these updates may not be synchronous to software execution. When tracing is disabled, the processor ensures that the MSR contains the latest value of proc_trace_output_offset.

Figure 35-1 provides an illustration (not to scale) of the table and associated pointers.

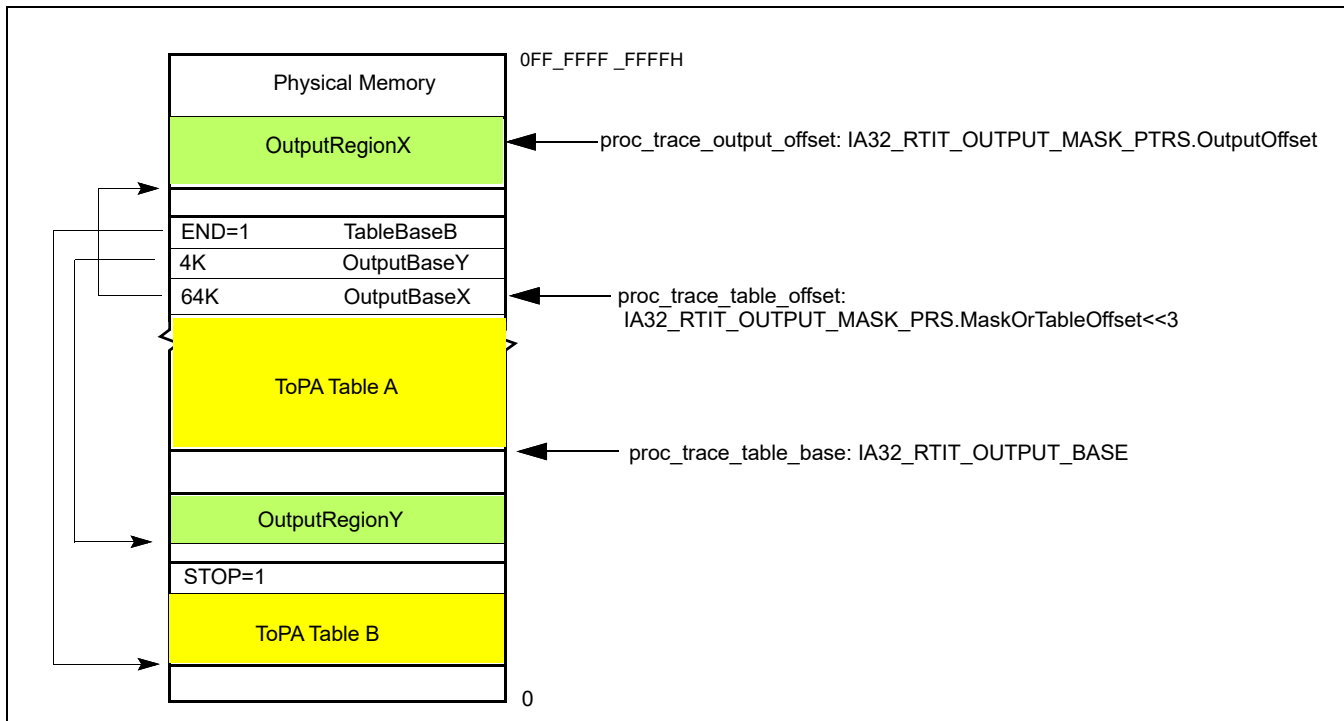


Figure 35-1. ToPA Memory Illustration

With the ToPA mechanism, the processor writes packets to the current output region (identified by `proc_trace_table_base` and the `proc_trace_table_offset`). The offset within that region to which the next byte will be written is identified by `proc_trace_output_offset`. When that region is filled with packet output (thus `proc_trace_output_offset = RegionSize-1`), `proc_trace_table_offset` is moved to the next ToPA entry, `proc_trace_output_offset` is set to 0, and packet writes begin filling the new output region specified by `proc_trace_table_offset`.

As packets are written out, each store derives its physical address as follows:

```
trace_store_phys_addr := Base address from current ToPA table entry +  
proc_trace_output_offset
```

Eventually, the regions represented by all entries in the table may become full, and the final entry of the table is reached. An entry can be identified as the final entry because it has either the END or STOP attribute. The END attribute indicates that the address in the entry does not point to another output region, but rather to another ToPA table. The STOP attribute indicates that tracing will be disabled once the corresponding region is filled. See Table 35-3 and the section that follows for details on STOP.

When an END entry is reached, the processor loads `proc_trace_table_base` with the base address held in this END entry, thereby moving the current table pointer to this new table. The `proc_trace_table_offset` is reset to 0, as is the `proc_trace_output_offset`, and packet writes will resume at the base address indicated in the first entry.

If the table has no STOP or END entry, and trace-packet generation remains enabled, eventually the maximum table size will be reached (`proc_trace_table_offset = 0FFFFFF8H`). In this case, the `proc_trace_table_offset` and `proc_trace_output_offset` are reset to 0 (wrapping back to the beginning of the current table) once the last output region is filled.

It is important to note that processor updates to the IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs are asynchronous to instruction execution. Thus, reads of these MSRs while Intel PT is enabled may return stale values. Like all IA32_RTIT_* MSRs, the values of these MSRs should not be trusted or saved unless trace packet generation is first disabled by clearing IA32_RTIT_CTL.TraceEn. This ensures that the output MSR values account for all packets generated to that point, after which the processor will cease updating the output MSR values until tracing resumes.¹

The processor may cache internally any number of entries from the current table or from tables that it references (directly or indirectly). If tracing is enabled, the processor may ignore or delay detection of modifications to these tables. To ensure that table changes are detected by the processor in a predictable manner, software should clear TraceEn before modifying the current table (or tables that it references) and only then re-enable packet generation.

Single Output Region ToPA Implementation

The first processor generation to implement Intel PT supports only ToPA configurations with a single ToPA entry followed by an END entry that points back to the first entry (creating one circular output buffer). Such processors enumerate CPUID.14H.00H:ECX.MENTRY[1] = 0 and CPUID.14H.00H:ECX.TOPAOUT[0] = 1.

If CPUID.14H.00H:ECX.MENTRY[1] = 0, ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Hence only one contiguous block can be used as output.

The lone output entry can have INT or STOP set, but nonetheless must be followed by an END entry as described above. Note that, if INT=1, the PMI will actually be delivered before the region is filled.

ToPA Table Entry Format

The format of ToPA table entries is shown in Figure 35-2. The size of the address field is determined by the processor’s physical-address width MAXPHYADDR.²

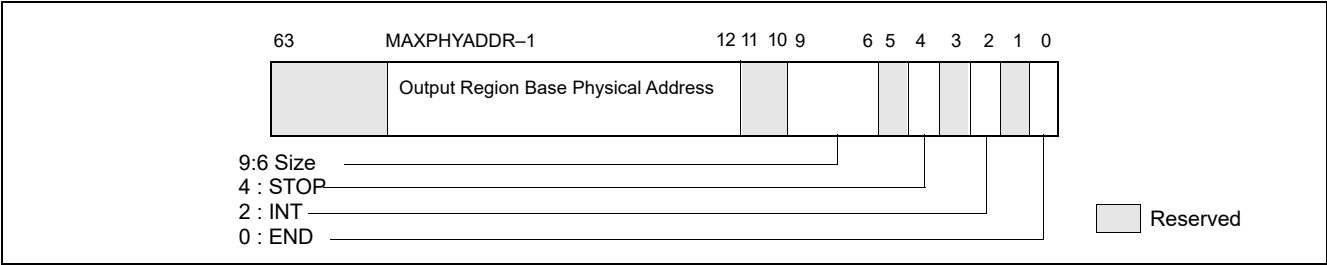


Figure 35-2. Layout of ToPA Table Entry

Table 35-3 describes the details of the ToPA table entry fields. If reserved bits are set to 1, an error is signaled.

1. Although WRMSR is a serializing instruction, the execution of WRMSR that forces packet writes by clearing TraceEn does not itself cause these writes to be globally observed.

2. Usually, MAXPHYADDR is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Table 35-3. ToPA Table Entry Fields

ToPA Entry Field	Description
Output Region Base Physical Address	If END=0, this is the base physical address of the output region specified by this entry. Note that all regions must be aligned based on their size. Thus a 2M region must have bits 20:12 clear. If the region is not properly aligned, an operational error will be signaled when the entry is reached. If END=1, this is the 4K-aligned base physical address of the next ToPA table (which may be the base of the current table, or the first table in the linked list if a circular buffer is desired). If the processor supports only a single ToPA output region (see above), this address must be the value currently in the IA32_RTIT_OUTPUT_BASE MSR.
Size	Indicates the size of the associated output region. Encodings are: 0: 4K, 1: 8K, 2: 16K, 3: 32K, 4: 64K, 5: 128K, 6: 256K, 7: 512K, 8: 1M, 9: 2M, 10: 4M, 11: 8M, 12: 16M, 13: 32M, 14: 64M, 15: 128M This field is ignored if END=1.
STOP	When the output region indicated by this entry is filled, software should disable packet generation. This will be accomplished by setting IA32_RTIT_STATUS.Stopped, which clears TriggerEn. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
INT	When the output region indicated by this entry is filled, signal PerfMon LVT interrupt. Note that if both INT and STOP are set in the same entry, the STOP will happen before the INT. Thus the interrupt handler should expect that the IA32_RTIT_STATUS.Stopped bit will be set, and will need to be reset before tracing can be resumed. This bit must be 0 if END=1; otherwise it is treated as reserved bit violation (see ToPA Errors).
END	If set, indicates that this is an END entry, and thus the address field points to a table base rather than an output region base. If END=1, INT and STOP must be set to 0; otherwise it is treated as reserved bit violation (see ToPA Errors). The Size field is ignored in this case. If the processor supports only a single ToPA output region (see above), END must be set in the second table entry.

ToPA STOP

Each ToPA entry has a STOP bit. If this bit is set, the processor will set the IA32_RTIT_STATUS.Stopped bit when the corresponding trace output region is filled. This will clear TriggerEn and thereby cease packet generation. See Section 35.2.8.4 for details on IA32_RTIT_STATUS.Stopped. This sequence is known as “ToPA Stop”.

No TIP.PGD packet will be seen in the output when the ToPA stop occurs, since the disable happens only when the region is already full. When this occurs, output ceases after the last byte of the region is filled, which may mean that a packet is cut off in the middle. Any packets remaining in internal buffers are lost and cannot be recovered.

When ToPA stop occurs, the IA32_RTIT_OUTPUT_BASE MSR will hold the base address of the table whose entry had STOP=1. IA32_RTIT_OUTPUT_MASK_PTRS.MaskOrTableOffset will hold the index value for that entry, and the IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset should be set to the size of the region minus one.

Note that this means the offset pointer is pointing to the next byte after the end of the region, a configuration that would produce an operational error if the configuration remained when tracing is re-enabled with IA32_RTIT_STATUS.Stopped cleared.

ToPA PMI

Each ToPA entry has an INT bit. If this bit is set, the processor will signal a performance-monitoring interrupt (PMI) when the corresponding trace output region is filled. This interrupt is not precise, and it is thus likely that writes to the next region will occur by the time the interrupt is taken.

The following steps should be taken to configure this interrupt:

1. Enable PMI via the LVT Performance Monitor register (at MMIO offset 340H in xAPIC mode; via MSR 834H in x2APIC mode). See the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B, for more details on this register. For ToPA PMI, set all fields to 0, save for the interrupt vector, which can be selected by software.
2. Set up an interrupt handler to service the interrupt vector that a ToPA PMI can raise.

3. Set the interrupt flag by executing STI.
4. Set the INT bit in the ToPA entry of interest and enable packet generation, using the ToPA output option. Thus, TraceEn=ToPA=1 in the IA32_RTIT_CTL MSR.

Once the INT region has been filled with packet output data, the interrupt will be signaled. This PMI can be distinguished from others by checking bit 55 (Trace_ToPA_PMI) of the IA32_PERF_GLOBAL_STATUS MSR (MSR 38EH). Once the ToPA PMI handler has serviced the relevant buffer, writing 1 to bit 55 of the MSR at 390H (IA32_GLOBAL_STATUS_RESET) clears IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI.

Intel PT is not frozen on PMI, and thus the interrupt handler will be traced (though filtering can prevent this). The Freeze_PerfMon_on_PMI and Freeze_LBRs_on_PMI settings in IA32_DEBUGCTL will be applied on ToPA PMI just as on other PMIs, and hence PerfMon counters are frozen.

Assuming the PMI handler wishes to read any buffered packets for persistent output, or wishes to modify any Intel PT MSRs, software should first disable packet generation by clearing TraceEn. This ensures that all buffered packets are written to memory and avoids tracing of the PMI handler. The configuration MSRs can then be used to determine where tracing has stopped. If packet generation is disabled by the handler, it should then be manually re-enabled before the IRET if continued tracing is desired.

In rare cases, it may be possible to trigger a second ToPA PMI before the first is handled. This can happen if another ToPA region with INT=1 is filled before, or shortly after, the first PMI is taken, perhaps due to EFLAGS.IF being cleared for an extended period of time. This can manifest in two ways: either the second PMI is triggered before the first is taken, and hence only one PMI is taken, or the second is triggered after the first is taken, and thus will be taken when the handler for the first completes. Software can minimize the likelihood of the second case by clearing TraceEn at the beginning of the PMI handler. Further, it can detect such cases by then checking the Interrupt Request Register (IRR) for PMI pending, and checking the ToPA table base and off-set pointers (in IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS) to see if multiple entries with INT=1 have been filled.

PMI Preservation

In some cases a ToPA PMI may be taken after completion of an XSAVES instruction that saves Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, the PMI Preservation feature has been added. Support for this feature is indicated by CPUID.14H.00H:EBX[6].

When IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, PMI preservation is enabled. When a ToPA region with INT=1 is filled, a PMI is pended and the new IA32_RTIT_STATUS.PendToPAPMI[7] is set to 1. If this bit is set when Intel PT is enabled, such that IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a ToPA PMI is pended. This behavior ensures that any ToPA PMI that is pended during XSAVES, and hence can't be properly handled, will be re-pended when the saved PT state is restored.

When this feature is enabled, the PMI handler should take the following actions:

1. Ignore ToPA PMIs that are taken when TraceEn = 0. This indicates that the PMI was pended during Intel PT disable, and the PendToPAPMI flag will ensure that the PMI is re-pended once Intel PT is re-enabled in the same context. For this reason, the PendToPAPMI bit should be left set to 1.
2. If TraceEn=1 and the PMI can be properly handled, clear the new PendTopaPMI bit. This will ensure that additional, spurious ToPA PMIs are not taken. It is required that PendToPAPMI is cleared before the PMI LVT mask is cleared in the APIC, and before any clearing of either LBRs_FROZEN or COUNTERS_FROZEN in IA32_PERF_GLOBAL_STATUS.

ToPA PMI and Single Output Region ToPA Implementation

A processor that supports only a single ToPA output region implementation (such that only one output region is supported; see above) will attempt to signal a ToPA PMI interrupt before the output wraps and overwrites the top of the buffer. To support this functionality, the PMI handler should disable packet generation as soon as possible.

Due to PMI skid, it is possible that, in rare cases, the wrap will have occurred before the PMI is delivered. Software can avoid this by setting the STOP bit in the ToPA entry (see Table 35-3); this will disable tracing once the region is filled, and no wrap will occur. This approach has the downside of disabling packet generation so that some of the instructions that led up to the PMI will not be traced. If the PMI skid is significant enough to cause the region to fill and tracing to be disabled, the PMI handler will need to clear the IA32_RTIT_STATUS.Stopped indication before tracing can resume.

ToPA PMI and XSAVES/XRSTORS State Handling

In some cases the ToPA PMI may be taken after completion of an XSAVES instruction that switches Intel PT state, and in such cases any modification of Intel PT MSRs within the PMI handler will not persist when the saved Intel PT context is later restored with XRSTORS. To account for such a scenario, it is recommended that the Intel PT output configuration be modified by altering the ToPA tables themselves, rather than the Intel PT output MSRs. On processors that support PMI preservation (CPUID.14H.00H:EBX[6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PMI that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendTopaPMI[7] = 1. A PMI will then be pended when the saved PT context is later restored.

Table 35-4 depicts a recommended PMI handler algorithm for managing multi-region ToPA output and handling ToPA PMIs that may arrive between XSAVES and XRSTORS, if PMI preservation is not in use. This algorithm is flexible to allow software to choose between adding entries to the current ToPA table, adding a new ToPA table, or using the current ToPA table as a circular buffer. It assumes that the ToPA entry that triggers the PMI is not the last entry in the table, which is the recommended treatment.

Table 35-4. Algorithm to Manage Intel PT ToPA PMI and XSAVES/XRSTORS

Pseudo Code Flow
<pre> IF (IA32_PERF_GLOBAL_STATUS.ToPA) Save IA32_RTIT_CTL value; IF (IA32_RTIT_CTL.TraceEN) Disable Intel PT by clearing TraceEn; FI; IF (there is space available to grow the current ToPA table) Add one or more ToPA entries after the last entry in the ToPA table; Point new ToPA entry address field(s) to new output region base(s); ELSE Modify an upcoming ToPA entry in the current table to have END=1; IF (output should transition to a new ToPA table) Point the address of the "END=1" entry of the current table to the new table base; ELSE /* Continue to use the current ToPA table, make a circular. */ Point the address of the "END=1" entry to the base of the current table; Modify the ToPA entry address fields for filled output regions to point to new, unused output regions; /* Filled regions are those with index in the range of 0 to (IA32_RTIT_MASK_PTRS.MaskOrTableOffset -1). */ FI; FI; Restore saved IA32_RTIT_CTL.value; FI; </pre>

ToPA Errors

When a malformed ToPA entry is found, an **operational error** results (see Section 35.3.10). A malformed entry can be any of the following:

- ToPA entry reserved bit violation.**
This describes cases where a bit marked as reserved in Section 35.2.7.2 above is set to 1.
- ToPA alignment violation.**
This includes cases where illegal ToPA entry base address bits are set to 1:
 - ToPA table base address is not 4KB-aligned. The table base can be from a WRMSR to IA32_RTIT_OUTPUT_BASE, or from a ToPA entry with END=1.
 - ToPA entry base address is not aligned to the ToPA entry size (e.g., a 2MB region with base address[20:12] not equal to 0), for ToPA entries with END=0.
 - ToPA entry base address sets physical address bits [in the range 63:MAXPHYADDR](#).¹

3. **Illegal ToPA Output Offset.**
IA32_RTIT_OUTPUT_MASK_PTRS.OutputOffset is greater than or equal to the size of the current ToPA output region size.
4. **ToPA rules violations.**
These are similar to ToPA entry reserved bit violations; they are cases when a ToPA entry is encountered with illegal field combinations. They include the following:

a. Setting the STOP or INT bit on an entry with END=1.

b. Setting the END bit in entry 0 of a ToPA table.

c. On processors that support only a single ToPA entry (see above), two additional illegal settings apply:

i) ToPA table entry 1 with END=0.

ii) ToPA table entry 1 with base address not matching the table base.
- In all cases, the error will be logged by setting IA32_RTIT_STATUS.Error, thereby disabling tracing when the problematic ToPA entry is reached (when proc_trace_table_offset points to the entry containing the error). Any packet bytes that are internally buffered when the error is detected may be lost.
- Note that operational errors may also be signaled due to attempts to access restricted memory. See Section 35.2.7.4 for details.
- A tracing software have a range of flexibility using ToPA to manage the interaction of Intel PT with application buffers, see Section 35.4.2.26.
- 35.2.7.3 Trace Transport Subsystem
- When IA32_RTIT_CTL.FabricEn is set, the IA32_RTIT_CTL.ToPA bit is ignored, and trace output is written to the trace transport subsystem. The endpoints of this transport are platform-specific, and details of configuration options should refer to the specific platform documentation. The FabricEn bit is available to be set if CPUID.14H.00H:EBX[3] = 1.
- 35.2.7.4 Restricted Memory Access
- Packet output cannot be directed to any regions of memory that are restricted by the platform. In particular, all memory accesses on behalf of packet output are checked against the SMRR, PRMRR, and SEAMRR protected regions. If there is any overlap with these regions, trace data collection will not function properly. Exact processor behavior is implementation-dependent; Table 35-5 summarizes several scenarios.
- Table 35-5. Behavior on Restricted Memory Access
- | Scenario | Description |
|--|--|
| ToPA output region overlaps with SMRR, PRMRR, and SEAMRR protected regions | Stores to the restricted memory region will be dropped, and that packet data will be lost. Any attempt to read from that restricted region will return all 1s. The processor also may signal an error (Section 35.3.10) and disable tracing when the output pointer reaches the restricted region. If packet generation remains enabled, then packet output may continue once stores are no longer directed to restricted memory (on wrap, or if the output region is larger than the restricted memory region). |
| ToPA table overlaps with SMRR, PRMRR, and SEAMRR protected regions | The processor will signal an error (Section 35.3.10) and disable tracing when the ToPA write pointer (IA32_RTIT_OUTPUT_BASE + proc_trace_table_offset) enters the restricted region. |
- It should also be noted that packet output should not be routed to the 4KB APIC MMIO region, as defined by the IA32_APIC_BASE MSR. For details about the APIC, refer to the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A. No error is signaled for this case.
1. Usually, MAXPHYADDR is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52). If IA32_TME_ACTIVATE[0]= 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.
- 35-16 Vol. 3C

Modifications to Restricted Memory Regions

It is recommended that software disable packet generation before modifying the SMRRs to change the scope of the SMRR regions. This is because the processor reserves the right to cache any number of ToPA table entries internally, after checking them against restricted memory ranges. Once cached, the entries will not be checked again, meaning one could potentially route packet output to a newly restricted region. Software can ensure that any cached entries are written to memory by clearing IA32_RTIT_CTL.TraceEn.

35.2.8 Enabling and Configuration MSRs

35.2.8.1 General Considerations

Trace packet generation is enabled and configured by a collection of model-specific registers (MSRs), which are detailed below. Some notes on the configuration MSR behavior:

- If Intel Processor Trace is not supported by the processor (see Section 35.3.1), RDMSR or WRMSR of the IA32_RTIT_* MSRs will cause #GP.
- A WRMSR to any of the IA32_RTIT_* configuration MSRs while packet generation is enabled (IA32_RTIT_CTL.TraceEn=1) will generate a #GP exception. Packet generation must be disabled before the configuration MSRs can be changed.
Note: Software may write the same value back to IA32_RTIT_CTL without #GP, even if TraceEn=1.
- All configuration MSRs for Intel PT are duplicated per logical processor
- For each configuration MSR, any MSR write that attempts to change bits marked reserved, or utilize encodings marked reserved, will cause a #GP fault.
- All configuration MSRs for Intel PT are cleared on a warm or cold RESET.
 - If CPUID.14H.00H:EBX[2] = 1, only the TraceEn bit is cleared on warm RESET; though this may have the impact of clearing other bits in IA32_RTIT_STATUS. Other MSR values of the trace configuration MSRs are preserved on warm RESET.
- The semantics of MSR writes to trace configuration MSRs in this chapter generally apply to explicit WRMSR to these registers, using VMexit or VM entry MSR load list to these MSRs, XRSTORS with requested feature bit map including XSAVE map component of state_8 (corresponding to IA32_XSS[bit 8]), and the write to IA32_RTIT_CTL.TraceEn by XSAVES (Section 35.3.5.2).

35.2.8.2 IA32_RTIT_CTL MSR

IA32_RTIT_CTL, at address 570H, is the primary enable and control MSR for trace packet generation. Bit positions are listed in Table 35-6.

Table 35-6. IA32_RTIT_CTL MSR

Position	Bit Name	At Reset	Bit Description
0	TraceEn	0	If 1, enables tracing; else tracing is disabled. When this bit transitions from 1 to 0, all buffered packets are flushed out of internal buffers. A further store, fence, or architecturally serializing instruction may be required to ensure that packet data can be observed at the trace endpoint. See Section 35.2.8.3 for details of enabling and disabling packet generation. Note that the processor will clear this bit on #SMI (Section 35.2.9.3) and warm reset. Other MSR bits of IA32_RTIT_CTL (and other trace configuration MSRs) are not impacted by these events.
1	CYCEn	0	0: Disables CYC Packet (see Section 35.4.2.14). 1: Enables CYC Packet. This bit is reserved if CPUID.14H.00H:EBX[1] = 0.

Table 35-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
2	OS	0	0: Packet generation is disabled when CPL = 0. 1: Packet generation may be enabled when CPL = 0.
3	User	0	0: Packet generation is disabled when CPL > 0. 1: Packet generation may be enabled when CPL > 0.
4	PwrEvtEn	0	0: Power Event Trace packets are disabled. 1: Power Event Trace packets are enabled (see Section 35.2.3, “Power Event Tracing”).
5	FUPonPTW	0	0: PTW packets are not followed by FUPs. 1: PTW packets are followed by FUPs. This bit is reserved when CPUID.14H.00H:EBX[4] (“PTWRITE Supported”) is 0.
6	FabricEn	0	0: Trace output is directed to the memory subsystem, mechanism depends on IA32_RTIT_CTL.ToPA. 1: Trace output is directed to the trace transport subsystem, IA32_RTIT_CTL.ToPA is ignored. This bit is reserved if CPUID.14H.00H:ECX[3] = 0.
7	CR3Filter	0	0: Disables CR3 filtering. 1: Enables CR3 filtering. This bit is reserved if CPUID.14H.00H:EBX[0] (“CR3 Filtering Support”) is 0.
8	ToPA	0	0: Single-range output scheme enabled if CPUID.14H.00H:ECX.SNGL_RNG_OUT[2] = 1 and IA32_RTIT_CTL.FabricEn=0. 1: ToPA output scheme enabled (see Section 35.2.7.2) if CPUID.14H.00H:ECX.TOPAOUT[0] = 1, and IA32_RTIT_CTL.FabricEn=0. Note: WRMSR to IA32_RTIT_CTL that sets TraceEn but clears this bit and FabricEn would cause #GP, if CPUID.14H.00H:ECX.SNGL_RNG_OUT[2] = 0. WRMSR to IA32_RTIT_CTL that sets this bit causes #GP, if CPUID.14H.00H:ECX.TOPAOUT[0] = 0.
9	MTCEn	0	0: Disables MTC Packet (see Section 35.4.2.16). 1: Enables MTC Packet. This bit is reserved if CPUID.14H.00H:EBX[3] = 0.
10	TSCEn	0	0: Disable TSC packets. 1: Enable TSC packets (see Section 35.4.2.11).
11	DisRETC	0	0: Enable RET compression. 1: Disable RET compression (see Section 35.2.1.2).
12	PTWEn	0	0: PTWRITE packet generation disabled. 1: PTWRITE packet generation enabled (see Table 35-40 “PTW Packet Definition”). This bit is reserved when CPUID.14H.00H:EBX[4] (“PTWRITE Supported”) is 0.
13	BranchEn	0	0: Disable COFI-based packets. 1: Enable COFI-based packets: FUP, TIP, TIP.PGE, TIP.PGD, TNT, MODE.Exec, MODE.TSX. See Section 35.2.6.4 for details on BranchEn.

Table 35-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
17:14	MTCFreq	0	Defines MTC packet Frequency, which is based on the core crystal clock, or Always Running Timer (ART). MTC will be sent each time the selected ART bit toggles. The following Encodings are defined: 0: ART(0), 1: ART(1), 2: ART(2), 3: ART(3), 4: ART(4), 5: ART(5), 6: ART(6), 7: ART(7), 8: ART(8), 9: ART(9), 10: ART(10), 11: ART(11), 12: ART(12), 13: ART(13), 14: ART(14), 15: ART(15) Software must use CPUID to query the supported encodings in the processor, see Section 35.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.14H.00H:EBX[3] = 0.
18	Reserved	0	Must be 0.
22:19	CycThresh	0	CYC packet threshold, see Section 35.3.6 for details. CYC packets will be sent with the first eligible packet after N cycles have passed since the last CYC packet. If CycThresh is 0 then N=0, otherwise N is defined as $2^{(\text{CycThresh}-1)}$. The following Encodings are defined: 0: 0, 1: 1, 2: 2, 3: 4, 4: 8, 5: 16, 6: 32, 7: 64, 8: 128, 9: 256, 10: 512, 11: 1024, 12: 2048, 13: 4096, 14: 8192, 15: 16384 Software must use CPUID to query the supported encodings in the processor, see Section 35.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.14H.00H:EBX[1] = 0.
23	Reserved	0	Must be 0.
27:24	PSBFreq	0	Indicates the frequency of PSB packets. PSB packet frequency is based on the number of Intel PT packet bytes output, so this field allows the user to determine the increment of IA32_IA32_RTIT_STATUS.PacketByteCnt that should cause a PSB to be generated. Note that PSB insertion is not precise, but the average output bytes per PSB should approximate the SW selected period. The following Encodings are defined: 0: 2K, 1: 4K, 2: 8K, 3: 16K, 4: 32K, 5: 64K, 6: 128K, 7: 256K, 8: 512K, 9: 1M, 10: 2M, 11: 4M, 12: 8M, 13: 16M, 14: 32M, 15: 64M Software must use CPUID to query the supported encodings in the processor, see Section 35.3.1. Use of unsupported encodings will result in a #GP fault. This field is reserved if CPUID.14H.00H:EBX[1] = 0.
30:28	Reserved	0	Must be 0.
31	EventEn	0	0: Event Trace packets are disabled. 1: Event Trace packets are enabled. This bit is reserved when CPUID.14H.00H:EBX[7] ("Event Trace Supported") is 0.
35:32	ADDR0_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR0_A/B based on the following encodings: 0: ADDR0 range unused. 1: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 35.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR0_A..IA32_RTIT_ADDR0_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See 4.2.8 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGECNT[2:0] < 1.

Table 35-6. IA32_RTIT_CTL MSR (Contd.)

Position	Bit Name	At Reset	Bit Description
39:36	ADDR1_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR1_A/B based on the following encodings: 0: ADDR1 range unused. 1: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 35.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR1_A..IA32_RTIT_ADDR1_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 35.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGE CNT[2:0] < 2.
43:40	ADDR2_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR2_A/B based on the following encodings: 0: ADDR2 range unused. 1: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 35.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR2_A..IA32_RTIT_ADDR2_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 35.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGE CNT[2:0] < 3.
47:44	ADDR3_CFG	0	Configures the base/limit register pair IA32_RTIT_ADDR3_A/B based on the following encodings: 0: ADDR3 range unused. 1: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a FilterEn range. FilterEn will only be set when the IP is within this range, though other FilterEn ranges can additionally be used. See Section 35.2.5.3 for details on IP filtering. 2: The [IA32_RTIT_ADDR3_A..IA32_RTIT_ADDR3_B] range defines a TraceStop range. TraceStop will be asserted if code branches into this range. See Section 35.4.2.10 for details on TraceStop. 3..15: Reserved (#GP). This field is reserved if CPUID.14H.01H:EAX.RANGE CNT[2:0] < 4.
54:48	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
55	DisTNT	0	0: Include TNT packets in control flow trace. 1: Omit TNT packets from control flow trace. This bit is reserved when CPUID.14H.00H:EBX[8] ("TNT Disable Supported") is 0. See Section 35.3.9 for details.
56	InjectPsbPmiOnEnable	0	1: Enables use of IA32_RTIT_STATUS bits PendPSB[6] and PendTopaPMI[7], see Section 35.2.8.4, "IA32_RTIT_STATUS MSR," for behavior of these bits. 0: IA32_RTIT_STATUS bits 6 and 7 are ignored. This field is reserved if CPUID.14H.00H:EBX[6] = 0.
59:57	Reserved	0	Reserved only for future trace content enables, or address filtering configuration enables. Must be 0.
63:60	Reserved	0	Must be 0.

35.2.8.3 Enabling and Disabling Packet Generation with TraceEn

When TraceEn transitions from 0 to 1, Intel Processor Trace is enabled, and a series of packets may be generated. These packets help ensure that the decoder is aware of the state of the processor when the trace begins, and that it can keep track of any timing or state changes that may have occurred while packet generation was disabled. A full PSB+ (see Section 35.4.2.17) will be generated if IA32_RTIT_STATUS.PacketByteCnt=0, and may be generated in other cases as well. Otherwise, timing packets will be generated, including TSC, TMA, and CBR (see Section 35.4.1.1).

In addition to the packets discussed above, if and when PacketEn (Section 35.2.6.1) transitions from 0 to 1 (which may happen immediately, depending on filtering settings), a TIP.PGE packet (Section 35.4.2.3) will be generated.

When TraceEn is set, the processor may read ToPA entries from memory and cache them internally. For this reason, software should disable packet generation before making modifications to the ToPA tables (or changing the configuration of restricted memory regions). See Section 35.7 for more details of packets that may be generated with modifications to TraceEn.

Disabling Packet Generation

Clearing TraceEn causes any packet data buffered within the logical processor to be flushed out, after which the output MSRs (IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS) will have stable values. When output is directed to memory, a store, fence, or architecturally serializing instruction may be required to ensure that the packet data is globally observed. No special packets are generated by disabling packet generation, though a TIP.PGD may result if PacketEn=1 at the time of disable.

Other Writes to IA32_RTIT_CTL

Any attempt to modify IA32_RTIT_CTL while TraceEn is set will result in a general-protection fault (#GP) unless the same write also clears TraceEn. However, writes to IA32_RTIT_CTL that do not modify any bits will not cause a #GP, even if TraceEn remains set.

35.2.8.4 IA32_RTIT_STATUS MSR

The IA32_RTIT_STATUS MSR is readable and writable by software, though some fields cannot be modified by software. See Table 35-7 for details. The WRMSR instruction ignores these bits in the source operand (attempts to modify these bits are ignored and do not cause WRMSR to fault).

This MSR can only be written when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP). The processor does not modify the value of this MSR while TraceEn is 0 (software can modify it with WRMSR).

Table 35-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
0	FilterEn	0	This bit is written by the processor, and indicates that tracing is allowed for the current IP, see Section 35.2.6.5. Writes are ignored.
1	ContextEn	0	The processor sets this bit to indicate that tracing is allowed for the current context. See Section 35.2.6.3. Writes are ignored.
2	TriggerEn	0	The processor sets this bit to indicate that tracing is enabled. See Section 35.2.6.2. Writes are ignored.
3	Reserved	0	Must be 0.
4	Error	0	The processor sets this bit to indicate that an operational error has been encountered. When this bit is set, TriggerEn is cleared to 0 and packet generation is disabled. For details, see “ToPA Errors” in Section 35.2.7.2. When TraceEn is cleared, software can write this bit. Once it is set, only software can clear it. It is not recommended that software ever set this bit, except in cases where it is restoring a prior saved state.

Table 35-7. IA32_RTIT_STATUS MSR

Position	Bit Name	At Reset	Bit Description
5	Stopped	0	The processor sets this bit to indicate that a ToPA Stop condition has been encountered. When this bit is set, TriggerEn is cleared to 0 and packet generation is disabled. For details, see “ToPA STOP” in Section 35.2.7.2. When TraceEn is cleared, software can write this bit. Once it is set, only software can clear it. It is not recommended that software ever set this bit, except in cases where it is restoring a prior saved state.
6	PendPSB	0	If IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, the processor sets this bit when the threshold for a PSB+ to be inserted has been reached. The processor will clear this bit when the PSB+ has been inserted into the trace. If PendPSB = 1 and InjectPsbPmiOnEnable = 1 when IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a PSB+ will be inserted into the trace. This field is reserved if CPUID.14H.00H:EBX[6] = 0.
7	PendTopaPMI	0	If IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1, the processor sets this bit when the threshold for a ToPA PMI to be inserted has been reached. Software should clear this bit once the ToPA PMI has been handled, see “ToPA PMI” for details. If PendTopaPMI = 1 and InjectPsbPmiOnEnable = 1 when IA32_RTIT_CTL.TraceEn[0] transitions from 0 to 1, a PMI will be pended. This field is reserved if CPUID.14H.00H:EBX[6] = 0.
31:8	Reserved	0	Must be 0.
48:32	PacketByteCnt	0	This field is written by the processor, and holds a count of packet bytes that have been sent out. The processor also uses this field to determine when the next PSB packet should be inserted. Note that the processor may clear or modify this field at any time while IA32_RTIT_CTL.TraceEn=1. It will have a stable value when IA32_RTIT_CTL.TraceEn=0. See Section 35.4.2.17 for details. This field is reserved when CPUID.14H.00H:EBX[1] (“Configurable PSB and CycleAccurate Mode Supported”) is 0.
63:49	Reserved	0	Must be 0.

35.2.8.5 IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B MSRs

The role of the IA32_RTIT_ADDRn_A/B register pairs, for each n, is determined by the corresponding ADDRn_CFG fields in IA32_RTIT_CTL (see Section 35.2.8.2). The number of these register pairs is enumerated by CPUID.14H.01H:EAX.RANGEcnt[2:0].

- Processors that enumerate support for 1 range support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
- Processors that enumerate support for 2 ranges support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
 - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
- Processors that enumerate support for 3 ranges support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
 - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
 - IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B
- Processors that enumerate support for 4 ranges support:
 - IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B
 - IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B
 - IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B

— IA32_RTIT_ADDR3_A, IA32_RTIT_ADDR3_B

Each register has a single 64-bit field that holds a linear address value. Writes must ensure that the address is in canonical form, otherwise a general-protection fault (#GP) fault will result.

Each MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

35.2.8.6 IA32_RTIT_CR3_MATCH MSR

The IA32_RTIT_CR3_MATCH register is compared against CR3 when IA32_RTIT_CTL.CR3Filter is 1. Bits 63:5 hold the CR3 address value to match, bits 4:0 are reserved to 0. For more details on CR3 filtering and the treatment of this register, see Section 35.2.5.2.

This MSR is accessible if CPUID.14H.00H:EBX[0], “CR3 Filtering Support”, is 1. This MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP). IA32_RTIT_CR3_MATCH[4:0] are reserved and must be 0; an attempt to set those bits using WRMSR causes a #GP.

35.2.8.7 IA32_RTIT_OUTPUT_BASE MSR

This MSR is used to configure the trace output destination, when output is directed to memory (IA32_RTIT_CTL.FabricEn = 0). The size of the address field is determined by [two values related to the processor’s physical-address width, MAXPHYADDR](#):

- [M](#) is the value enumerated in CPUID.80000008H:EAX[7:0] (at most 52).
- [MAXPHYADDR](#) is typically the same as [M](#). However, if IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), [MAXPHYADDR](#) is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.

See Table 35-8 for details.

WRMSR and XRSTORS will fault if any of bits 63:M would be set to 1; they allow setting bits M–1:MAXPHYADDR; however, if any of those bits are set outside SEAM, trace output may fail in a model-specific manner.

When the ToPA output scheme is used, the processor may update this MSR when packet generation is enabled, and those updates are asynchronous to instruction execution. Therefore, the values in this MSR should be considered unreliable unless packet generation is disabled (IA32_RTIT_CTL.TraceEn = 0).

Accesses to this MSR are supported only if Intel PT output to memory is supported, hence when either CPUID.14H.00H:ECX[0] or CPUID.14H.00H:ECX[2] are set. Otherwise WRMSR or RDMSR cause a general-protection fault (#GP). If supported, this MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

Table 35-8. IA32_RTIT_OUTPUT_BASE MSR

Position	Bit Name	At Reset	Bit Description
6:0	Reserved	0	Must be 0.
MAXPHYADDR-1:7	BasePhysAddr	0	The base physical address. How this address is used depends on the value of IA32_RTIT_CTL.ToPA: 0: This is the base physical address of a single, contiguous physical output region. This could be mapped to DRAM or to MMIO, depending on the value. The base address should be aligned with the size of the region, such that none of the 1s in the mask value(Section 35.2.8.8) overlap with 1s in the base address. If the base is not aligned, an operational error will result (see Section 35.3.10). 1: The base physical address of the current ToPA table. The address must be 4K aligned. Writing an address in which bits 11:7 are non-zero will not cause a #GP, but an operational error will be signaled once TraceEn is set. See “ToPA Errors” in Section 35.2.7.2, as well as Section 35.3.10.
M-1:MAXPHYADDR	Reserved	0	Should be 0 when processor trace is enabled.
63:M	Reserved	0	Must be 0.

35.2.8.8 IA32_RTIT_OUTPUT_MASK_PTRS MSR

This MSR holds any mask or pointer values needed to indicate where the next byte of trace output should be written. The meaning of the values held in this MSR depend on whether the ToPA output mechanism is in use. See Section 35.2.7.2 for details.

The processor updates this MSR while when packet generation is enabled, and those updates are asynchronous to instruction execution. Therefore, the values in this MSR should be considered unreliable unless packet generation is disabled (IA32_RTIT_CTL.TraceEn = 0).

Accesses to this MSR are supported only if Intel PT output to memory is supported, hence when either CPUID.14H.00H:ECX[0] or CPUID.14H.00H:ECX[2] are set. Otherwise WRMSR or RDMSR cause a general-protection fault (#GP). If supported, this MSR can be written only when IA32_RTIT_CTL.TraceEn is 0; otherwise WRMSR causes a general-protection fault (#GP).

Table 35-9. IA32_RTIT_OUTPUT_MASK_PTRS MSR

Position	Bit Name	At Reset	Bit Description
6:0	LowerMask	7FH	Forced to 1, writes are ignored.
31:7	MaskOrTableOffset	0	The use of this field depends on the value of IA32_RTIT_CTL.ToPA: 0: This field holds bits 31:7 of the mask value for the single, contiguous physical output region. The size of this field indicates that regions can be of size 128B up to 4GB. This value (combined with the lower 7 bits, which are reserved to 1) will be ANDed with the OutputOffset field to determine the next write address. All 1s in this field should be consecutive and starting at bit 7, otherwise the region will not be contiguous, and an operational error (Section 35.3.10) will be signaled when TraceEn is set. 1: This field holds bits 27:3 of the offset pointer into the current ToPA table. This value can be added to the IA32_RTIT_OUTPUT_BASE value to produce a pointer to the current ToPA table entry, which itself is a pointer to the current output region. In this scenario, the lower 7 reserved bits are ignored. This field supports tables up to 256 MBytes in size.
63:32	OutputOffset	0	The use of this field depends on the value of IA32_RTIT_CTL.ToPA: 0: This is bits 31:0 of the offset pointer into the single, contiguous physical output region. This value will be added to the IA32_RTIT_OUTPUT_BASE value to form the physical address at which the next byte of packet output data will be written. This value must be less than or equal to the MaskOrTableOffset field, otherwise an operational error (Section 35.3.10) will be signaled when TraceEn is set. 1: This field holds bits 31:0 of the offset pointer into the current ToPA output region. This value will be added to the output region base field, found in the current ToPA table entry, to form the physical address at which the next byte of trace output data will be written. This value must be less than the ToPA entry size, otherwise an operational error (Section 35.3.10) will be signaled when TraceEn is set.

35.2.9 Interaction of Intel® Processor Trace and Other Processor Features

35.2.9.1 Intel® Transactional Synchronization Extensions (Intel® TSX)

The operation of Intel TSX is described in Chapter 14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1. For tracing purpose, packet generation does not distinguish between hardware lock elision (HLE) and restricted transactional memory (RTM), but speculative execution does have impacts on the trace output. Specifically, packets are generated as instructions complete, even for instructions in a transactional region that is later aborted. For this reason, debugging software will need indication of the beginning and end of a transactional region; this will allow software to understand when instructions are part of a transactional region and whether that region has been committed.

To enable this, TSX information is included in a MODE packet leaf. The mode bits in the leaf are:

- **InTX:** Set to 1 on an TSX transaction begin, and cleared on transaction commit or abort.
- **TXAbort:** Set to 1 only when InTX transitions from 1 to 0 on an abort. Cleared otherwise.

If BranchEn=1, this MODE packet will be sent each time the transaction status changes. See Table 35-10 for details.

Table 35-10. TSX Packet Scenarios with BranchEn=1

TSX Event	Instruction	Packets
Transaction Begin	Either XBEGIN or XACQUIRE lock (the latter if executed transactionally)	MODE(TXAbort=0, InTX=1), FUP(CurrentIP)
Transaction Commit	Either XEND or XRELEASE lock, if transactional execution ends. This happens only on the outermost commit	MODE(TXAbort=0, InTX=0), FUP(CurrentIP)

Table 35-10. TSX Packet Scenarios with BranchEn=1

TSX Event	Instruction	Packets
Transaction Abort	XABORT or other transactional abort	MODE(TXAbort=1, InTX=0), FUP(CurrentIP), TIP(TargetIP)
Other	One of the following: ▪ Nested XBEGIN or XACQUIRE lock ▪ An outer XACQUIRE lock that doesn't begin a transaction (InTX not set) ▪ Non-outermost XEND or XRELEASE lock	None. No change to TSX mode bits for these cases.

The CurrentIP listed above is the IP of the associated instruction. The TargetIP is the IP of the next instruction to be executed; for HLE, this is the XACQUIRE lock; for RTM, this is the fallback handler.

Intel PT stores are non-transactional, and thus packet writes are not rolled back on TSX abort.

35.2.9.2 TSX and IP Filtering

A complication with tracking transactions is handling transactions that start or end outside of the tracing region. Transactions can't span across a change in ContextEn, because CPL changes and CR3 changes each cause aborts. But a transaction can start within the IP filter region and end outside it.

To assist the decoder handling this situation, MODE.TSX packets can be sent even if FilterEn=0, though there will be no FUP attached. Instead, they will merely serve to indicate to the decoder when transactions are active and when they are not. When tracing resumes (due to PacketEn=1), the last MODE.TSX preceding the TIP.PGE will indicate the current transaction status.

35.2.9.3 System Management Mode (SMM)

SMM code has special privileges that non-SMM code does not have. Intel Processor Trace can be used to trace SMM code, but special care is taken to ensure that SMM handler context is not exposed in any non-SMM trace collection. Additionally, packet output from tracing non-SMM code cannot be written into memory space that is either protected by SMRR or used by the SMM handler.

SMM is entered via a system management interrupt (SMI). SMI delivery saves the value of IA32_RTIT_CTL.TraceEn into SMRAM and then clears it, thereby disabling packet generation.

The saving and clearing of IA32_RTIT_CTL.TraceEn ensures two things:

1. All internally buffered packet data is flushed before entering SMM (see Section 35.2.8.2).
2. Packet generation ceases before entering SMM, so any tracing that was configured outside SMM does not continue into SMM. No SMM instruction pointers or other state will be exposed in the non-SMM trace.

When the RSM instruction is executed to return from SMM, the TraceEn value that was saved by SMI delivery is restored, allowing tracing to be resumed. As is done any time packet generation is enabled, ContextEn is re-evaluated, based on the values of CPL, CR3, etc., established by RSM.

Like other interrupts, delivery of an SMI produces a FUP containing the IP of the next instruction to execute. By toggling TraceEn, SMI and RSM can produce TIP.PGD and TIP.PGE packets, respectively, indicating that tracing was disabled or re-enabled. See Table 35.7 for more information about packets entering and leaving SMM.

Although #SMI and RSM change CR3, PIP packets are not generated in these cases. With #SMI tracing is disabled before the CR3 change; with RSM TraceEn is restored after CR3 is written.

TraceEn must be cleared before executing RSM, otherwise it will cause a shutdown. Further, on processors that restrict use of Intel PT with LBRs (see Section 35.3.1.2), any RSM that results in enabling of both will cause a shutdown.

Intel PT can support tracing of System Transfer Monitor operating in SMM, see Section 35.6.

35.2.9.4 Virtual-Machine Extensions (VMX)

Initial implementations of Intel Processor Trace do not support tracing in VMX operation. Such processors indicate this by returning 0 for IA32_VMX_MISC[bit 14]. On these processors, execution of the VMXON instruction clears IA32_RTIT_CTL.TraceEn and any attempt to write IA32_RTIT_CTL in VMX operation causes a general-protection exception (#GP).

Processors that support Intel Processor Trace in VMX operation return 1 for IA32_VMX_MISC[bit 14]. Details of tracing in VMX operation are described in Section 35.4.2.26.

35.2.9.5 Intel® Software Guard Extensions (Intel® SGX)

Intel SGX provides an application with the ability to instantiate a protective container (an enclave) with confidentiality and integrity (see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D). On a processor with both Intel PT and Intel SGX enabled, when executing code within a production enclave, no control flow packets are produced by Intel PT. An enclave entry will clear ContextEn, thereby blocking control flow packet generation. A TIP.PGD packet will be generated if PacketEn=1 at the time of the entry.

Upon enclave exit, ContextEn will no longer be forced to 0. If other enables are set at the time, a TIP.PGE may be generated to indicate that tracing is resumed.

During the enclave execution, Intel PT remains enabled, and periodic or timing packets such as PSB, TSC, MTC, or CBR can still be generated. No IPs or other architectural state will be exposed.

For packet generation examples on enclave entry or exit, see Section 35.7.

Debug Enclaves

Intel SGX allows an enclave to be configured with relaxed protection of confidentiality for debug purposes, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D. In a debug enclave, Intel PT continues to function normally. Specifically, ContextEn is not impacted by an enclave entry or exit. Hence, the generation of ContextEn-dependent packets within a debug enclave is allowed.

35.2.9.6 SENTER/ENTERACCS and ACM

GETSEC[SENDER] and GETSEC[ENTERACCS] instructions clear TraceEn, and it is not restored when those instructions complete. SENTER also causes TraceEn to be cleared on other logical processors when they rendezvous and enter the SENTER sleep state. In these two cases, the disabling of packet generation is not guaranteed to flush internally buffered packets. Some packets may be dropped.

When executing an authenticated code module (ACM), packet generation is silently disabled during ACRAM setup. TraceEn will be cleared, but no TIP.PGD packet is generated. After completion of the module, the TraceEn value will be restored. There will be no TIP.PGE packet, but timing packets, like TSC and CBR, may be produced.

35.2.9.7 Intel® Memory Protection Extensions (Intel® MPX)

Bounds exceptions (#BR) caused by Intel MPX are treated like other exceptions, producing FUP and TIP packets that indicate the source and destination IPs.

35.3 CONFIGURATION AND PROGRAMMING GUIDELINE

35.3.1 Detection of Intel Processor Trace and Capability Enumeration

Processor support for Intel Processor Trace is indicated by CPUID.07H.00H:EBX[25]= 1. CPUID function 14H is dedicated to enumerate the resource and capability of processors that report CPUID.07H.00H:EBX[25]= 1. Different processor generations may have architecturally-defined variation in capabilities. Table 35-11 describes details of the enumerable capabilities that software must use across generations of processors that support Intel Processor Trace.

Table 35-11. CPUID.14H Enumeration of Intel Processor Trace Capabilities

CPUID.14H.00H		Name	Description Behavior
Register	Bits		
EAX	31:0	Maximum valid sub-leaf Index	Specifies the index of the maximum valid sub-leaf for this CPUID leaf.
EBX	0	CR3 Filtering Support	1: Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. See Section 35.2.8. 0: Indicates that writes that set IA32_RTIT_CTL.CR3Filter to 1, or any access to IA32_RTIT_CR3_MATCH, will generate a #GP exception.
	1	Configurable PSB and Cycle-Accurate Mode Supported	1: (a) IA32_RTIT_CTL.PSBFreq can be set to a non-zero value, in order to select the preferred PSB frequency (see below for allowed values). (b) IA32_RTIT_STATUS.PacketByteCnt can be set to a non-zero value, and will be incremented by the processor when tracing to indicate progress towards the next PSB. If trace packet generation is enabled by setting TraceEn, a PSB will only be generated if PacketByteCnt=0. (c) IA32_RTIT_CTL.CYCEn can be set to 1 to enable Cycle-Accurate Mode. See Section 35.2.8. 0: (a) Any attempt to write a non-zero value to IA32_RTIT_CTL.PSBFreq or IA32_RTIT_STATUS.PacketByteCnt will generate a #GP exception. (b) If trace packet generation is enabled by setting TraceEn, a PSB is always generated. (c) Any attempt to write a non-zero value to IA32_RTIT_CTL.CYCEn or IA32_RTIT_CTL.CycThresh will generate a #GP exception.
	2	IP Filtering and TraceStop supported, and Preserve Intel PT MSRs across warm reset	1: (a) IA32_RTIT_CTL provides at one or more ADDRn_CFG field to configure the corresponding address range MSRs for IP Filtering or IP TraceStop. Each ADDRn_CFG field accepts a value in the range of 0:2 inclusive. The number of ADDRn_CFG fields is reported by CPUID.14H.01H:EAX.RANGE CNT[2:0]. (b) At least one register pair IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B are provided to configure address ranges for IP filtering or IP TraceStop. (c) On warm reset, all Intel PT MSRs will retain their pre-reset values, though IA32_RTIT_CTL.TraceEn will be cleared. The Intel PT MSRs are listed in Section 35.2.8. 0: (a) An Attempt to write IA32_RTIT_CTL.ADDRn_CFG with non-zero encoding values will cause #GP. (b) Any access to IA32_RTIT_ADDRn_A and IA32_RTIT_ADDRn_B, will generate a #GP exception. (c) On warm reset, all Intel PT MSRs will be cleared.
	3	MTC Supported	1: IA32_RTIT_CTL.MTCEn can be set to 1, and MTC packets will be generated. See Section 35.2.8. 0: An attempt to set IA32_RTIT_CTL.MTCEn or IA32_RTIT_CTL.MTCFreq to a non-zero value will generate a #GP exception.
	4	PTWRITE Supported	1: Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets. 0: Writes that set IA32_RTIT_CTL[12] or IA32_RTIT_CTL[5] will generate a #GP exception, and PTWRITE will #UD fault.
	5	Power Event Trace Supported	1: Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. 0: Writes that set IA32_RTIT_CTL[4] will generate a #GP exception.

Table 35-11. CPUID.14H Enumeration of Intel Processor Trace Capabilities (Contd.)

CPUID.14H.00H		Name	Description Behavior
Register	Bits		
	6	PSB and PMI Preservation Supported	1: Writes can set IA32_RTIT_CTL[56] (InjectPsbPmiOnEnable), enabling the processor to set IA32_RTIT_STATUS[7] (PendToPaPMI) and/or IA32_RTIT_STATUS[6] (PendPSB) in order to preserve ToPA PMIs and/or PSBs otherwise lost due to Intel PT disable. Writes can also set PendToPAPMI and PendPSB. 0: Writes that set IA32_RTIT_CTL[56], IA32_RTIT_STATUS[7], or IA32_RTIT_STATUS[6] will generate a #GP exception.
	7	Event Trace Supported	1: Writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation. 0: Writes that set IA32_RTIT_CTL[31] will generate a #GP exception.
	8	TNT Disable Supported	1: Writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation. 0: Writes that set IA32_RTIT_CTL[55] will generate a #GP exception.
	31:9	Reserved	
ECX	0	ToPA Output Supported	1: Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme (Section 35.2.7.2) IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. 0: Unless CPUID.14H.00H:ECX.SNGL_RNG_OUT[2] = 1, writes to IA32_RTIT_OUTPUT_BASE or IA32_RTIT_OUTPUT_MASK_PTRS MSRs will generate a #GP exception.
	1	ToPA Tables Allow Multiple Output Entries	1: ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOffsetTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. 0: ToPA tables can hold only one output entry, which must be followed by an END=1 entry which points back to the base of the table. Further, ToPA PMIs will be delivered before the region is filled. See ToPA PMI in Section 35.2.7.2. If there is more than one output entry before the END entry, or if the END entry has the wrong base address, an operational error will be signaled (see “ToPA Errors” in Section 35.2.7.2).
	2	Single-Range Output Supported	1: Enabling tracing (TraceEn=1) with IA32_RTIT_CTL.ToPA=0 is supported. 0: Unless CPUID.14H.00H:ECX.TOPAOUT[0] = 1, writes to IA32_RTIT_OUTPUT_BASE or IA32_RTIT_OUTPUT_MASK_PTRS MSRs will generate a #GP exception.
	3	Output to Trace Transport Subsystem Supported	1: Setting IA32_RTIT_CTL.FabricEn to 1 is supported. 0: IA32_RTIT_CTL.FabricEn is reserved. Write 1 to IA32_RTIT_CTL.FabricEn will generate a #GP exception.
	30:4	Reserved	
	31	IP Payloads are LIP	1: Generated packets which contain IP payloads have LIP values, which include the CS base component. 0: Generated packets which contain IP payloads have RIP values, which are the offset from CS base.
EDX	31:0	Reserved	

If CPUID.14H.00H:EAX reports a non-zero value, additional capabilities of Intel Processor Trace are described in the sub-leaves of CPUID.14H.

Table 35-12. CPUID.14H.01H Enumeration of Intel Processor Trace Capabilities

CPUID.14H.01H		Name	Description Behavior
Register	Bits		
EAX	2:0	Number of Address Ranges	A non-zero value specifies the number ADDRn_CFG field supported in IA32_RTIT_CTL and the number of register pair IA32_RTIT_ADDRn_A/IA32_RTIT_ADDRn_B supported for IP filtering and IP TraceStop. NOTE: Currently, no processors support more than 4 address ranges.
	15:3	Reserved	
	31:16	Bitmap of supported MTC Period Encodings	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.MTCFreq field. This applies only if CPUID.14H.00H:EBX[3] = 1 (MTC Packet generation is supported), otherwise the MTCFreq field is reserved to 0. Each bit position in this field represents 1 encoding value in the 4-bit MTCFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: MTCFreq can be assigned the associated encoding value. 0: MTCFreq cannot be assigned to the associated encoding value. A write to IA32_RTIT_CTL.MTCFreq with unsupported encoding will cause #GP fault.
EBX	15:0	Bitmap of supported Cycle Threshold values	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.CycThresh field. This applies only if CPUID.14H.00H:EBX[1] = 1 (Cycle-Accurate Mode is Supported), otherwise the CycThresh field is reserved to 0. See Section 35.2.8. Each bit position in this field represents 1 encoding value in the 4-bit CycThresh field (ie, bit 0 is associated with encoding value 0). For each bit: 1: CycThresh can be assigned the associated encoding value. 0: CycThresh cannot be assigned to the associated encoding value. A write to CycThresh with unsupported encoding will cause #GP fault.
	31:16	Bitmap of supported Configurable PSB Frequency encoding	The non-zero bits indicate the map of supported encoding values for the IA32_RTIT_CTL.PSBFreq field. This applies only if CPUID.14H.00H:EBX[1] = 1 (Configurable PSB is supported), otherwise the PSBFreq field is reserved to 0. See Section 35.2.8. Each bit position in this field represents 1 encoding value in the 4-bit PSBFreq field (ie, bit 0 is associated with encoding value 0). For each bit: 1: PSBFreq can be assigned the associated encoding value. 0: PSBFreq cannot be assigned to the associated encoding value. A write to PSBFreq with unsupported encoding will cause #GP fault.
ECX	31:0	Reserved	
EDX	31:0	Reserved	

35.3.1.1 Packet Decoding of RIP versus LIP

FUP, TIP, TIP.PGE, and TIP.PGE packets can contain an instruction pointer (IP) payload. On some processor generations, this payload will be an effective address (RIP), while on others this will be a linear address (LIP). In the former case, the payload is the offset from the current CS base address, while in the latter it is the sum of the offset and the CS base address (Note that in real mode, the CS base address is the value of CS<4, while in protected

mode the CS base address is the base linear address of the segment indicated by the CS register.). Which IP type is in use is indicated by enumeration (see CPUID.14H.00H:ECX.LIP[31] in Table 35-11).

For software that executes while the CS base address is 0 (including all software executing in 64-bit mode), the difference is indistinguishable. A trace decoder must account for cases where the CS base address is not 0 and the resolved LIP will not be evident in a trace generated on a CPU that enumerates use of RIP. This is likely to cause problems when attempting to link the trace with the associated binaries.

Note that IP comparison logic, for IP filtering and TraceStop range calculation, is based on the same IP type as these IP packets. For processors that output RIP, the IP comparison mechanism is also based on RIP, and hence on those processors RIP values should be written to IA32_RTIT_ADDRn_[AB] MSRs. This can produce differing behavior if the same trace configuration setting is run on processors reporting different IP types, i.e., CPUID.14H.00H:ECX.LIP[31]. Care should be taken to check CPUID when configuring IP filters.

35.3.1.2 Model Specific Capability Restrictions

Some processor generations impose restrictions that prevent use of LBRs/BTS/BTM/LERs when software has enabled tracing with Intel Processor Trace. On these processors, when TraceEn is set, updates of LBR, BTS, BTM, LERs are suspended but the states of the corresponding IA32_DEBUGCTL control fields remained unchanged as if it were still enabled. When TraceEn is cleared, the LBR array is reset, and LBR/BTS/BTM/LERs updates will resume. Further, reads of these registers will return 0, and writes will be dropped.

The list of MSRs whose updates/accesses are restricted follows.

- MSR_LASTBRANCH_x_TO_IP, MSR_LASTBRANCH_x_FROM_IP, MSR_LBR_INFO_x, MSR_LASTBRANCH_TOS
- MSR_LER_FROM_LIP, MSR_LER_TO_LIP
- MSR_LBR_SELECT

For processors with CPUID DisplayFamily_DisplayModel signatures of 06_3DH, 06_47H, 06_4EH, 06_4FH, 06_56H, and 06_5EH, the use of Intel PT and LBRs are mutually exclusive.

35.3.2 Enabling and Configuration of Trace Packet Generation

To configure trace packets, enable packet generation, and capture packets, software starts with using CPUID instruction to detect its feature flag, CPUID.07H.00H:EBX[25] = 1; followed by enumerating the capabilities described in Section 35.3.1.

Based on the capability queried from Section 35.3.1, software must configure a number of model-specific registers. This section describes programming considerations related to those MSRs.

35.3.2.1 Enabling Packet Generation

When configuring and enabling packet generation, the IA32_RTIT_CTL MSR should be written after any other Intel PT MSRs have been written, since writes to the other configuration MSRs cause a general-protection fault (#GP) if TraceEn = 1. If a prior trace collection context is not being restored, then software should first clear IA32_RTIT_STATUS. This is important since the Stopped, and Error fields are writable; clearing the MSR clears any values that may have persisted from prior trace packet collection contexts. See Section 35.2.8.2 for details of packets generated by setting TraceEn to 1.

If setting TraceEn to 1 causes an operational error (see Section 35.3.10), there may be a delay after the WRMSR completes before the error is signaled in the IA32_RTIT_STATUS MSR.

While packet generation is enabled, the values of some configuration MSRs (e.g., IA32_RTIT_STATUS and IA32_RTIT_OUTPUT_*) are transient, and reads may return values that are out of date. Only after packet generation is disabled (by clearing TraceEn) do reads of these MSRs return reliable values.

35.3.2.2 Disabling Packet Generation

After disabling packet generation by clearing IA32_RTIT_CTL, it is advisable to read the IA32_RTIT_STATUS MSR (Section 35.2.8.4):

- If the Error bit is set, an operational error was encountered, and the trace is most likely compromised. Software should check the source of the error (by examining the output MSR values), correct the source of the problem, and then attempt to gather the trace again. For details on operational errors, see Section 35.3.10. Software should clear IA32_RTIT_STATUS.Error before re-enabling packet generation.
- If the Stopped bit is set, software execution encountered an IP TraceStop (see Section 35.2.5.3) or the ToPA Stop condition (see “ToPA STOP” in Section 35.2.7.2) before packet generation was disabled.

35.3.3 Flushing Trace Output

Packets are first buffered internally and then written out asynchronously. To collect packet output for post-processing, a collector needs first to ensure that all packet data has been flushed from internal buffers. Software can ensure this by stopping packet generation by clearing IA32_RTIT_CTL.TraceEn (see “Disabling Packet Generation” in Section 35.2.8.2).

When software clears IA32_RTIT_CTL.TraceEn to flush out internally buffered packets, the logical processor issues an SFENCE operation which ensures that WC trace output stores will be ordered with respect to the next store, or serializing operation. A subsequent read from the same logical processor will see the flushed trace data, while a read from another logical processor should be preceded by a store, fence, or architecturally serializing operation on the tracing logical processor.

When the flush operations complete, the IA32_RTIT_OUTPUT_* MSR values indicate where the trace ended. While TraceEn is set, these MSRs may hold stale values. Further, if a ToPA region with INT=1 is filled, meaning a ToPA PMI has been triggered, IA32_PERF_GLOBAL_STATUS.Trace_ToPA_PMI[55] will be set by the time the flush completes.

35.3.4 Warm Reset

The MSRs software uses to program Intel Processor Trace are cleared after a power-on RESET (or cold RESET). On a warm RESET, the contents of those MSRs can retain their values from before the warm RESET with the exception that IA32_RTIT_CTL.TraceEn will be cleared (which may have the side effect of clearing some bits in IA32_RTIT_STATUS).

35.3.5 Context Switch Consideration

To facilitate construction of instruction execution traces at the granularity of a software process or thread context, software can save and restore the states of the trace configuration MSRs across the process or thread context switch boundary. The principle is the same as saving and restoring the typical architectural processor states across context switches.

35.3.5.1 Manual Trace Configuration Context Switch

The configuration can be saved and restored through a sequence of instructions of RDMSR, management of MSR content and WRMSR. To stop tracing and to ensure that all configuration MSRs contain stable values, software must clear IA32_RTIT_CTL.TraceEn before reading any other trace configuration MSRs. The recommended method for saving trace configuration context manually follows:

1. RDMSR IA32_RTIT_CTL, save value to memory
2. WRMSR IA32_RTIT_CTL with saved value from RDMSR above and TraceEn cleared
3. RDMSR all other configuration MSRs whose values had changed from previous saved value, save changed values to memory

When restoring the trace configuration context, IA32_RTIT_CTL should be restored last:

1. Read saved configuration MSR values, aside from IA32_RTIT_CTL, from memory, and restore them with WRMSR
2. Read saved IA32_RTIT_CTL value from memory, and restore with WRMSR.

35.3.5.2 Trace Configuration Context Switch Using XSAVES/XRSTORS

On processors whose XSAVE feature set supports XSAVES and XRSTORS, the Trace configuration state can be saved using XSAVES and restored by XRSTORS, in conjunction with the bit field associated with supervisory state component in IA32_XSS. See Chapter 13, “Managing State Using the XSAVE Feature Set,” of Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1.

35.3.6 Cycle-Accurate Mode

Intel PT can be run in a cycle-accurate mode which enables CYC packets (see Section 35.4.2.14) that provide low-level information in the processor core clock domain. This cycle counter data in CYC packets can be used to compute IPC (Instructions Per Cycle), or to track wall-clock time on a fine-grain level.

To enable cycle-accurate mode packet generation, software should set IA32_RTIT_CTL.CYCEn=1. It is recommended that software also set TSCEn=1 anytime cycle-accurate mode is in use. With this, all CYC-eligible packets will be preceded by a CYC packet, the payload of which indicates the number of core clock cycles since the last CYC packet. In cases where multiple CYC-eligible packets are generated in a single cycle, only a single CYC will be generated before the CYC-eligible packets, otherwise each CYC-eligible packet will be preceded by its own CYC. The CYC-eligible packets are:

- TNT, TIP, TIP.PGE, TIP.PGD, MODE.EXEC, MODE.TSX, PIP, VMCS, OVF, MTC, TSC, PTWRITE, EXSTOP

TSC packets are generated when there is insufficient information to reconstruct wall-clock time, due to tracing being disabled (TriggerEn=0), or power down scenarios like a transition to a deep-sleep MWAIT C-state. In this case, the CYC that is generated along with the TSC will indicate the number of cycles actively tracing (those powered up, with TriggerEn=1) executed between the last CYC packet and the TSC packet. And hence the amount of time spent while tracing is inactive can be inferred from the difference in time between that expected based on the CYC value, and the actual time indicated by the TSC.

Additional CYC packets may be sent stand-alone, so that the processor can ensure that the decoder is aware of the number of cycles that have passed before the internal hardware counter wraps, or is reset due to other micro-architectural condition. There is no guarantee at what intervals these standalone CYC packets will be sent, except that they will be sent before the wrap occurs. An illustration is given below.

Example 35-1. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Generated Packets	Comment
x	call %eax	CYC(?), TIP	?Elapsed cycles from the previous CYC unknown
x + 2	call %ebx	CYC(2), TIP	1 byte CYC packet; 2 cycles elapsed from the previous CYC
x + 8	jnz Foo (not taken)	CYC(6)	1 byte CYC packet
x + 9	ret (compressed)		
x + 12	jnz Bar (taken)		
x + 16	ret (uncompressed)	TNT, CYC(8), TIP	1 byte CYC packet
x + 4111		CYC(4095)	2 byte CYC packet
x + 12305		CYC(8194)	3 byte CYC packet
x + 16332	mov cr3, %ebx	CYC(4027), PIP	2 byte CYC packet

35.3.6.1 Cycle Counter

The cycle counter is implemented in hardware (independent of the time stamp counter or performance monitoring counters), and is a simple incrementing counter that does not saturate, but rather wraps. The size of the counter is implementation specific.

The cycle counter is reset to zero any time that TriggerEn is cleared, and when a CYC packet is sent. The cycle counter will continue to count when ContextEn or FilterEn are cleared, and cycle packets will still be generated. It will not count during sleep states that result in Intel PT logic being powered-down, but will count up to the point where clocks are disabled, and resume counting once they are re-enabled.

35.3.6.2 Cycle Packet Semantics

Cycle-accurate mode adheres to the following protocol:

- All packets that precede a CYC packet represent instructions or events that took place before the CYC time.
- All packets that follow a CYC packet represent instructions or events that took place at the same time as, or after, the CYC time.
- The CYC-eligible packet that immediately follows a CYC packet represents an instruction or event that took place at the same time as the CYC time.

These items above give the decoder a means to apply CYC packets to a specific instruction in the assembly stream. Most packets represent a single instruction or event, and hence the CYC packet that precedes each of those packets represents the retirement time of that instruction or event. In the case of TNT packets, up to 6 conditional branches and/or compressed RETs may be contained in the packet. In this case, the preceding CYC packet provides the retirement time of the first branch in the packet. It is possible that multiple branches retired in the same cycle as that first branch in the TNT, but the protocol will not make that obvious. Also note that a MTC packet could be generated in the same cycle as the first JCC in the TNT packet. In this case, the CYC would precede both the MTC and the TNT, and apply to both.

Note that there are times when the cycle counter will stop counting, though cycle-accurate mode is enabled. After any such scenario, a CYC packet followed by TSC packet will be sent. See Section 35.8.3.2 to understand how to interpret the payload values

Multi-packet Instructions or Events

Some operations, such as interrupts or task switches, generate multiple packets. In these cases, multiple CYC packets may be sent for the operation, preceding each CYC-eligible packet in the operation. An example, using a task switch on a software interrupt, is shown below.

Example 35-2. An Example of CYC in the Presence of Multi-Packet Operations

Time (cycles)	Instruction Snapshot	Generated Packets
x	jnz Foo (not taken)	CYC(?)
x + 2	ret (compressed)	
x + 8	jnz Bar (taken)	
x + 9	jmp %eax	TNT, CYC(9), TIP
x + 12	jnz Bar (not taken)	CYC(3)
x + 32	int3 (task gate)	TNT, FUP, CYC(10), PIP, CYC(20), MODE.Exec, TIP

35.3.6.3 Cycle Thresholds

Software can opt to reduce the frequency of cycle packets, a trade-off to save bandwidth and intrusion at the expense of precision. This is done by utilizing a cycle threshold (see Section 35.2.8.2).

IA32_RTIT_CTL.CycThresh indicates to the processor the minimum number of cycles that must pass before the next CYC packet should be sent. If this value is 0, no threshold is used, and CYC packets can be sent every cycle in which a CYC-eligible packet is generated. If this value is greater than 0, the hardware will wait until the associated

number of cycles have passed since the last CYC packet before sending another. CPUID provides the threshold options for CycThresh, see Section 35.3.1.

Note that the cycle threshold does not dictate how frequently a CYC packet will be posted, it merely assigns the maximum frequency. If the cycle threshold is 16, a CYC packet can be posted no more frequently than every 16 cycles. However, once that threshold of 16 cycles has passed, it still requires a new CYC-eligible packet to be generated before a CYC will be inserted. Table 35-13 illustrates the threshold behavior.

Table 35-13. An Illustrative CYC Packet Example

Time (cycles)	Instruction Snapshot	Threshold			
		0	16	32	64
x	jmp %eax	CYC, TIP	CYC, TIP	CYC, TIP	CYC, TIP
x + 9	call %ebx	CYC, TIP	TIP	TIP	TIP
x + 15	call %ecx	CYC, TIP	TIP	TIP	TIP
x + 30	jmp %edx	CYC, TIP	CYC, TIP	TIP	TIP
x + 38	mov cr3, %eax	CYC, PIP	PIP	CYC, PIP	PIP
x + 46	jmp [%eax]	CYC, TIP	CYC, TIP	TIP	TIP
x + 64	call %edx	CYC, TIP	CYC, TIP	TIP	CYC, TIP
x + 71	jmp %edx	CYC, TIP	TIP	CYC, TIP	TIP

35.3.7 Decoder Synchronization (PSB+)

The PSB packet (Section 35.4.2.17) serves as a synchronization point for a trace-packet decoder. It is a pattern in the trace log for which the decoder can quickly scan to align packet boundaries. No legal packet combination can result in such a byte sequence. As such, it serves as the starting point for packet decode. To decode a trace log properly, the decoder needs more than simply to be aligned: it needs to know some state and potentially some timing information as well. The decoder should never need to retain any information (e.g., LastIP, call stack, compound packet event) across a PSB; all compound packet events will be completed before a PSB, and any compression state will be reset.

When a PSB packet is generated, it is followed by a PSBEND packet (Section 35.4.2.18). One or more packets may be generated in between those two packets, and these inform the decoder of the current state of the processor. These packets, known collectively as PSB+, should be interpreted as “status only”, since they do not imply any change of state at the time of the PSB, nor are they associated directly with any instruction or event. Thus, the normal binding and ordering rules that apply to these packets outside of PSB+ can be ignored when these packets are between a PSB and PSBEND. They inform the decoder of the state of the processor at the time of the PSB.

PSB+ can include:

- Timestamp (TSC), if IA32_RTIT_CTL.TSCEn=1.
- Timestamp-MTC Align (TMA), if IA32_RTIT_CTL.TSCEn=1 && IA32_RTIT_CTL.MTCEn=1.
- Paging Information Packet (PIP), if ContextEn=1 and IA32_RTIT_CTL.OS=1. The non-root bit (NR) is set if the logical processor is in VMX non-root operation and the “conceal VMX from PT” VM-execution control is 0.
- VMCS packet, if either the logical is in VMX root operation or the logical processor is in VMX non-root operation and the “conceal VMX from PT” VM-execution control is 0.
- Core Bus Ratio (CBR).
- MODE.TSX, if ContextEn=1 and BranchEn = 1.
- MODE.Exec, if PacketEn=1 or (ContextEn=1 and IA32_RTIT_CTL.EventEn=1).
- Flow Update Packet (FUP), if PacketEn=1.

PSB is generated only when TriggerEn=1; hence PSB+ has the same dependencies. The ordering of packets within PSB+ is not fixed. Timing packets such as CYC and MTC may be generated between PSB and PSBEND, and their meanings are the same as outside PSB+.

A PSB+ can be lost in some scenarios. If IA32_RTIT_STATUS.TriggerEn is cleared just as the PSB threshold is reached, e.g., due to TraceEn being cleared, the PSB+ may not be generated. On processors that support PSB preservation (CPUID.14H.00H:EBX[6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PSB+ that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendPSB[6] = 1. A PSB will be inserted, and PendPSB cleared, when PT is later re-enabled while PendPSB = 1.

Note that an overflow can occur during PSB+, and this could cause the PSBEND packet to be lost. For this reason, the OVF packet should also be viewed as terminating PSB+. If IA32_RTIT_STATUS.TriggerEn is cleared just as the PSB threshold is reached, the PSB+ may not be generated. TriggerEn can be cleared by a WRMSR that clears IA32_RTIT_CTL.TraceEn, a VM exit that clears IA32_RTIT_CTL.TraceEn, an #SMI, or any time that either IA32_RTIT_STATUS.Stopped is set (e.g., by a TraceStop or ToPA stop condition) or IA32_RTIT_STATUS.Error is set (e.g., by an Intel PT output error). On processors that support PSB preservation (CPUID.14H.00H:EBX[6] = 1), setting IA32_RTIT_CTL.InjectPsbPmiOnEnable[56] = 1 will ensure that a PSB+ that is pending at the time PT is disabled will be recorded by setting IA32_RTIT_STATUS.PendPSB[6] = 1. A PSB will then be pended when the saved PT context is later restored.

35.3.8 Internal Buffer Overflow

In the rare circumstances when new packets need to be generated but the processor's dedicated internal buffers are all full, an "internal buffer overflow" occurs. On such an overflow packet generation ceases (as packets would need to enter the processor's internal buffer) until the overflow resolves. Once resolved, packet generation resumes.

When the buffer overflow is cleared, an OVF packet (Section 35.4.2.16) is generated, and the processor ensures that packets which follow the OVF are not compressed (IP compression or RET compression) against packets that were lost.

If IA32_RTIT_CTL.BranchEn = 1, the OVF packet will be followed by a FUP if the overflow resolves while PacketEn=1. If the overflow resolves while PacketEn = 0 no packet is generated, but a TIP.PGE will naturally be generated later, once PacketEn = 1. The payload of the FUP or TIP.PGE will be the Current IP of the first instruction upon which tracing resumes after the overflow is cleared. If the overflow resolves while PacketEn=1, only timing packets may come between the OVF and the FUP. If the overflow resolves while PacketEn=0, any other packets that are not dependent on PacketEn may come between the OVF and the TIP.PGE.

35.3.8.1 Overflow Impact on Enables

The address comparisons to ADDRn ranges, for IP filtering and TraceStop (Section 35.2.5.3), continue during a buffer overflow, and TriggerEn, ContextEn, and FilterEn may change during a buffer overflow. Like other packets, however, any TIP.PGE or TIP.PGD packets that would have been generated will be lost. Further, IA32_RTIT_STATUS.PacketByteCnt will not increment, since it is only incremented when packets are generated.

If a TraceStop event occurs during the buffer overflow, IA32_RTIT_STATUS.Stopped will still be set, tracing will cease as a result. However, the TraceStop packet, and any TIP.PGD that result from the TraceStop, may be dropped.

35.3.8.2 Overflow Impact on Timing Packets

Any timing packets that are generated during a buffer overflow will be dropped. If only a few MTC packets are dropped, a decoder should be able to detect this by noticing that the time value in the first MTC packet after the buffer overflow incremented by more than one. If the buffer overflow lasted long enough that 256 MTC packets are lost (and thus the MTC packet 'wraps' its 8-bit CTC value), then the decoder may be unable to properly understand the trace. This is not an expected scenario. No CYC packets are generated during overflow, even if the cycle counter wraps.

Note that, if cycle-accurate mode is enabled, the OVF packet will generate a CYC packet. Because the cycle counter counts during overflows, this CYC packet can provide the duration of the overflow. However, there is a risk that the cycle counter wrapped during the overflow, which could render this CYC misleading.

35.3.9 TNT Disable

Software can opt to omit TNT packets from control flow trace (BranchEn=1) by setting IA32_RTIT_CTL.DisTNT[bit 55]. This can dramatically reduce trace size. Results will vary by workload, but trace size reductions of 40-75% are typical, which will have a corresponding reduction in performance overhead and memory bandwidth consumption from Intel PT. However, omitting TNT packets means the decoder is not able to follow the full control flow trace, since conditional branch and compressed RET results won't be known. Thus, TNT Disable should be employed only for usages that do not depend on full control flow trace.

NOTE

To avoid loss of RET results with TNT Disable, software may wish to disable RET compression by setting IA32_RTIT_CTL.DisRETC[bit 11].

35.3.10 Operational Errors

Errors are detected as a result of packet output configuration problems, which can include output alignment issues, ToPA reserved bit violations, or overlapping packet output with restricted memory. See “ToPA Errors” in Section 35.2.7.2 for details on ToPA errors, and Section 35.2.7.4 for details on restricted memory errors. Operational errors are only detected and signaled when TraceEn=1.

When an operational error is detected, tracing is disabled and the error is logged. Specifically, IA32_RTIT_STATUS.Error is set, which will cause IA32_RTIT_STATUS.TriggerEn to be 0. This will disable generation of all packets. Some causes of operational errors may lead to packet bytes being dropped.

It should be noted that the timing of error detection may not be predictable. Errors are signaled when the processor encounters the problematic configuration. This could be as soon as packet generation is enabled but could also be later when the problematic entry or field needs to be used.

Once an error is signaled, software should disable packet generation by clearing TraceEn, diagnose and fix the error condition, and clear IA32_RTIT_STATUS.Error. At this point, packet generation can be re-enabled.

35.4 TRACE PACKETS AND DATA TYPES

This section details the data packets generated by Intel Processor Trace. It is useful for developers writing the interpretation code that will decode the data packets and apply it to the traced source code.

35.4.1 Packet Relationships and Ordering

This section introduces the concept of packet “binding”, which involves determining the IP in a binary disassembly at which the change indicated by a given packet applies. Some packets have the associated IP as the payload (FUP, TIP), while for others the decoder need only search for the next instance of a particular instruction (or instructions) to bind the packet (TNT). However, in many cases, the decoder will need to consider the relationship between packets, and to use this packet context to determine how to bind the packet.

Section 35.4.1.1 below provides detailed descriptions of the packets, including how packets bind to IPs in the disassembly, to other packets, or to nothing at all. Many packets listed are simple to bind, because they are generated in only a few scenarios. Those that require more consideration are typically part of “compound packet events”, such as interrupts, exceptions, and some instructions, where multiple packets are generated by a single operation (instruction or event). These compound packet events frequently begin with a FUP to indicate the source address (if it is not clear from the disassembly), and are concluded by a TIP or TIP.PGD packet that indicates the destination address (if one is provided). In this scenario, the FUP is said to be “coupled” with the TIP packet.

Other packets could be in between the coupled FUP and TIP packet. Timing packets, such as TSC, MTC, CYC, or CBR, could arrive at any time, and hence could intercede in a compound packet event. If an operation changes CR3 or the processor’s mode of execution, a state update packet (i.e., PIP or MODE) is generated. The state changes indicated by these intermediate packets should be applied at the IP of the TIP* packet. A summary of compound packet events is provided in Table 35-14; see Section 35.4.1.1 for more per-packet details and Section 35.7 for more detailed packet generation examples.

Table 35-14. Compound Packet Event Summary

Event Type	Beginning	Middle	End	Comment
Unconditional, uncompressed control-flow transfer	FUP or none	Any combination of PIP, VMCS, MODE.Exec, or none	TIP or TIP.PGD	FUP only for asynchronous events. Order of middle packets may vary. PIP/VMCS/MODE only if the operation modifies the state tracked by these respective packets.
TSX Update	MODE.TSX, and (FUP or none)	None	TIP, TIP.PGD, or none	FUP TIP/TIP.PGD only for TSX abort cases.
Overflow	OVF	PSB, PSBEND, or none	FUP or TIP.PGE	FUP if overflow resolves while ContextEn=1, else TIP.PGE.

35.4.1.1 Packet Blocks

Packet blocks are a means to dump one or more groups of state values. Packet blocks begin with a Block Begin Packet (BBP), which indicates what type of state is held within the block. Following each BBP there may be one or more Block Item Packets (BIPs), which contain the state values. The block is terminated by either a Block End Packet (BEP) or another BBP indicating the start of a new block.

The BIP packet includes an ID value that, when combined with the Type field from the BBP that preceded it, uniquely identifies the state value held in the BIP payload. The size of each BIP packet payload is provided by the Size field in the preceding BBP packet.

Each block type can have up to 32 items defined for it. There is no guarantee, however, that each block of that type will hold all 32 items. For more details on which items to expect, see documentation on the specific block type of interest.

See the BBP packet description (Section 35.4.2.26) for details on packet block generation scenarios.

Packet blocks are entirely generated within an instruction or between instructions, which dictates the types of packets (aside from BIPs) that may be seen within a packet block. Packets that indicate control flow changes, or other indication of instruction completion, cannot be generated within a block. These are listed in the following table. Other packets, including timing packets, may occur between BBP and BEP.

Table 35-15. Packets Forbidden Between BBP and BEP

TNT
TIP, TIP.PGE, TIP.PGD
MODE.Exec, MODE.TSX
PIP, VMCS
TraceStop
PSB, PSBEND
PTW
MWAIT

It is possible to encounter an internal buffer overflow in the middle of a block. In such a case, it is guaranteed that packet generation will not resume in the middle of a block, and hence the OVF packet terminates the current block. Depending on the duration of the overflow, subsequent blocks may also be lost.

Decoder Implications

When a Block Begin Packet (BBP) is encountered, the decoder will need to decode some packets within the block differently from those outside a block. The Block Item Packet (BIP) header byte has the same encoding as a TNT packet outside of a block, but must be treated as a BIP header (with following payload) within one.

When an OVF packet is encountered, the decoder should treat that as a block ending condition. Packet generation will not resume within a block.

35.4.2 Packet Definitions

The following description of packet definitions are in tabular format. Figure 35-3 explains how to interpret them. Packet bits listed as “RSVD” are not guaranteed to be 0.

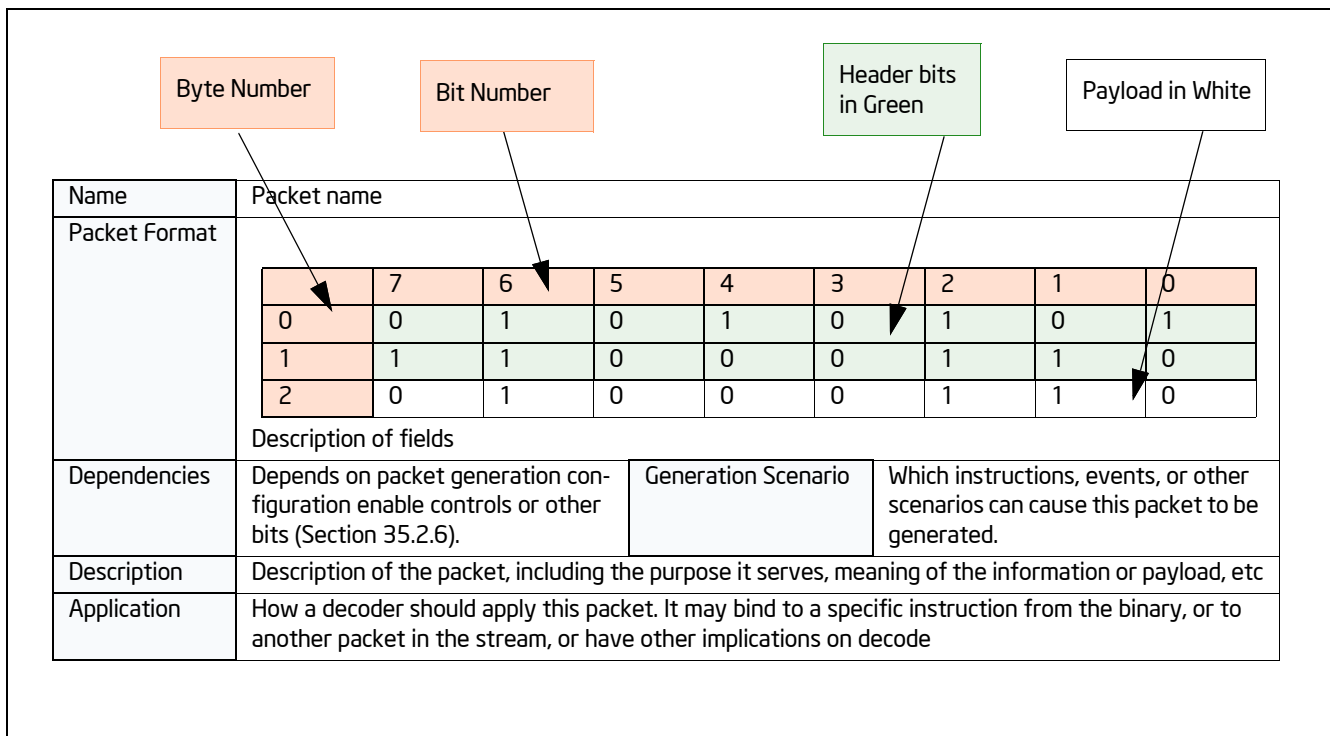


Figure 35-3. Interpreting Tabular Definition of Packet Format

35.4.2.1 Taken/Not-taken (TNT) Packet

Table 35-16. TNT Packet Definition

Name	Taken/Not-taken (TNT) Packet									
Packet Format										
		7	6	5	4	3	2	1	0	
	0	1	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	0	Short TNT
	B ₁ ...B _N represent the last N conditional branch or compressed RET (Section 35.4.2.2) results, such that B ₁ is oldest and B _N is youngest. The short TNT packet can contain from 1 to 6 TNT bits. The long TNT packet can contain from 1 to 47 TNT bits.									

Table 35-16. TNT Packet Definition (Contd.)

		7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	1	0	Long TNT
1	1	0	1	0	0	0	0	1	1	
2	B ₄₀	B ₄₁	B ₄₂	B ₄₃	B ₄₄	B ₄₅	B ₄₆	B ₄₇		
3	B ₃₂	B ₃₃	B ₃₄	B ₃₅	B ₃₆	B ₃₇	B ₃₈	B ₃₉		
4	B ₂₄	B ₂₅	B ₂₆	B ₂₇	B ₂₈	B ₂₉	B ₃₀	B ₃₁		
5	B ₁₆	B ₁₇	B ₁₈	B ₁₉	B ₂₀	B ₂₁	B ₂₂	B ₂₃		
6	B ₈	B ₉	B ₁₀	B ₁₁	B ₁₂	B ₁₃	B ₁₄	B ₁₅		
7	1	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇		
	Irrespective of how many TNT bits is in a packet, the last valid TNT bit is followed by a trailing 1, or Stop bit, as shown above. If the TNT packet is not full (fewer than 6 TNT bits for the Short TNT, or fewer than 47 TNT bits for the Long TNT), the Stop bit moves up, and the trailing bits of the packet are filled with 0s. Examples of these “partial TNTs” are shown below. An implementation may choose to use long TNTs, short TNTs, or both.									
		7	6	5	4	3	2	1	0	
0	0	0	1	B ₁	B ₂	B ₃	B ₄	0		Short TNT
		7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	1	0	Long TNT
1	1	0	1	0	0	0	0	1	1	
2	B ₂₄	B ₂₅	B ₂₆	B ₂₇	B ₂₈	B ₂₉	B ₃₀	B ₃₁		
3	B ₁₆	B ₁₇	B ₁₈	B ₁₉	B ₂₀	B ₂₁	B ₂₂	B ₂₃		
4	B ₈	B ₉	B ₁₀	B ₁₁	B ₁₂	B ₁₃	B ₁₄	B ₁₅		
5	1	B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇		
6	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	
Dependencies	PacketEn && ~IA32_R-TIT_CTL.DisTNT			Generation Scenario	On a conditional branch or compressed RET, if it fills the TNT. Also, partial TNTs may be generated at any time, as a result of other packets being generated, or certain micro-architectural conditions occurring, before the TNT is full.					
Description	Provides the taken/not-taken results for the last 1..6 (Short TNT) or 1..47 (Long TNT) conditional branches (Jcc, J*CXZ, or LOOP) or compressed RETs (Section 35.4.2.2). The TNT payload bits should be interpreted as follows: 1 indicates a taken conditional branch, or a compressed RET 0 indicates a not-taken conditional branch TNT payload bits are stored internal to the processor in a TNT buffer, until either the buffer is filled or another packet is to be generated. In either case a TNT packet holding the buffered bits will be emitted, and the TNT buffer will be marked as empty.									
Application	Each valid payload bit (that is, bits between the header bits and the trailing Stop bit) applies to an upcoming conditional branch or RET instruction. Once a decoder consumes a TNT packet with N valid payload bits, these bits should be applied to (and hence provide the destination for) the next N conditional branches or RETs.									

35.4.2.2 Target IP (TIP) Packet

Table 35-17. TIP Packet Definition

Name	Target IP (TIP) Packet																																																																																																	
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td colspan="3">IPBytes</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td colspan="8">TargetIP[7:0]</td></tr><tr><td>2</td><td colspan="8">TargetIP[15:8]</td></tr><tr><td>3</td><td colspan="8">TargetIP[23:16]</td></tr><tr><td>4</td><td colspan="8">TargetIP[31:24]</td></tr><tr><td>5</td><td colspan="8">TargetIP[39:32]</td></tr><tr><td>6</td><td colspan="8">TargetIP[47:40]</td></tr><tr><td>7</td><td colspan="8">TargetIP[55:48]</td></tr><tr><td>8</td><td colspan="8">TargetIP[63:56]</td></tr></table>									7	6	5	4	3	2	1	0	0	IPBytes			0	1	1	0	1	1	TargetIP[7:0]								2	TargetIP[15:8]								3	TargetIP[23:16]								4	TargetIP[31:24]								5	TargetIP[39:32]								6	TargetIP[47:40]								7	TargetIP[55:48]								8	TargetIP[63:56]							
	7	6	5	4	3	2	1	0																																																																																										
0	IPBytes			0	1	1	0	1																																																																																										
1	TargetIP[7:0]																																																																																																	
2	TargetIP[15:8]																																																																																																	
3	TargetIP[23:16]																																																																																																	
4	TargetIP[31:24]																																																																																																	
5	TargetIP[39:32]																																																																																																	
6	TargetIP[47:40]																																																																																																	
7	TargetIP[55:48]																																																																																																	
8	TargetIP[63:56]																																																																																																	
Dependencies	PacketEn	Generation Scenario	Indirect branch (including un-compressed RET), far branch, interrupt, exception, INIT, SIPI, VM exit, VM entry, TSX abort, EENTER, EEXIT, ERESUME, AEX ¹ .																																																																																															
Description	Provides the target for some control flow transfers																																																																																																	
Application	Anytime a TIP is encountered, it indicates that control was transferred to the IP provided in the payload. The source of this control flow change, and hence the IP or instruction to which it binds, depends on the packets that precede the TIP. If a TIP is encountered and all preceding packets have already been bound, then the TIP will apply to the upcoming indirect branch, far branch, or VMRESUME. However, if there was a preceding FUP that remains unbound, it will bind to the TIP. Here, the TIP provides the target of an asynchronous event or TSX abort that occurred at the IP given in the FUP payload. Note that there may be other packets, in addition to the FUP, which will bind to the TIP packet. See the packet application descriptions for other packets for details.																																																																																																	

NOTES:

1. EENTER, EEXIT, ERESUME, AEX would be possible only for a debug enclave.

IP Compression

The IP payload in a TIP, FUP, TIP.PGE, or TIP.PGD packet can vary in size, based on the mode of execution, and the use of IP compression. IP compression is an optional compression technique the processor may choose to employ to reduce bandwidth. With IP compression, the IP to be represented in the payload is compared with the last IP sent out, via any of FUP, TIP, TIP.PGE, or TIP.PGD. If that previous IP had the same upper (most significant) address bytes, those matching bytes may be suppressed in the current packet. The processor maintains an internal state of the “Last IP” that was encoded in trace packets, thus the decoder will need to keep track of the “Last IP” state in software, to match fidelity with packets generated by hardware. “Last IP” is initialized to zero, hence if the first IP in the trace may be compressed if the upper bytes are zeroes.

The “IPBytes” field of the IP packets (FUP, TIP, TIP.PGE, TIP.PGD) serves to indicate how many bytes of payload are provided, and how the decoder should fill in any suppressed bytes. The algorithm for reconstructing the IP for a TIP/FUP packet is shown in the table below.

Table 35-18. FUP/TIP IP Reconstruction

IPBytes	Uncompressed IP Value							
	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
000b	None, IP is out of context							
001b	Last IP[63:16]						IP Payload[15:0]	

Table 35-18. FUP/TIP IP Reconstruction

010b	Last IP[63:32]	IP Payload[31:0]
011b	IP Payload[47] extended	IP Payload[47:0]
100b	Last IP [63:48]	IP Payload[47:0]
101b	Reserved	
110b	IP Payload[63:0]	
111b	Reserved	

The processor-internal Last IP state is guaranteed to be reset to zero when a PSB is sent out. This means that the IP that follows the PSB with either be un-compressed (011b or 110b, see Table 35-18), or compressed against zero.

At times, “IPbytes” will have a value of 0. As shown above, this does not mean that the IP payload matches the full address of the last IP, but rather that the IP for this packet was suppressed. This is used for cases where the IP that applies to the packet is out of context. An example is the TIP.PGD sent on a SYSCALL, when tracing only USR code. In that case, no TargetIP will be included in the packet, since that would expose an instruction point at CPL = 0. When the IP payload is suppressed in this manner, Last IP is not cleared, and instead refers to the last IP packet with a non-zero IPBytes field.

On processors that support a maximum linear address size of 32 bits, IP payloads may never exceed 32 bits (IPBytes <= 010b).

Indirect Transfer Compression for Returns (RET)

In addition to IP compression, TIP packets for near return (RET) instructions can also be compressed. If the RET target matches the next IP of the corresponding CALL, then the TIP packet is unneeded, since the decoder can deduce the target IP by maintaining a CALL/RET stack of its own.

When a RET is compressed, a Taken indication is added to the TNT buffer. Because the RET generates no TIP packet, it also does not update the internal Last IP value, and thus the decoder should treat it the same way. If the RET is not compressed, it will generate a TIP packet (just like when RET compression is disabled, via IA32_R-TIT_CTL.DisRETC).

A CALL/RET stack can be maintained by the decoder by doing the following:

1. Allocate space to store 64 RET targets.
2. For near CALLs, push the Next IP onto the stack. Once the stack is full, new CALLs will force the oldest entry off the end of the stack, such that only the youngest 64 entries are stored. Note that this excludes zero-length CALLs, which are direct near CALLs with displacement zero (to the next IP). These CALLs typically don't have matching RETs.
3. For near RETs, pop the top (youngest) entry off the stack. This will be the expected target of the RET.

In cases where a RET is compressed, the RET target is guaranteed to match the expected target from 3) above. If the target is not compressed, a TIP packet will be generated with the RET target, which may differ from the expected target in some cases.

The hardware ensures that packets read by the decoder will always have seen the CALL that corresponds to any compressed RET. The processor will never compress a RET across a PSB, a buffer overflow, or scenario where PacketEn=0. This means that a RET whose corresponding CALL executed while PacketEn=0, or before the last PSB, etc., will not be compressed.

If the CALL/RET stack is manipulated or corrupted by software, and thereby causes a RET to transfer control to a target that is inconsistent with the CALL/RET stack, then the RET will not be compressed, and will produce a TIP packet. This can happen, for example, if software executes a PUSH instruction to push a target onto the stack, and a later RET uses this target.

For processors that employ deferred TIPs (Section 35.4.2.3), an uncompressed RET will not be deferred, and hence will force out any accumulated TNTs or TIPs. This serves to avoid ambiguity, and make clear to the decoder whether the near RET was compressed, and hence a bit in the in-progress TNT should be consumed, or uncompressed, in which case there will be no in-progress TNT and thus a TIP should be consumed.

Note that in the unlikely case that a RET executes in a different execution mode than the associated CALL, the decoder will need to model the same behavior with its CALL stack. For instance, if a CALL executes in 64-bit mode, a 64-bit IP value will be pushed onto the software stack. If the corresponding RET executes in 32-bit mode, then only the lower 32 target bits will be popped off of the stack, which may mean that the RET does not go to the CALL's Next IP. This is architecturally correct behavior, and this RET could be compressed, thus the decoder should match this behavior.

35.4.2.3 Deferred TIPs

The processor may opt to defer sending out the TNT when TIPs are generated. Thus, rather than sending a partial TNT followed by a TIP, both packets will be deferred while the TNT accumulates more Jcc/RET results. Any number of TIP packets may be accumulated this way, such that only once the TNT is filled, or once another packet (e.g., FUP) is generated, the TNT will be sent, followed by all the deferred TIP packets, and finally terminated by the other packet(s) that forced out the TNT and TIP packets. Generation of many other packets (see list below) will force out the TNT and any accumulated TIP packets. This is an optional optimization in hardware to reduce the bandwidth consumption, and hence the performance impact, incurred by tracing.

Table 35-19. TNT Examples with Deferred TIPs

Code Flow	Packets, Non-Deferred TIPS	Packets, Deferred TIPS
0x1000 cmp %rcx, 0 0x1004 jnz Foo // not-taken 0x1008 jmp %rdx	TNT(0b0), TIP(0x1308)	
0x1308 cmp %rcx, 1 0x130c jnz Bar // not-taken 0x1310 cmp %rcx, 2 0x1314 jnz Baz // taken 0x1500 cmp %eax, 7 0x1504 jg Exit // not-taken 0x1508 jmp %r15	TNT(0b010), TIP(0x1100)	
0x1100 cmp %rbx, 1 0x1104 jg Start // not-taken 0x1108 add %rcx, %eax 0x110c ... // an asynchronous Interrupt arrives INThandler: 0xcc00 pop %rdx	TNT(0b0), FUP(0x110c), TIP(0xcc00)	TNT(0b00100), TIP(0x1308), TIP(0x1100), FUP(0x110c), TIP(0xcc00)

35.4.2.4 Packet Generation Enable (TIP.PGE) Packet

Table 35-20. TIP.PGE Packet Definition

Name	Target IP - Packet Generation Enable (TIP.PGE) Packet

Table 35-20. TIP.PGE Packet Definition

Packet Format		7	6	5	4	3	2	1	0
	0	IPBytes			1	0	0	0	1
	1	TargetIP[7:0]							
	2	TargetIP[15:8]							
	3	TargetIP[23:16]							
	4	TargetIP[31:24]							
	5	TargetIP[39:32]							
	6	TargetIP[47:40]							
	7	TargetIP[55:48]							
	8	TargetIP[63:56]							
Dependencies	PacketEn transitions to 1				Generation Scenario	Any branch instruction, control flow transfer, or MOV CR3 that sets PacketEn, a WRMSR that enables packet generation and sets PacketEn			
Description	Indicates that PacketEn has transitioned to 1. It provides the IP at which the tracing begins. This can occur due to any of the enables that comprise PacketEn transitioning from 0 to 1, as long as all the others are asserted. Examples: ▪ TriggerEn: This is set on software write to set IA32_RTIT_CTL.TraceEn as long as the Stopped and Error bits in IA32_RTIT_STATUS are clear. The IP payload will be the Next IP of the WRMSR. ▪ FilterEn: This is set when software jumps into the tracing region. This region is defined by enabling IP filtering in IA32_RTIT_CTL.ADDRn_CFG, and defining the range in IA32_RTIT_ADDRn_[AB], see. Section 35.2.5.3. The IP payload will be the target of the branch. ▪ ContextEn: This is set on a CPL change, a CR3 write or any other means of changing ContextEn. The IP payload will be the Next IP of the instruction that changes context if it is not a branch, otherwise it will be the target of the branch.								
Application	TIP.PGE packets bind to the instruction at the IP given in the payload.								

35.4.2.5 Packet Generation Disable (TIP.PGD) Packet

Table 35-21. TIP.PGD Packet Definition

Name	Target IP - Packet Generation Disable (TIP.PGD) Packet								
Packet Format		7	6	5	4	3	2	1	0
	0	IPBytes			0	0	0	0	1
	1	TargetIP[7:0]							
	2	TargetIP[15:8]							
	3	TargetIP[23:16]							
	4	TargetIP[31:24]							
	5	TargetIP[39:32]							
	6	TargetIP[47:40]							
	7	TargetIP[55:48]							
	8	TargetIP[63:56]							
Dependencies	PacketEn transitions to 0			Generation Scenario	Any branch instruction, control flow transfer, or MOV CR3 that clears PacketEn, a WRMSR that disables packet generation and clears PacketEn				

Table 35-21. TIP.PGD Packet Definition

Description	Indicates that PacketEn has transitioned to 0. It will include the IP at which the tracing ends, unless ContextEn= 0 or TraceEn=0 at the conclusion of the instruction or event that cleared PacketEn. PacketEn can be cleared due to any of the enables that comprise PacketEn transitioning from 1 to 0. Examples: ▪ TriggerEn: This is cleared on software write to clear IA32_RTIT_CTL.TraceEn, or when IA32_RTIT_STATUS.Stopped is set, or on operational error. The IP payload will be suppressed in this case, and the “IPBytes” field will have the value 0. ▪ FilterEn: This is cleared when software jumps out of the tracing region. This region is defined by enabling IP filtering in IA32_RTIT_CTL.ADDRn_CFG, and defining the range in IA32_RTIT_ADDRn_[AB], see. Section 35.2.5.3. The IP payload will depend on the type of the branch. For conditional branches, the payload is suppressed (IPBytes = 0), and in this case the destination can be inferred from the disassembly. For any other type of branch, the IP payload will be the target of the branch. ▪ ContextEn: This can happen on a CPL change, a CR3 write or any other means of changing ContextEn. See Section 35.2.5.3 for details. In this case, when ContextEn is cleared, there will be no IP payload. The “IPBytes” field will have value 0. Note that, in cases where a branch that would normally produce a TIP packet (i.e., far transfer, indirect branch, interrupt, etc) or TNT update (conditional branch or compressed RT) causes PacketEn to transition from 1 to 0, the TIP or TNT bit will be replaced with TIP.PGD. The payload of the TIP.PGD will be the target of the branch, unless the result of the instruction causes TraceEn or ContextEn to be cleared (ie, SYSCALL when IA32_RTIT_CTL.OS=0, In the case where a conditional branch clears FilterEn and hence PacketEn, there will be no TNT bit for this branch, replaced instead by the TIP.PGD.
Application	TIP.PGD can be produced by any branch instructions, as well as some non-branch instructions, that clear PacketEn. When produced by a branch, it replaces any TIP or TNT update that the branch would normally produce. In cases where there is an unbound FUP preceding the TIP.PGD, then the TIP.PGD is part of compound operation (i.e., asynchronous event or TSX abort) which cleared PacketEn. For most such cases, the TIP.PGD is simply replacing a TIP, and should be treated the same way. The TIP.PGD may or may not have an IP payload, depending on whether the operation cleared ContextEn. If there is not an associated FUP, the binding will depend on whether there is an IP payload. If there is an IP payload, then the TIP.PGD should be applied to either the next direct branch whose target matches the TIP.PGD payload, or the next branch that would normally generate a TIP or TNT packet. If there is no IP payload, then the TIP.PGD should apply to the next branch or MOV CR3 instruction.

35.4.2.6 Flow Update (FUP) Packet

Table 35-22. FUP Packet Definition

Name	Flow Update (FUP) Packet																																																																																																	
Packet Format	<table border="1"> <tr> <th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> <tr> <td>0</td><td colspan="3">IPBytes</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td colspan="8">IP[7:0]</td></tr> <tr> <td>2</td><td colspan="8">IP[15:8]</td></tr> <tr> <td>3</td><td colspan="8">IP[23:16]</td></tr> <tr> <td>4</td><td colspan="8">IP[31:24]</td></tr> <tr> <td>5</td><td colspan="8">IP[39:32]</td></tr> <tr> <td>6</td><td colspan="8">IP[47:40]</td></tr> <tr> <td>7</td><td colspan="8">IP[55:48]</td></tr> <tr> <td>8</td><td colspan="8">IP[63:56]</td></tr> </table>									7	6	5	4	3	2	1	0	0	IPBytes			1	1	1	0	1	1	IP[7:0]								2	IP[15:8]								3	IP[23:16]								4	IP[31:24]								5	IP[39:32]								6	IP[47:40]								7	IP[55:48]								8	IP[63:56]							
	7	6	5	4	3	2	1	0																																																																																										
0	IPBytes			1	1	1	0	1																																																																																										
1	IP[7:0]																																																																																																	
2	IP[15:8]																																																																																																	
3	IP[23:16]																																																																																																	
4	IP[31:24]																																																																																																	
5	IP[39:32]																																																																																																	
6	IP[47:40]																																																																																																	
7	IP[55:48]																																																																																																	
8	IP[63:56]																																																																																																	

Table 35-22. FUP Packet Definition

Dependencies	TriggerEn && ContextEn. (Typically depends on BranchEn and FilterEn as well, see Section 35.2.5, Section 35.4.2.21, and Section 35.4.2.22 for details.)	Generation Scenario	Asynchronous Events (interrupts, exceptions, INIT, SIPI, SMI, VM exit, #MC), PSB+, XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, EENTER, EEXIT, ERESUME, EEE, AEX, ¹ , INTO, INT1, INT3, INT <i>n</i> , a WRMSR that disables packet generation.
Description	Provides the source address for asynchronous events, and some other instructions. Is never sent alone, always sent with an associated TIP or MODE packet, and potentially others.		
Application	FUP packets provide the IP to which they bind. However, they are never standalone, but are coupled with other packets. In TSX cases, the FUP is immediately preceded by a MODE.TSX, which binds to the same IP. A TIP will follow only in the case of TSX aborts, see Section 35.4.2.8 for details. Otherwise, FUPs are part of compound packet events (see Section 35.4.1). In these compound cases, the FUP provides the source IP for an instruction or event, while a following TIP (or TIP.PGD) packet will provide the destination IP. Other packets may be included in the compound event between the FUP and TIP.		

NOTES:

1. EENTER, EEXIT, ERESUME, EEE, AEX apply only if Intel Software Guard Extensions is supported.

FUP IP Payload

Flow Update Packet gives the source address of an instruction when it is needed. In general, branch instructions do not need a FUP, because the source address is clear from the disassembly. For asynchronous events, however, the source address cannot be inferred from the source, and hence a FUP will be sent. Table 35-23 illustrates cases where FUPs are sent, and which IP can be expected in those cases.

Table 35-23. FUP Cases and IP Payload

Event	Flow Update IP	Comment
External Interrupt, NMI/SMI, Traps, Machine Check (trap-like), INIT/SIPI	Address of next instruction (Next IP) that would have been executed	Functionally, this matches the LBR FROM field value and also the EIP value which is saved onto the stack.
Exceptions/Faults, Machine check (fault-like)	Address of the instruction which took the exception/fault (Current IP)	This matches the similar functionality of LBR FROM field value and also the EIP value which is saved onto the stack.
Software Interrupt	Address of the software interrupt instruction (Current IP)	This matches the similar functionality of LBR FROM field value, but does not match the EIP value which is saved onto the stack (Next Linear Instruction Pointer - NLIP).
EENTER, EEXIT, ERESUME, Enclave Exiting Event (EEE), AEX ¹	Current IP of the instruction	This matches the LBR FROM field value and also the EIP value which is saved onto the stack.
XACQUIRE	Address of the X* instruction	
XRELEASE, XBEGIN, XEND, XABORT, other transactional abort	Current IP	
#SMI	IP that is saved into SMRAM	
WRMSR that clears TraceEn, PSB+	Current IP	

NOTES:

1. Information on EENTER, EEXIT, ERESUME, EEE, Asynchronous Enclave eXit (AEX) can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D.

On a canonical fault due to sequentially fetching an instruction in non-canonical space (as opposed to jumping to non-canonical space), the IP of the fault (and thus the payload of the FUP) will be a non-canonical address. This is consistent with what is pushed on the stack for such faulting cases.

If there are post-commit task switch faults, the IP value of the FUP will be the original IP when the task switch started. This is the same value as would be seen in the LBR_FROM field. But it is a different value as is saved on the stack or VMCS.

35.4.2.7 Paging Information (PIP) Packet**Table 35-24. PIP Packet Definition**

Name	Paging Information (PIP) Packet																																																																																								
Packet Format	<table border="1"> <tr> <td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr> <td>2</td><td colspan="7">CR3[11:5] or 0</td><td>RSVD/NR</td></tr> <tr> <td>3</td><td colspan="8">CR3[19:12]</td></tr> <tr> <td>4</td><td colspan="8">CR3[27:20]</td></tr> <tr> <td>5</td><td colspan="8">CR3[35:28]</td></tr> <tr> <td>6</td><td colspan="8">CR3[43:36]</td></tr> <tr> <td>7</td><td colspan="8">CR3[51:44]</td></tr> </table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	1	1	2	CR3[11:5] or 0							RSVD/NR	3	CR3[19:12]								4	CR3[27:20]								5	CR3[35:28]								6	CR3[43:36]								7	CR3[51:44]							
	7	6	5	4	3	2	1	0																																																																																	
0	0	0	0	0	0	0	1	0																																																																																	
1	0	1	0	0	0	0	1	1																																																																																	
2	CR3[11:5] or 0							RSVD/NR																																																																																	
3	CR3[19:12]																																																																																								
4	CR3[27:20]																																																																																								
5	CR3[35:28]																																																																																								
6	CR3[43:36]																																																																																								
7	CR3[51:44]																																																																																								
Dependencies	TriggerEn && ContextEn && IA32_R-TIT_CTL.OS		Generation Scenario		MOV CR3, Task switch, INIT, SIPI, PSB+, VM exit, VM entry																																																																																				

Table 35-24. PIP Packet Definition

Description	The CR3 payload shown includes only the address portion of the CR3 value. For PAE paging, CR3[11:5] are thus included. For other paging modes (32-bit and 4-level paging¹), these bits are 0. This packet holds the CR3 address value. It will be generated on operations that modify CR3: ▪ MOV CR3 operation ▪ Task Switch ▪ INIT and SIPI ▪ VM exit, if “conceal VMX from PT” VM-exit control is 0 (see Section 35.5.1) ▪ VM entry, if “conceal VMX from PT” VM-entry control is 0 PIPs are not generated, despite changes to CR3, on SMI and RSM. This is due to the special behavior on these operations, see Section 35.2.9.3 for details. Note that, for some cases of task switch where CR3 is not modified, no PIP will be produced. The purpose of the PIP is to indicate to the decoder which application is running, so that it can apply the proper binaries to the linear addresses that are being traced. The PIP packet contains the new CR3 value when CR3 is written. PIPs generated by VM entries set the NR bit. PIPs generated in VMX non-root operation set the NR bit if the “conceal VMX from PT” VM-execution control is 0 (see Section 35.5.1). All other PIPs clear the NR bit.
Application	The purpose of the PIP packet is to help the decoder uniquely identify what software is running at any given time. When a PIP is encountered, a decoder should do the following: 1) If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this PIP is part of a compound packet event (Section 35.4.1). Find the ending TIP and apply the new CR3/NR values to the TIP payload IP. 2) Otherwise, look for the next MOV CR3, far branch, or VMRESUME/VMLAUNCH in the disassembly, and apply the new CR3 to the next (or target) IP. For examples of the packets generated by these flows, see Section 35.7.

NOTES:

1. Earlier versions of this manual used the term “IA-32e paging” to identify 4-level paging.

35.4.2.8 MODE Packets

MODE packets keep the decoder informed of various processor modes about which it needs to know in order to properly manage the packet output, or to properly disassemble the associated binaries. MODE packets include a header and a mode byte, as shown below.

Table 35-25. General Form of MODE Packets

	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1
1	Leaf ID			Mode				

The MODE Leaf ID indicates which set of mode bits are held in the lower bits.

MODE.Exec Packet**Table 35-26. MODE.Exec Packet Definition**

Name	MODE.Exec Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	1	0	0	1	1	0	0	1
	1	0	0	0	Reserved		IF	C.S.D	(C.S.L & LMA)
	MODE Leaf ID is '000.								

Table 35-26. MODE.Exec Packet Definition

Dependencies	TriggerEn && ContextEn && FilterEn	Generation Scenario	Any operation that changes the CS.L, CS.D, or EFER.LMA, if IA32_R- TIT_CTL.BranchEn=1. Any operation that changes RFLAGS.IF, if IA32_RTIT_CTL.EventEn=1. Any TIP.PGE scenario, such that any of the mode bits tracked may have changed since the last MODE.Exec.															
Description	Indicates whether software is in 16, 32, or 64-bit mode, by providing the CS.D and (CS.L & IA32_EFER.LMA) values. Essential for the decoder to properly disassemble the associated binary. Further, if CPUID.14H.00H:EBX[7]=1 (“Event Trace Support”), it indicates when interrupts are masked by providing the RFLAGS.IF value. MODE.Exec is sent at the time of a mode change, if dependencies are met at the time, otherwise it is sent when tracing resumes. In the former case, the MODE.Exec packet is generated along with other packets that result from the operation that changes the mode, and is guaranteed to be followed by a TIP or TIP.PGE for branch operations, or a FUP for non-branch operations (CLI, STI, or POPF if EventEn=1). In cases where the mode changes while filtering dependencies are not met, the processor ensures that the decoder doesn’t lose track of the mode by sending any needed MODE.Exec once tracing resumes (preceding the TIP.PGE, if BranchEn=1). The processor may opt to suppress the MODE.Exec when tracing resumes if the mode matches that of the last MODE.Exec packet. MODE.Exec packets are generated on CS.L, CS.D, or EFER.LMA changes only if control flow tracing is enabled (BranchEn=1). This is essential for the decoder to properly disassemble the associated binary. <table><tr><td>CS.D</td><td>(CS.L & IA32_EFER.LMA)</td><td>Addressing Mode</td></tr><tr><td>1</td><td>1</td><td>N/A</td></tr><tr><td>0</td><td>1</td><td>64-bit mode</td></tr><tr><td>1</td><td>0</td><td>32-bit mode</td></tr><tr><td>0</td><td>0</td><td>16-bit mode</td></tr></table> MODE.Exec packets are generated on interrupt flag (RFLAGS.IF) changes only if event tracing is enabled (EventEn=1). The IF field in MODE.Exec packets is populated only if EventEn=1 (IF = EFLAGS.IF & EventEn).			CS.D	(CS.L & IA32_EFER.LMA)	Addressing Mode	1	1	N/A	0	1	64-bit mode	1	0	32-bit mode	0	0	16-bit mode
CS.D	(CS.L & IA32_EFER.LMA)	Addressing Mode																
1	1	N/A																
0	1	64-bit mode																
1	0	32-bit mode																
0	0	16-bit mode																
Application	MODE.Exec always precedes an IP packet (TIP, TIP.PGE, or FUP). The mode change applies to the IP address in the payload of the IP packet. When MODE.Exec is followed by a FUP, it is a stand-alone FUP and should be consumed by the MODE.Exec.																	

MODE.TSX Packet**Table 35-27. MODE.TSX Packet Definition**

Name	MODE.TSX Packet																																		
Packet Format	<table border="1"> <thead> <tr> <th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr> </thead> <tbody> <tr> <td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>TXAbort</td><td>InTX</td></tr> </tbody> </table>									7	6	5	4	3	2	1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	TXAbort	InTX
	7	6	5	4	3	2	1	0																											
0	1	0	0	1	1	0	0	1																											
1	0	0	1	0	0	0	TXAbort	InTX																											
Dependencies	TriggerEn && ContextEn		Generation Scenario	XBEGIN, XEND, XABORT, XACQUIRE, XRELEASE, if InTX changes, Asynchronous TSX Abort, PSB+																															

Table 35-27. MODE.TSX Packet Definition

Description	Indicates when a TSX transaction (either HLE or RTM) begins, commits, or aborts. Instructions executed transactionally will be “rolled back” if the transaction is aborted.		
	TXAbort	InTX	Implication
	1	1	N/A
	0	1	Transaction begins, or executing transactionally
	1	0	Transaction aborted
	0	0	Transaction committed, or not executing transactionally
Application	If PacketEn=1, MODE.TSX always immediately precedes a FUP. If the TXAbort bit is zero, then the mode change applies to the IP address in the payload of the FUP. If TXAbort=1, then the FUP will be followed by a TIP, and the mode change will apply to the IP address in the payload of the TIP. MODE.TSX packets may be generated when PacketEn=0, due to FilterEn=0. In this case, only the last MODE.TSX generated before TIP.PGE need be applied.		

35.4.2.9 TraceStop Packet

Table 35-28. TraceStop Packet Definition

Name	TraceStop Packet																																		
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	1
	7	6	5	4	3	2	1	0																											
0	0	0	0	0	0	0	1	0																											
1	1	0	0	0	0	0	1	1																											
Dependencies	TriggerEn && ContextEn		Generation Scenario	Taken branch with target in TraceStop IP region, MOV CR3 in TraceStop IP region, or WRMSR that sets TraceEn in TraceStop IP region.																															
Description	Indicates when software has entered a user-configured TraceStop region. When the IP matches a TraceStop range while ContextEn and TriggerEn are set, a TraceStop action occurs. This disables tracing by setting IA32_RTIT_STATUS.Stopped, thereby clearing TriggerEn, and causes a TraceStop packet to be generated. The TraceStop action also forces FilterEn to 0. Note that TraceStop may not force a flush of internally buffered packets, and thus trace packet generation should still be manually disabled by clearing IA32_RTIT_CTL.TraceEn before examining output. See Section 35.2.5.3 for more details.																																		
Application	If TraceStop follows a TIP.PGD (before the next TIP.PGE), then it was triggered either by the instruction that cleared PacketEn, or it was triggered by some later instruction that executed while FilterEn=0. In either case, the TraceStop can be applied at the IP of the TIP.PGD (if any). If TraceStop follows a TIP.PGE (before the next TIP.PGD), it should be applied at the last known IP.																																		

35.4.2.10 Core:Bus Ratio (CBR) Packet

Table 35-29. CBR Packet Definition

Name	Core:Bus Ratio (CBR) Packet

Table 35-29. CBR Packet Definition

Packet Format		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	0	0	0	0	0	1	1
	2	Core:Bus Ratio							
	3	Reserved							
Dependencies	TriggerEn			Generation Scenario	After any frequency change, on C-state wake up, PSB+, and after enabling trace packet generation.				
Description	Indicates the core:bus ratio of the processor core. Useful for correlating wall-clock time and cycle time.								
Application	The CBR packet indicates the point in the trace when a frequency transition has occurred. On some implementations, software execution will continue during transitions to a new frequency, while on others software execution ceases during frequency transitions. There is not a precise IP provided, to which to bind the CBR packet.								

35.4.2.11 Timestamp Counter (TSC) Packet

Table 35-30. TSC Packet Definition

Name	Timestamp Counter (TSC) Packet																																																																																								
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td colspan="8">SW TSC[7:0]</td></tr><tr><td>2</td><td colspan="8">SW TSC[15:8]</td></tr><tr><td>3</td><td colspan="8">SW TSC[23:16]</td></tr><tr><td>4</td><td colspan="8">SW TSC[31:24]</td></tr><tr><td>5</td><td colspan="8">SW TSC[39:32]</td></tr><tr><td>6</td><td colspan="8">SW TSC[47:40]</td></tr><tr><td>7</td><td colspan="8">SW TSC[55:48]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	1	1	0	0	1	1	SW TSC[7:0]								2	SW TSC[15:8]								3	SW TSC[23:16]								4	SW TSC[31:24]								5	SW TSC[39:32]								6	SW TSC[47:40]								7	SW TSC[55:48]							
		7	6	5	4	3	2	1	0																																																																																
	0	0	0	0	1	1	0	0	1																																																																																
	1	SW TSC[7:0]																																																																																							
	2	SW TSC[15:8]																																																																																							
	3	SW TSC[23:16]																																																																																							
	4	SW TSC[31:24]																																																																																							
	5	SW TSC[39:32]																																																																																							
	6	SW TSC[47:40]																																																																																							
7	SW TSC[55:48]																																																																																								
Dependencies	IA32_RTIT_CTL.TSCEn && TriggerEn		Generation Scenario	Sent after any event that causes the processor clocks or Intel PT timing packets (such as MTC or CYC) to stop, This may include P-state changes, wake from C-state, or clock modulation. Also on transition of TraceEn from 0 to 1.																																																																																					
Description	When enabled by software, a TSC packet provides the lower 7 bytes of the current TSC value, as returned by the RDTSC instruction. This may be useful for tracking wall-clock time, and synchronizing the packets in the log with other timestamped logs.																																																																																								
Application	TSC packet provides a wall-clock proxy of the event which generated it (packet generation enable, sleep state wake, etc). In all cases, TSC does not precisely indicate the time of any control flow packets; however, all preceding packets represent instructions that executed before the indicated TSC time, and all subsequent packets represent instructions that executed after it. There is not a precise IP to which to bind the TSC packet.																																																																																								

35.4.2.12 Mini Time Counter (MTC) Packet

Table 35-31. MTC Packet Definition

Name	Mini time Counter (MTC) Packet

Table 35-31. MTC Packet Definition

Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td colspan="8">CTC[N+7:N]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	1	0	1	1	0	0	1	1	CTC[N+7:N]							
	7	6	5	4	3	2	1	0																											
0	0	1	0	1	1	0	0	1																											
1	CTC[N+7:N]																																		
Dependencies	IA32_RTIT_CTL.MTCEn && TriggerEn	Generation Scenario	Periodic, based on the core crystal clock, or Always Running Timer (ART).																																
Description	When enabled by software, an MTC packet provides a periodic indication of wall-clock time. The 8-bit CTC (Common Timestamp Copy) payload value is set to (ART >> N) & FFH. The frequency of the ART is related to the Maximum Non-Turbo frequency, and the ratio can be determined from CPUID.15H, as described in Section 35.8.3. Software can select the threshold N, which determines the MTC frequency by setting the IA32_RTIT_CTL.MTCFreq field (see Section 35.2.8.2) to a supported value using the lookup enumerated by CPUID (see Section 35.3.1). See Section 35.8.3 for details on how to use the MTC payload to track TSC time. MTC provides 8 bits from the ART, starting with the bit selected by MTCFreq to dictate the frequency of the packet. Whenever that 8-bit range being watched changes, an MTC packet will be sent out with the new value of that 8-bit range. This allows the decoder to keep track of how much wall-clock time has elapsed since the last TSC packet was sent, by keeping track of how many MTC packets were sent and what their value was. The decoder can infer the truncated bits, CTC[N-1:0], are 0 at the time of the MTC packet. There are cases in which MTC packet can be dropped, due to overflow or other micro-architectural conditions. The decoder should be able to recover from such cases by checking the 8-bit payload of the next MTC packet, to determine how many MTC packets were dropped. It is not expected that >256 consecutive MTC packets should ever be dropped.																																		
Application	MTC does not precisely indicate the time of any other packet, nor does it bind to any IP. However, all preceding packets represent instructions or events that executed before the indicated ART time, and all subsequent packets represent instructions that executed after, or at the same time as, the ART time.																																		

35.4.2.13 TSC/MTC Alignment (TMA) Packet**Table 35-32. TMA Packet Definition**

Name	TSC/MTC Alignment (TMA) Packet																																																																															
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td colspan="8">CTC[7:0]</td></tr><tr><td>3</td><td colspan="8">CTC[15:8]</td></tr><tr><td>4</td><td colspan="7">Reserved</td><td>0</td></tr><tr><td>5</td><td colspan="8">FastCounter[7:0]</td></tr><tr><td>6</td><td colspan="7">Reserved</td><td>FC[8]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	0	1	1	2	CTC[7:0]								3	CTC[15:8]								4	Reserved							0	5	FastCounter[7:0]								6	Reserved							FC[8]
	7	6	5	4	3	2	1	0																																																																								
0	0	0	0	0	0	0	1	0																																																																								
1	0	1	1	1	0	0	1	1																																																																								
2	CTC[7:0]																																																																															
3	CTC[15:8]																																																																															
4	Reserved							0																																																																								
5	FastCounter[7:0]																																																																															
6	Reserved							FC[8]																																																																								
Dependencies	IA32_RTIT_CTL.MTCEn && IA32_RTIT_CTL.TSCEn && TriggerEn			Generation Scenario		Sent with any TSC packet.																																																																										
Description	The TMA packet serves to provide the information needed to allow the decoder to correlate MTC packets with TSC packets. With this packet, when a MTC packet is encountered, the decoder can determine how many timestamp counter ticks have passed since the last TSC or MTC packet. See Section 35.8.3.2 for details on how to make this calculation.																																																																															
Application	TMA is always sent immediately following a TSC packet, and the payload values are consistent with the TSC payload value. Thus the application of TMA matches that of TSC.																																																																															

35.4.2.14 Cycle Count (CYC) Packet

Table 35-33. Cycle Count Packet Definition

Name	Cycle Count (CYC) Packet																																															
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td colspan="5">Cycle Counter[4:0]</td><td>Exp</td><td>1</td><td>1</td></tr><tr><td>1</td><td colspan="7">Cycle Counter[11:5]</td><td>Exp</td></tr><tr><td>2</td><td colspan="7">Cycle Counter[18:12]</td><td>Exp</td></tr><tr><td>...</td><td colspan="8">... (if Exp = 1 in the previous byte)</td></tr></table>				7	6	5	4	3	2	1	0	0	Cycle Counter[4:0]					Exp	1	1	1	Cycle Counter[11:5]							Exp	2	Cycle Counter[18:12]							Exp	...	... (if Exp = 1 in the previous byte)							
	7	6	5	4	3	2	1	0																																								
0	Cycle Counter[4:0]					Exp	1	1																																								
1	Cycle Counter[11:5]							Exp																																								
2	Cycle Counter[18:12]							Exp																																								
...	... (if Exp = 1 in the previous byte)																																															
Dependencies	IA32_RTIT_CTL.CYCEn&&TriggerEn	Generation Scenario	Can be sent at any time, though a maximum of one CYC packet is sent per core clock cycle. See Section 35.3.6 for CYC-eligible packets.																																													
Description	The Cycle Counter field increments at the same rate as the processor core clock ticks, but with a variable length format (using a trailing EXP bit field) and a range-capped byte length. If the CYC value is less than 32, a 1-byte CYC will be generated, with Exp=0. If the CYC value is between 32 and 4095 inclusive, a 2-byte CYC will be generated, with byte 0 Exp=1 and byte 1 Exp=0. And so on. CYC provides the number of core clocks that have passed since the last CYC packet. CYC can be configured to be sent in every cycle in which an eligible packet is generated, or software can opt to use a threshold to limit the number of CYC packets, at the expense of some precision. These settings are configured using the IA32_RTIT_CTL.CycThresh field (see Section 35.2.8.2). For details on Cycle-Accurate Mode, IPC calculation, etc, see Section 35.3.6. When CycThresh=0, and hence no threshold is in use, then a CYC packet will be generated in any cycle in which any CYC-eligible packet is generated. The CYC packet will precede the other packets generated in the cycle, and provides the precise cycle time of the packets that follow. In addition to these CYC packets generated with other packets, CYC packets can be sent stand-alone. These packets serve simply to update the decoder with the number of cycles passed, and are used to ensure that a wrap of the processor's internal cycle counter doesn't cause cycle information to be lost. These stand-alone CYC packets do not indicate the cycle time of any other packet or operation, and will be followed by another CYC packet before any other CYC-eligible packet is seen. When CycThresh>0, CYC packets are generated only after a minimum number of cycles have passed since the last CYC packet. Once this threshold has passed, the behavior above resumes, where CYC will either be sent in the next cycle that produces other CYC-eligible packets, or could be sent stand-alone. When using CYC thresholds, only the cycle time of the operation (instruction or event) that generates the CYC packet is truly known. Other operations simply have their execution time bounded: they completed at or after the last CYC time, and before the next CYC time.																																															
Application	CYC provides the offset cycle time (since the last CYC packet) for the CYC-eligible packet that follows. If another CYC is encountered before the next CYC-eligible packet, the cycle values should be accumulated and applied to the next CYC-eligible packet. If a CYC packet is generated by a TNT, note that the cycle time provided by the CYC packet applies to the first branch in the TNT packet.																																															

35.4.2.15 VMCS Packet

Table 35-34. VMCS Packet Definition

Name	VMCS Packet

Table 35-34. VMCS Packet Definition

Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>2</td><td colspan="8">VMCS pointer [19:12]</td></tr><tr><td>3</td><td colspan="8">VMCS pointer [27:20]</td></tr><tr><td>4</td><td colspan="8">VMCS pointer [35:28]</td></tr><tr><td>5</td><td colspan="8">VMCS pointer [43:36]</td></tr><tr><td>6</td><td colspan="8">VMCS pointer [51:44]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	0	0	0	2	VMCS pointer [19:12]								3	VMCS pointer [27:20]								4	VMCS pointer [35:28]								5	VMCS pointer [43:36]								6	VMCS pointer [51:44]							
	7	6	5	4	3	2	1	0																																																																								
0	0	0	0	0	0	0	1	0																																																																								
1	1	1	0	0	1	0	0	0																																																																								
2	VMCS pointer [19:12]																																																																															
3	VMCS pointer [27:20]																																																																															
4	VMCS pointer [35:28]																																																																															
5	VMCS pointer [43:36]																																																																															
6	VMCS pointer [51:44]																																																																															
Dependencies	TriggerEn && ContextEn; Also in VMX operation.		Generation Scenario		Generated on successful VMPTRLD, and optionally on PSB+, SMM VM exits, and VM entries that return from SMM (see Section 35-53).																																																																											
Description	The VMCS packet provides a VMCS pointer for a decoder to determine the transition of code contexts: - On a successful VMPTRLD (i.e., a VMPTRLD that doesn't fault, fail, or VM exit), the VMCS packet contains the logical processor's VMCS pointer established by VMPTRLD (for subsequent execution of a VM guest context). - An SMM VM exit loads the logical processor's VMCS pointer with the SMM-transfer VMCS pointer. If the "conceal VMX from PT" VM-exit control is 0 (see Section 35.5.1), a VMCS packet provides this pointer. See Section 35.6 on tracing inside and outside STM. - A VM entry that returns from SMM loads the logical processor's VMCS pointer from a field in the SMM-transfer VMCS. If the "conceal VMX from PT" VM-entry control is 0, a VMCS packet provides this pointer. Whether the VM entry is to VMX root operation or VMX non-root operation is indicated by the PIP.NR bit. A VMCS packet generated before a VMCS pointer has been loaded, or after the VMCS pointer has been cleared will set all 64 bits in the VMCS pointer field. VMCS packets will not be seen on processors with IA32_VMX_MISC[bit 14]=0, as these processors do not allow TraceEn to be set in VMX operation.																																																																															
Application	The purpose of the VMCS packet is to help the decoder uniquely identify changes in the executing software context in situations that CR3 may not be unique. When a VMCS packet is encountered, a decoder should do the following: - If there was a prior unbound FUP (that is, a FUP not preceded by a packet such as MODE.TSX that consumes it, and it hence pairs with a TIP that has not yet been seen), then this VMCS is part of a compound packet event (Section 35.4.1). Find the ending TIP and apply the new VMCS base pointer value to the TIP payload IP. - Otherwise, look for the next VMPTRLD, VMRESUME, or VMLAUNCH in the disassembly, and apply the new VMCS base pointer on the next VM entry. For examples of the packets generated by these flows, see Section 35.7.																																																																															

35.4.2.16 Overflow (OVF) Packet

Table 35-35. OVF Packet Definition

Name	Overflow (OVF) Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	1	1	1	1	0	0	1	1
Dependencies	TriggerEn		Generation Scenario		On resolution of internal buffer overflow				

Table 35-35. OVF Packet Definition

Description	OVF simply indicates to the decoder that an internal buffer overflow occurred, and packets were likely lost. If BranchEN= 1, OVF is followed by a FUP or TIP.PGE which will provide the IP at which packet generation resumes. See Section 35.3.8.
Application	When an OVF packet is encountered, the decoder should skip to the IP given in the subsequent FUP or TIP.PGE. The cycle counter for the CYC packet will be reset at the time the OVF packet is sent. Software should reset its call stack depth on overflow, since no RET compression is allowed across an overflow. Similarly, any IP compression that follows the OVF is guaranteed to use as a reference LastIP the IP payload of an IP packet that preceded the overflow.

35.4.2.17 Packet Stream Boundary (PSB) Packet

Table 35-36. PSB Packet Definition

Name	Packet Stream Boundary (PSB) Packet																																																																																																																																																																
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>3</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>5</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>6</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>7</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>8</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>9</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>10</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>11</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>12</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>13</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>14</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>15</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	2	0	0	0	0	0	0	1	0	3	1	0	0	0	0	0	1	0	4	0	0	0	0	0	0	1	0	5	1	0	0	0	0	0	1	0	6	0	0	0	0	0	0	1	0	7	1	0	0	0	0	0	1	0	8	0	0	0	0	0	0	1	0	9	1	0	0	0	0	0	1	0	10	0	0	0	0	0	0	1	0	11	1	0	0	0	0	0	1	0	12	0	0	0	0	0	0	1	0	13	1	0	0	0	0	0	1	0	14	0	0	0	0	0	0	1	0	15	1	0	0	0	0	0	1	0
	7	6	5	4	3	2	1	0																																																																																																																																																									
0	0	0	0	0	0	0	1	0																																																																																																																																																									
1	1	0	0	0	0	0	1	0																																																																																																																																																									
2	0	0	0	0	0	0	1	0																																																																																																																																																									
3	1	0	0	0	0	0	1	0																																																																																																																																																									
4	0	0	0	0	0	0	1	0																																																																																																																																																									
5	1	0	0	0	0	0	1	0																																																																																																																																																									
6	0	0	0	0	0	0	1	0																																																																																																																																																									
7	1	0	0	0	0	0	1	0																																																																																																																																																									
8	0	0	0	0	0	0	1	0																																																																																																																																																									
9	1	0	0	0	0	0	1	0																																																																																																																																																									
10	0	0	0	0	0	0	1	0																																																																																																																																																									
11	1	0	0	0	0	0	1	0																																																																																																																																																									
12	0	0	0	0	0	0	1	0																																																																																																																																																									
13	1	0	0	0	0	0	1	0																																																																																																																																																									
14	0	0	0	0	0	0	1	0																																																																																																																																																									
15	1	0	0	0	0	0	1	0																																																																																																																																																									
Dependencies	TriggerEn	Generation Scenario	Periodic, based on the number of output bytes generated while tracing. PSB is sent when IA32_RTIT_STATUS.PacketByteCnt=0, and each time it crosses the software selected threshold after that. May be sent for other micro-architectural conditions as well.																																																																																																																																																														
Description	PSB is a unique pattern in the packet output log, and hence serves as a sync point for the decoder. It is a pattern that the decoder can search for in order to get aligned on packet boundaries. This packet is periodic, based on the number of output bytes, as indicated by IA32_RTIT_STATUS.PacketByteCnt. The period is chosen by software, via IA32_RTIT_CTL.PSBFreq (see Section 35.2.8.2). Note, however, that the PSB period is not precise, it simply reflects the average number of output bytes that should pass between PSBs. The processor will make a best effort to insert PSB as quickly after the selected threshold is reached as possible. The processor also may send extra PSB packets for some micro-architectural conditions. PSB also serves as the leading packet for a set of “status-only” packets collectively known as PSB+ (Section 35.3.7).																																																																																																																																																																

Table 35-36. PSB Packet Definition (Contd.)

Application	When a PSB is seen, the decoder should interpret all following packets as “status only”, until either a PSBEND or OVF packet is encountered. “Status only” implies that the binding and ordering rules to which these packets normally adhere are ignored, and the state they carry can instead be applied to the IP payload in the FUP packet that is included.

35.4.2.18 PSBEND Packet**Table 35-37. PSBEND Packet Definition**

Name	PSBEND Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	0	1	0	0	0	1	1
Dependencies	TriggerEn			Generation Scenario		Always follows PSB packet, separated by PSB+ packets			
Description	PSBEND is simply a terminator for the series of “status only” (PSB+) packets that follow PSB (Section 35.3.7).								
Application	When a PSBEND packet is seen, the decoder should cease to treat packets as “status only”.								

35.4.2.19 Maintenance (MNT) Packet**Table 35-38. MNT Packet Definition**

Name	Maintenance (MNT) Packet								
Packet Format		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	1	1	0	0	0	0	1	1
	2	1	0	0	0	1	0	0	0
	3	Payload[7:0]							
	4	Payload[15:8]							
	5	Payload[23:16]							
	6	Payload[31:24]							
	7	Payload[39:32]							
	8	Payload[47:40]							
	9	Payload[55:48]							
	10	Payload[63:56]							
Dependencies	TriggerEn		Generation Scenario		Implementation specific.				
Description	This packet is generated by hardware, the payload meaning is model-specific.								
Application	Unless a decoder has been extended for a particular family/model/stepping to interpret MNT packet payloads, this packet should simply be ignored. It does not bind to any IP.								

35.4.2.20 PAD Packet

Table 35-39. PAD Packet Definition

Name	PAD Packet																										
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>										7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	0	0
	7	6	5	4	3	2	1	0																			
0	0	0	0	0	0	0	0	0																			
Dependencies	TriggerEn			Generation Scenario		Implementation specific																					
Description	PAD is simply a NOP packet. Processor implementations may choose to add pad packets to improve packet alignment or for implementation-specific reasons.																										
Application	Ignore PAD packets.																										

35.4.2.21 PTWRITE (PTW) Packet

Table 35-40. PTW Packet Definition

Name	PTW Packet																																																																																																										
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>IP</td><td colspan="2">PayloadBytes</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td colspan="8">Payload[7:0]</td></tr><tr><td>3</td><td colspan="8">Payload[15:8]</td></tr><tr><td>4</td><td colspan="8">Payload[23:16]</td></tr><tr><td>5</td><td colspan="8">Payload[31:24]</td></tr><tr><td>6</td><td colspan="8">Payload[39:32]</td></tr><tr><td>7</td><td colspan="8">Payload[47:40]</td></tr><tr><td>8</td><td colspan="8">Payload[55:48]</td></tr><tr><td>9</td><td colspan="8">Payload[63:56]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	IP	PayloadBytes		1	0	0	1	0	2	Payload[7:0]								3	Payload[15:8]								4	Payload[23:16]								5	Payload[31:24]								6	Payload[39:32]								7	Payload[47:40]								8	Payload[55:48]								9	Payload[63:56]							
	7	6	5	4	3	2	1	0																																																																																																			
0	0	0	0	0	0	0	1	0																																																																																																			
1	IP	PayloadBytes		1	0	0	1	0																																																																																																			
2	Payload[7:0]																																																																																																										
3	Payload[15:8]																																																																																																										
4	Payload[23:16]																																																																																																										
5	Payload[31:24]																																																																																																										
6	Payload[39:32]																																																																																																										
7	Payload[47:40]																																																																																																										
8	Payload[55:48]																																																																																																										
9	Payload[63:56]																																																																																																										
	The PayloadBytes field indicates the number of bytes of payload that follow the header bytes. Encodings are as follows: <table><tr><th>PayloadBytes</th><th>Bytes of Payload</th></tr><tr><td>'00</td><td>4</td></tr><tr><td>'01</td><td>8</td></tr><tr><td>'10</td><td>Reserved</td></tr><tr><td>'11</td><td>Reserved</td></tr></table> IP bit indicates if a FUP, whose payload will be the IP of the PTWRITE instruction, will follow.								PayloadBytes	Bytes of Payload	'00	4	'01	8	'10	Reserved	'11	Reserved																																																																																									
PayloadBytes	Bytes of Payload																																																																																																										
'00	4																																																																																																										
'01	8																																																																																																										
'10	Reserved																																																																																																										
'11	Reserved																																																																																																										
Dependencies	TriggerEn && ContextEn && FilterEn && PTWEn		Generation Scenario		PTWRITE Instruction																																																																																																						

Table 35-40. PTW Packet Definition (Contd.)

Description	Contains the value held in the PTWRITE operand. This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.
Application	Binds to the associated PTWRITE instruction. The IP of the PTWRITE will be provided by a following FUP, when PTW.IP=1.

35.4.2.22 Execution Stop (EXSTOP) Packet

Table 35-41. EXSTOP Packet Definition

Name	EXSTOP Packet																													
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>IP</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table> IP bit indicates if a FUP will follow.				7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	IP	1	1	0	0	0	1	0
	7	6	5	4	3	2	1	0																						
0	0	0	0	0	0	0	1	0																						
1	IP	1	1	0	0	0	1	0																						
Dependencies	TriggerEn && PwrEvtEn	Generation Scenario	C-state entry, P-state change, or other processor clock power-down. Includes : ▪ Entry to C-state deeper than C0.0 ▪ TM1/2 ▪ STPCLK# ▪ Frequency change due to IA32_CLOCK_MODULATION, Turbo																											
Description	This packet indicates that software execution has stopped due to processor clock powerdown. Later packets will indicate when execution resumes. If EXSTOP is generated while ContextEn is set, the IP bit will be set, and EXSTOP will be followed by a FUP packet containing the IP at which execution stopped. More precisely, this will be the IP of the oldest instruction that has not yet completed. This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.																													
Application	If a FUP follows EXSTOP (hence IP bit set), the EXSTOP can be bound to the FUP IP. Otherwise the IP is not known. Time of powerdown can be inferred from the preceding CYC, if CYCEn=1. Combined with the TSC at the time of wake (if TSCEn=1), this can be used to determine the duration of the powerdown.																													

35.4.2.23 MWAIT Packet

Table 35-42. MWAIT Packet Definition

Name	MWAIT Packet

Table 35-42. MWAIT Packet Definition (Contd.)

Packet Format		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	1	1	0	0	0	0	1	0
	2	MWAIT Hints[7:0]							
	3	Reserved							
	4	Reserved							
	5	Reserved							
	6	Reserved						EXT[1:0]	
	7	Reserved							
	8	Reserved							
	9	Reserved							
Dependencies	TriggerEn && PwrEvtEn && ContextEn			Generation Scenario	MWAIT, UMWAIT, or TPAUSE instructions, or I/O redirection to MWAIT, that complete without fault or VMexit.				
Description	Indicates that an MWAIT operation to C-state deeper than C0.0 completed. The MWAIT hints and extensions passed in by software are exposed in the payload. For UMWAIT and TPAUSE, the EXT field holds the input register value that determines the optimized state requested. For entry to some highly optimized C0 sub-C-states, such as C0.1, no MWAIT packet is generated. This packet is CYC-eligible, and hence will generate a CYC packet if IA32_RTIT_CTL.CYCEn=1 and any CYC Threshold has been reached.								
Application	The binding for the upcoming EXSTOP packet also applies to the MWAIT packet. See Section 35.4.2.22.								

35.4.2.24 Power Entry (PWRE) Packet

Table 35-43. PWRE Packet Definition

Name	PWRE Packet								
Packet Format									
		7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	1	0
	1	0	0	1	0	0	0	1	0
	2	HW	Reserved						
	3	Resolved Thread C-State				Resolved Thread Sub C-State			
Dependencies	TriggerEn && PwrEvtEn			Generation Scenario	Transition to a C-state deeper than C0.0.				

Table 35-43. PWRE Packet Definition (Contd.)

Description	Indicates processor entry to the resolved thread C-state and sub C-state indicated. The processor will remain in this C-state until either another PWRE indicates the processor has moved to a C-state deeper than C0.0, or a PWRX packet indicates a return to C0.0. For entry to some highly optimized C0 sub-C-states, such as C0.1, no PWRE packet is generated. Note that some CPUs may allow MWAIT to request a deeper C-state than is supported by the core. These deeper C-states may have platform-level implications that differentiate them. However, the PWRE packet will provide only the resolved thread C-state, which will not exceed that supported by the core. If the C-state entry was initiated by hardware, rather than a direct software request (such as MWAIT, UMWAIT, TPAUSE, HLT, or shutdown), the HW bit will be set to indicate this. Hardware Duty Cycling (see Section 16.5, “Hardware Duty Cycling (HDC),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B) is an example of such a case.
Application	When transitioning from C0.0 to a deeper C-state, the PWRE packet will be followed by an EXSTOP. If that EXSTOP packet has the IP bit set, then the following FUP will provide the IP at which the C-state entry occurred. Subsequent PWRE packets generated before the next PWRX should bind to the same IP.

35.4.2.25 Power Exit (PWRX) Packet

Table 35-44. PWRX Packet Definition

Name	PWRX Packet																																																																															
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td colspan="4">Last Core C-State</td><td colspan="4">Deepest Core C-State</td></tr><tr><td>3</td><td colspan="4">Reserved</td><td colspan="4">Wake Reason</td></tr><tr><td>4</td><td colspan="8">Reserved</td></tr><tr><td>5</td><td colspan="8">Reserved</td></tr><tr><td>6</td><td colspan="8">Reserved</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	1	0	2	Last Core C-State				Deepest Core C-State				3	Reserved				Wake Reason				4	Reserved								5	Reserved								6	Reserved							
	7	6	5	4	3	2	1	0																																																																								
0	0	0	0	0	0	0	1	0																																																																								
1	1	0	1	0	0	0	1	0																																																																								
2	Last Core C-State				Deepest Core C-State																																																																											
3	Reserved				Wake Reason																																																																											
4	Reserved																																																																															
5	Reserved																																																																															
6	Reserved																																																																															
Dependencies	TriggerEn && PwrEvtEn		Generation Scenario		Transition from a C-state deeper than C0.0 to C0.																																																																											
Description	Indicates processor return to thread C0 from a C-state deeper than C0.0. For return from some highly optimized C0 sub-C-states, such as C0.1, no PWRX packet is generated. The Last Core C-State field provides the MWAIT encoding for the core C-state at the time of the wake. The Deepest Core C-State provides the MWAIT encoding for the deepest core C-state achieved during the sleep session, or since leaving thread C0. MWAIT encodings for C-states can be found in Table 4-11 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B. Note that these values reflect only the core C-state, and hence will not exceed the maximum supported core C-state, even if deeper C-states can be requested. The Wake Reason field is one-hot, encoded as follows: <table><tr><td>Bit</td><td>Field</td><td>Meaning</td></tr><tr><td>0</td><td>Interrupt</td><td>Wake due to external interrupt received.</td></tr><tr><td>1</td><td>Timer Deadline</td><td>Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.</td></tr><tr><td>2</td><td>Store to Monitored Address</td><td>Wake due to store to monitored address.</td></tr><tr><td>3</td><td>HW Wake</td><td>Wake due to hardware autonomous condition, such as HDC.</td></tr></table>								Bit	Field	Meaning	0	Interrupt	Wake due to external interrupt received.	1	Timer Deadline	Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.	2	Store to Monitored Address	Wake due to store to monitored address.	3	HW Wake	Wake due to hardware autonomous condition, such as HDC.																																																									
Bit	Field	Meaning																																																																														
0	Interrupt	Wake due to external interrupt received.																																																																														
1	Timer Deadline	Wake due to timer expiration, such as UMWAIT/TPAUSE TSC-quanta.																																																																														
2	Store to Monitored Address	Wake due to store to monitored address.																																																																														
3	HW Wake	Wake due to hardware autonomous condition, such as HDC.																																																																														

Table 35-44. PWRX Packet Definition (Contd.)

Application	PWRX will always apply to the same IP as the PWRE. The time of wake can be discerned from (optional) timing packets that precede PWRX.

35.4.2.26 Block Begin Packet (BBP)**Table 35-45. Block Begin Packet Definition**

Name	BBP																																											
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td>SZ</td><td colspan="2">Reserved</td><td colspan="5">Type[4:0]</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0	1	1	2	SZ	Reserved		Type[4:0]				
	7	6	5	4	3	2	1	0																																				
0	0	0	0	0	0	0	1	0																																				
1	0	1	1	0	0	0	1	1																																				
2	SZ	Reserved		Type[4:0]																																								
Dependencies	TriggerEn		Generation Scenario	PEBS event, if IA32_PEBS_ENABLE.OUTPUT=1.																																								
Description	This packet indicates the beginning of a block of packets which are collectively tied to a single event or instruction. The size of the block item payloads within this block is provided by the Size (SZ) bit: SZ=0: 8-byte block items SZ=1: 4-byte block items The meaning of the BIP payloads is provided by the Type field: <table><tr><th>BBP.Type</th><th>Block name</th></tr><tr><td>0x00</td><td>Reserved</td></tr><tr><td>0x01</td><td>General-Purpose Registers</td></tr><tr><td>0x02..0x03</td><td>Reserved</td></tr><tr><td>0x04</td><td>PEBS Basic</td></tr><tr><td>0x05</td><td>PEBS Memory</td></tr><tr><td>0x06..0x07</td><td>Reserved</td></tr><tr><td>0x08</td><td>LBR Block 0</td></tr><tr><td>0x09</td><td>LBR Block 1</td></tr><tr><td>0x0A</td><td>LBR Block 2</td></tr><tr><td>0x0B..0x0F</td><td>Reserved</td></tr><tr><td>0x10</td><td>XMM Registers</td></tr><tr><td>0x11..0x1F</td><td>Reserved</td></tr></table>								BBP.Type	Block name	0x00	Reserved	0x01	General-Purpose Registers	0x02..0x03	Reserved	0x04	PEBS Basic	0x05	PEBS Memory	0x06..0x07	Reserved	0x08	LBR Block 0	0x09	LBR Block 1	0x0A	LBR Block 2	0x0B..0x0F	Reserved	0x10	XMM Registers	0x11..0x1F	Reserved										
BBP.Type	Block name																																											
0x00	Reserved																																											
0x01	General-Purpose Registers																																											
0x02..0x03	Reserved																																											
0x04	PEBS Basic																																											
0x05	PEBS Memory																																											
0x06..0x07	Reserved																																											
0x08	LBR Block 0																																											
0x09	LBR Block 1																																											
0x0A	LBR Block 2																																											
0x0B..0x0F	Reserved																																											
0x10	XMM Registers																																											
0x11..0x1F	Reserved																																											
Application	A BBP will always be followed by a Block End Packet (BEP), and when the block is generated while ContextEn=1 that BEP will have IP=1 and be followed by a FUP that provides the IP to which the block should be bound. Note that, in addition to BEP, a block can be terminated by a BBP (indicating the start of a new block) or an OVF packet.																																											

35.4.2.27 Block Item Packet (BIP)**Table 35-46. Block Item Packet Definition**

Name	BIP

Table 35-46. Block Item Packet Definition (Contd.)

Packet Format	If the preceding BBP.SZ=0:								
		7	6	5	4	3	2	1	0
	0	ID[5:0]					1	0	0
	1	Payload[7:0]							
	2	Payload[15:8]							
	3	Payload[23:16]							
	4	Payload[31:24]							
	5	Payload[39:32]							
	6	Payload[47:40]							
	7	Payload[55:48]							
	8	Payload[63:56]							
	If the preceding BBP.SZ=1:								
		7	6	5	4	3	2	1	0
	0	ID[5:0]					1	0	0
	1	Payload[7:0]							
	2	Payload[15:8]							
	3	Payload[23:16]							
	4	Payload[31:24]							
Dependencies	TriggerEn			Generation Scenario		See BBP.			
Description	The size of the BIP payload is determined by the Size field in the preceding BBP packet. The BIP header provides the ID value that, when combined with the Type field from the preceding BBP, uniquely identifies the state value held in the BIP payload. See Table 35-47 below for the complete list.								
Application	See BBP.								

BIP State Value Encodings

The table below provides the encoding values for all defined block items. State items that are larger than 8 bytes, such as XMM register values, are broken into multiple 8-byte components. BIP packets with Size=1 (4 byte payload) will provide only the lower 4 bytes of the associated state value.

Table 35-47. BIP Encodings

BBP.Type	BIP.ID	State Value
General-Purpose Registers		
0x01	0x00	R/EFLAGS
0x01	0x01	R/EIP
0x01	0x02	R/EAX
0x01	0x03	R/ECX
0x01	0x04	R/EDX
0x01	0x05	R/EBX
0x01	0x06	R/ESP

Table 35-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x01	0x07	R/EBP
0x01	0x08	R/ESI
0x01	0x09	R/EDI
0x01	0x0A	R8
0x01	0x0B	R9
0x01	0x0C	R10
0x01	0x0D	R11
0x01	0x0E	R12
0x01	0x0F	R13
0x01	0x10	R14
0x01	0x11	R15
PEBS Basic Info (Section 21.9.2.2.1)		
0x04	0x00	Instruction Pointer
0x04	0x01	Applicable Counters
0x04	0x02	Timestamp
PEBS Memory Info (Section 21.9.2.2.2)		
0x05	0x00	MemAccessAddress
0x05	0x01	MemAuxInfo
0x05	0x02	MemAccessLatency
0x05	0x03	TSXAuxInfo
LBR_0		
0x08	0x00	LBR[TOS-0]_FROM_IP
0x08	0x01	LBR[TOS-0]_TO_IP
0x08	0x02	LBR[TOS-0]_INFO
0x08	0x03	LBR[TOS-1]_FROM_IP
0x08	0x04	LBR[TOS-1]_TO_IP
0x08	0x05	LBR[TOS-1]_INFO
0x08	0x06	LBR[TOS-2]_FROM_IP
0x08	0x07	LBR[TOS-2]_TO_IP
0x08	0x08	LBR[TOS-2]_INFO
0x08	0x09	LBR[TOS-3]_FROM_IP
0x08	0x0A	LBR[TOS-3]_TO_IP
0x08	0x0B	LBR[TOS-3]_INFO
0x08	0x0C	LBR[TOS-4]_FROM_IP
0x08	0x0D	LBR[TOS-4]_TO_IP
0x08	0x0E	LBR[TOS-4]_INFO
0x08	0x0F	LBR[TOS-5]_FROM_IP
0x08	0x10	LBR[TOS-5]_TO_IP

Table 35-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x08	0x11	LBR[TOS-5]_INFO
0x08	0x12	LBR[TOS-6]_FROM_IP
0x08	0x13	LBR[TOS-6]_TO_IP
0x08	0x14	LBR[TOS-6]_INFO
0x08	0x15	LBR[TOS-7]_FROM_IP
0x08	0x16	LBR[TOS-7]_TO_IP
0x08	0x17	LBR[TOS-7]_INFO
0x08	0x18	LBR[TOS-8]_FROM_IP
0x08	0x19	LBR[TOS-8]_TO_IP
0x08	0x1A	LBR[TOS-8]_INFO
0x08	0x1B	LBR[TOS-9]_FROM_IP
0x08	0x1C	LBR[TOS-9]_TO_IP
0x08	0x1D	LBR[TOS-9]_INFO
0x08	0x1E	LBR[TOS-10]_FROM_IP
0x08	0x1F	LBR[TOS-10]_TO_IP
LBR_1		
0x09	0x00	LBR[TOS-10]_INFO
0x09	0x01	LBR[TOS-11]_FROM_IP
0x09	0x02	LBR[TOS-11]_TO_IP
0x09	0x03	LBR[TOS-11]_INFO
0x09	0x04	LBR[TOS-12]_FROM_IP
0x09	0x05	LBR[TOS-12]_TO_IP
0x09	0x06	LBR[TOS-12]_INFO
0x09	0x07	LBR[TOS-13]_FROM_IP
0x09	0x08	LBR[TOS-13]_TO_IP
0x09	0x09	LBR[TOS-13]_INFO
0x09	0x0A	LBR[TOS-14]_FROM_IP
0x09	0x0B	LBR[TOS-14]_TO_IP
0x09	0x0C	LBR[TOS-14]_INFO
0x09	0x0D	LBR[TOS-15]_FROM_IP
0x09	0x0E	LBR[TOS-15]_TO_IP
0x09	0x0F	LBR[TOS-15]_INFO
0x09	0x10	LBR[TOS-16]_FROM_IP
0x09	0x11	LBR[TOS-16]_TO_IP
0x09	0x12	LBR[TOS-16]_INFO
0x09	0x13	LBR[TOS-17]_FROM_IP
0x09	0x14	LBR[TOS-17]_TO_IP
0x09	0x15	LBR[TOS-17]_INFO

Table 35-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x09	0x16	LBR[TOS-18]_FROM_IP
0x09	0x17	LBR[TOS-18]_TO_IP
0x09	0x18	LBR[TOS-18]_INFO
0x09	0x19	LBR[TOS-19]_FROM_IP
0x09	0x1A	LBR[TOS-19]_TO_IP
0x09	0x1B	LBR[TOS-19]_INFO
0x09	0x1C	LBR[TOS-20]_FROM_IP
0x09	0x1D	LBR[TOS-20]_TO_IP
0x09	0x1E	LBR[TOS-20]_INFO
0x09	0x1F	LBR[TOS-21]_FROM_IP
LBR_2		
0x0A	0x00	LBR[TOS-21]_TO_IP
0x0A	0x01	LBR[TOS-21]_INFO
0x0A	0x02	LBR[TOS-22]_FROM_IP
0x0A	0x03	LBR[TOS-22]_TO_IP
0x0A	0x04	LBR[TOS-22]_INFO
0x0A	0x05	LBR[TOS-23]_FROM_IP
0x0A	0x06	LBR[TOS-23]_TO_IP
0x0A	0x07	LBR[TOS-23]_INFO
0x0A	0x08	LBR[TOS-24]_FROM_IP
0x0A	0x09	LBR[TOS-24]_TO_IP
0x0A	0x0A	LBR[TOS-24]_INFO
0x0A	0x0B	LBR[TOS-25]_FROM_IP
0x0A	0x0C	LBR[TOS-25]_TO_IP
0x0A	0x0D	LBR[TOS-25]_INFO
0x0A	0x0E	LBR[TOS-26]_FROM_IP
0x0A	0x0F	LBR[TOS-26]_TO_IP
0x0A	0x10	LBR[TOS-26]_INFO
0x0A	0x11	LBR[TOS-27]_FROM_IP
0x0A	0x12	LBR[TOS-27]_TO_IP
0x0A	0x13	LBR[TOS-27]_INFO
0x0A	0x14	LBR[TOS-28]_FROM_IP
0x0A	0x15	LBR[TOS-28]_TO_IP
0x0A	0x16	LBR[TOS-28]_INFO
0x0A	0x17	LBR[TOS-29]_FROM_IP
0x0A	0x18	LBR[TOS-29]_TO_IP
0x0A	0x19	LBR[TOS-29]_INFO
0x0A	0x1A	LBR[TOS-30]_FROM_IP

Table 35-47. BIP Encodings (Contd.)

BBP.Type	BIP.ID	State Value
0x0A	0x1B	LBR[TOS-30]_TO_IP
0x0A	0x1C	LBR[TOS-30]_INFO
0x0A	0x1D	LBR[TOS-31]_FROM_IP
0x0A	0x1E	LBR[TOS-31]_TO_IP
0x0A	0x1F	LBR[TOS-31]_INFO
XMM Registers		
0x10	0x00	XMM0_Q0
0x10	0x01	XMM0_Q1
0x10	0x02	XMM1_Q0
0x10	0x03	XMM1_Q1
0x10	0x04	XMM2_Q0
0x10	0x05	XMM2_Q1
0x10	0x06	XMM3_Q0
0x10	0x07	XMM3_Q1
0x10	0x08	XMM4_Q0
0x10	0x09	XMM4_Q1
0x10	0x0A	XMM5_Q0
0x10	0x0B	XMM5_Q1
0x10	0x0C	XMM6_Q0
0x10	0x0D	XMM6_Q1
0x10	0x0E	XMM7_Q0
0x10	0x0F	XMM7_Q1
0x10	0x10	XMM8_Q0
0x10	0x11	XMM8_Q1
0x10	0x12	XMM9_Q0
0x10	0x13	XMM9_Q1
0x10	0x14	XMM10_Q0
0x10	0x15	XMM10_Q1
0x10	0x16	XMM11_Q0
0x10	0x17	XMM11_Q1
0x10	0x18	XMM12_Q0
0x10	0x19	XMM12_Q1
0x10	0x1A	XMM13_Q0
0x10	0x1B	XMM13_Q1
0x10	0x1C	XMM14_Q0
0x10	0x1D	XMM14_Q1
0x10	0x1E	XMM15_Q0
0x10	0x1F	XMM15_Q1

35.4.2.28 Block End Packet (BEP)

Table 35-48. Block End Packet Definition

Name	BEP																																		
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>IP</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	IP	0	1	1	0	0	1	1
	7	6	5	4	3	2	1	0																											
0	0	0	0	0	0	0	1	0																											
1	IP	0	1	1	0	0	1	1																											
Dependencies	TriggerEn		Generation Scenario		See BBP.																														
Description	Indicates the end of a packet block. The IP bit indicates if a FUP will follow, and will be set if ContextEn=1.																																		
Application	The block, from initial BBP to the BEP, binds to the FUP IP, if IP=1, and consumes the FUP.																																		

35.4.2.29 Control Flow Event (CFE) Packet

Table 35-49. Control Flow Event Packet Definition

Name	CFE																																																				
Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td>IP</td><td colspan="2">Reserved</td><td colspan="5">Type[4:0]</td></tr><tr><td>3</td><td colspan="8">Vector[7:0]</td></tr></table> IP bit indicates if a stand-alone FUP will follow.									7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	2	IP	Reserved		Type[4:0]					3	Vector[7:0]							
	7	6	5	4	3	2	1	0																																													
0	0	0	0	0	0	0	1	0																																													
1	0	0	0	1	0	0	1	1																																													
2	IP	Reserved		Type[4:0]																																																	
3	Vector[7:0]																																																				
Dependencies	IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn On ContextEn transitions, the CFE will be generated regardless of direction (1→0 or 0→1). VM exit is an exception, where CFE.VMEXIT depends only on the prior value of ContextEn.			Generation Scenario	Software interrupt, external interrupt, user interrupt, or exception, including those injected on VM entry. INIT, SIPI, SMI, RSM, IRET, Shutdown. VM exit, if “Conceal VMX in PT” VMCS exit control is 0. VM entry, if “Conceal VMX in PT” VMCS entry control is 0. TSX Abort.																																																
Description	This packet indicates that an asynchronous event or related event (see list above) has occurred. The type of event is provided in the packet (see Table 35-50 below), and, if the IP bit is set, the IP at which the event occurred is provided in a stand-alone FUP packet that follows. Further, in the case of an interrupt or exception, the vector field provides the vector of the event. The IP bit will be set only when ContextEn=1 before the event is taken, and either BranchEn=0 or else no FUP is generated for this event by BranchEn=1. There are some cases, such as SIPI and RSM, where no FUP is generated. Note that events that are not delivered to software, such as nested events or events which cause a VM exit, do not generate CFE packets.																																																				
Application	If the IP bit is set, a FUP will follow that is stand-alone (not part of a compound packet event), and the CFE consumes the FUP. If the IP bit is not set, the CFE binds to the next FUP if PacketEn=1 (hence the CFE comes after a TIP.PGE but before the next TIP.PGD), and is stand-alone if PacketEn=0.																																																				

CFE Packet Type and Vector Fields

Every CFE has a Type field, which provides the type of event which generated the packet. For a subset of CFE Types, the CFE.Vector field may be valid. Details on these fields, as well as the IP to be expected in any following FUP packet, are provided in the table below.

Table 35-50. CFE Packet Type and Vector Fields Details

CFE Subtype	Type	Vector	FUP IP	Details
INTR	0x1	Event Vector	Varies	Used for interrupts (external and software), Exceptions, Faults, and NMI. FUP contains that address of the instruction that has not completed (NLIP for trap events, CLIP for fault events).
IRET	0x2	Invalid	CLIP	
SMI	0x3	Invalid	NLIP	
RSM	0x4	Invalid	None	
SIPI	0x5	SIPI Vector	None	
INIT	0x6	Invalid	NLIP	
VMENTRY	0x7	Invalid	CLIP	FUP contains IP of VMLAUNCH/VMRESUME.
VMEXIT	0x8	Invalid	Varies	FUP IP varies depending on type of VM exit, but will be the address of the instruction that has not completed. Will be consistent with Guest IP saved in VMCS.
VMEXIT_INTR	0x9	Event Vector	Varies	Sent in cases where VM exit was caused by an INTR event (interrupt, exception, fault, or NMI). Vector provided is for the event which caused the VM exit. FUP IP behavior matches that of INTR type above.
SHUTDOWN	0xa	Invalid	Varies	FUP IP varies depending on the type of event that caused shutdown, but will be the address of the instruction that has not completed.
Reserved	0xb	N/A	N/A	
UINTR	0xc	User Interrupt Vector	NLIP	User interrupt delivered.
UIRET	0xd	Invalid	CLIP	Exiting from user interrupt routine.
Reserved	0xe...0xf	N/A	N/A	Reserved

35.4.2.30 Event Data (EVD) Packet

Table 35-51. Event Data Packet Definition

Name	EVD

Table 35-51. Event Data Packet Definition (Contd.)

Packet Format	<table><tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>2</td><td colspan="2">Reserved</td><td colspan="6">Type[5:0]</td></tr><tr><td>3</td><td colspan="8">Payload[7:0]</td></tr><tr><td>4</td><td colspan="8">Payload[15:8]</td></tr><tr><td>5</td><td colspan="8">Payload[23:16]</td></tr><tr><td>6</td><td colspan="8">Payload[31:24]</td></tr><tr><td>7</td><td colspan="8">Payload[39:32]</td></tr><tr><td>8</td><td colspan="8">Payload[47:40]</td></tr><tr><td>9</td><td colspan="8">Payload[55:48]</td></tr><tr><td>10</td><td colspan="8">Payload[63:56]</td></tr></table>										7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	1	2	Reserved		Type[5:0]						3	Payload[7:0]								4	Payload[15:8]								5	Payload[23:16]								6	Payload[31:24]								7	Payload[39:32]								8	Payload[47:40]								9	Payload[55:48]								10	Payload[63:56]							
	7	6	5	4	3	2	1	0																																																																																																													
0	0	0	0	0	0	0	1	0																																																																																																													
1	0	1	0	1	0	0	1	1																																																																																																													
2	Reserved		Type[5:0]																																																																																																																		
3	Payload[7:0]																																																																																																																				
4	Payload[15:8]																																																																																																																				
5	Payload[23:16]																																																																																																																				
6	Payload[31:24]																																																																																																																				
7	Payload[39:32]																																																																																																																				
8	Payload[47:40]																																																																																																																				
9	Payload[55:48]																																																																																																																				
10	Payload[63:56]																																																																																																																				
Dependencies	IA32_RTIT_CTL.EventEn && TriggerEn && ContextEn		Generation Scenario	Page fault, including those injected on VM entry. VM exit, if “Suppress VMX packets on exit” VMCS exit control is 0.																																																																																																																	
Description	Provides additional data about the event that caused the following CFE. The Payload field is dictated by the Type. <table><tr><th>Type</th><th>Payload</th></tr><tr><td>'000000</td><td>Page Fault Linear Address, same as CR2 (PFA)</td></tr><tr><td>'000001</td><td>VMX Exit Qualification (VMXQ)</td></tr><tr><td>'000010</td><td>VMX Exit Reason (VMXR)</td></tr><tr><td>'000011 - '111111</td><td>Reserved</td></tr></table> EVD packets are never generated in cases where a CFE is not.								Type	Payload	'000000	Page Fault Linear Address, same as CR2 (PFA)	'000001	VMX Exit Qualification (VMXQ)	'000010	VMX Exit Reason (VMXR)	'000011 - '111111	Reserved																																																																																																			
Type	Payload																																																																																																																				
'000000	Page Fault Linear Address, same as CR2 (PFA)																																																																																																																				
'000001	VMX Exit Qualification (VMXQ)																																																																																																																				
'000010	VMX Exit Reason (VMXR)																																																																																																																				
'000011 - '111111	Reserved																																																																																																																				
Application	EVD packets bind to the same IP (if any) as the subsequent CFE packet.																																																																																																																				

35.5 TRACING IN VMX OPERATION

On processors that IA32_VMX_MISC[bit 14] reports 1, TraceEn can be set in VMX operation. The VMM can configure specific VMX controls to control what virtualization-specific data is included within the trace packets (see Section 35.5.1 for details). The VMM can also configure the VMCS to limit tracing to non-root operation, or to trace across both root and non-root operation. The VMCS controls exist to simplify virtualization of Intel PT for guest use, including the “Clear IA32_RTIT_CTL” exit control (See Section 26.7.1), “Load IA32_RTIT_CTL” entry control (See Section 26.8.1), and “Intel PT uses guest physical addresses” execution control (See Section 27.5.3).

For older processors that do not support these VMCS controls, the MSR-load areas used by VMX transitions can be employed by the VMM to restrict tracing to the desired context. See Section 35.5.2 for details. Tracing with SMM Transfer Monitor is described in Section 35.6.

35.5.1 VMX-Specific Packets and VMCS Controls

In all of the usages of VMX and Intel PT, a decoder in the host or VMM context can identify the occurrences of VMX transitions with the aid of VMX-specific packets. There are four kinds of packets relevant to VMX:

- **VMCS packet.** The VMX transitions of individual VMs can be distinguished by a decoder using the VMCS-pointer field in a VMCS packet. A VMCS packet is sent on a successful execution of VMPTRLD, and its VMCS-pointer field stores the VMCS pointer loaded by that execution. See Section 35.4.2.15 for details.

- **The NR (non-root) bit in a PIP packet.** Normally, the NR bit is set in any PIP packet generated in VMX non-root operation. In addition, PIP packets are generated with each VM entry and VM exit. Thus a transition of the NR bit from 0 to 1 indicates the occurrence of a VM entry, and a transition of 1 to 0 indicates the occurrence of a VM exit.
- **CFE packet.** Identifies VM exit and VM entry operations.
- **EVD packet.** Provides the exit reason and exit qualification for VM exits.

There are VMX controls that a VMM can set to conceal some of this VMX-specific information (by suppressing its recording) and thereby prevent it from leaking across virtualization boundaries. There is one of these controls (each of which is called “conceal VMX from PT”) of each type of VMX control.

Table 35-52. VMX Controls For Intel Processor Trace

Type of VMX Control	Bit Position ¹	Value	Behavior
Secondary processor-based VM-execution control	19	0	Each PIP generated in VM non-root operation will set the NR bit. PSB+ in VMX non-root operation will include the VMCS packet, to ensure that the decoder knows which guest is currently in use.
		1	Each PIP generated in VMX non-root operation will clear the NR bit. PSB+ in VMX non-root operation will not include the VMCS packet.
VM-exit control	24	0	Each VM exit generates a PIP in which the NR bit is clear, and a CFE/EVD if Event Trace is enabled. In addition, SMM VM exits generate VMCS packets.
		1	VM exits do not generate PIPs, CFEs, or EVDs, and no VMCS packets are generated on SMM VM exits.
VM-entry control	17	0	Each VM entry generates a PIP in which the NR bit is set (except VM entries that return from SMM to VMX root operation), and a CFE if Event Trace is enabled. In addition, VM entries that return from SMM generate VMCS packets.
		1	VM entries do not generate PIPs or CFEs, and no VMCS packets are generated on VM entries that return from SMM.

NOTES:

1. These are the positions of the control bits in the relevant VMX control fields.

The 0-settings of these VMX controls enable all VMX-specific packet information. The scenarios that would use these default settings also do not require the VMM to use VMX MSR-load areas to enable and disable trace-packet generation across VMX transitions.

If IA32_VMX_MISC[bit 14] reports 0, the 1-settings of the VMX controls in Table 35-52 are not supported, and VM entry will fail on any attempt to set them.

35.5.2 Managing Trace Packet Generation Across VMX Transitions

In tracing scenarios that collect packets for both VMX root operation and VMX non-root operation, a host executive can manage the MSRs associated with trace packet generation directly. The states of these MSRs need not be modified across VMX transitions.

For tracing scenarios that collect packets only within VMX root operation or only within VMX non-root operation, the VMM can toggle IA32_RTIT_CTL.TraceEn on VMX transitions.

35.5.2.1 System-Wide Tracing

When a host or VMM configures Intel PT to collect trace packets of the entire system, it can leave the relevant VMX controls clear to allow VMX-specific packets to provide information across VMX transitions.

The decoder will desire to identify the occurrence of VMX transitions. The packets of interests to a decoder are shown in Table 35-53.

Table 35-53. Packets on VMX Transitions (System-Wide Tracing)

Event	Packets	Enable	Description
VM exit	EVD.VMXR, EVD.VMXQ, CFE.VMEXIT*	EventEn	The CFE identifies the transfer as a VM exit, while the associated EVDs provide the exit reason and exit qualification.
	FUP(GuestIP)	BranchEn or EventEn	The FUP indicates at which point in the guest flow the VM exit occurred. This is important, since VM exit can be an asynchronous event. The IP will match that written into the VMCS.
	PIP(HostCR3, NR=0)		The PIP packet provides the new host CR3 value, as well as indication that the logical processor is entering VMX root operation. This allows the decoder to identify the change of executing context from guest to host and load the appropriate set of binaries to continue decode.
	TIP(HostIP)	BranchEn	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX root operation. Note, this packet could be preceded by a MODE.Exec packet (Section 35.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.
VM entry	CFEVMENTRY, FUP(CLIP)	EventEn	The CFE identifies the transfer as a VM entry, while the FUP identifies the VMLAUNCH/VMRESUME IP.
	PIP(GuestCR3, NR=1)	BranchEn	The PIP packet provides the new guest CR3 value, as well as indication that the logical processor is entering VMX non-root operation. This allows the decoder to identify the change of executing context from host to guest and load the appropriate set of binaries to continue decode.
	TIP(GuestIP)	BranchEn	The TIP indicates the destination IP, the IP of the first instruction to be executed in VMX non-root operation. This should match the RIP loaded from the VMCS. Note, this packet could be preceded by a MODE.Exec packet (Section 35.4.2.8). This is generated only in cases where CS.D or (CS.L & EFER.LMA) change during the transition.

Since the VMX controls that suppress packet generation are cleared, a VMCS packet will be included in all PSB+ for this usage scenario. Additionally, VMPTRLD will generate such a packet. Thus the decoder can distinguish the execution context of different VMs.

When the host VMM configures a system to collect trace packets in this scenario, it should emulate CPUID to report CPUID.07H.00H:EBX[26] as 0 to guests, indicating to guests that Intel PT is not available.

VMX TSC Manipulation

The TSC packets generated while in VMX non-root operation will include any changes resulting from the use of a VMM's use of the TSC offsetting or TSC scaling VMX controls (see Chapter 27, "VMX Non-Root Operation"). In this system-wide usage model, the decoder may need to account for the effect of per-VM adjustments in the TSC packets generated in VMX non-root operation and the absence of TSC adjustments in TSC packets generated in VMX root operation. The VMM can supply this information to the decoder.

35.5.2.2 Guest-Only Tracing

A VMM can configure trace-packet generation while in VMX non-root operation for guests executing normally. This is accomplished by utilizing VMCS controls to manipulate the guest IA32_RTIT_CTL value on VMX transitions. For older processors that do not support these VMCS controls, a VMM can use the VMX MSR-load areas on VM exits (see Section 26.7.2, "VM-Exit Controls for MSRs") and VM entries (see Section 26.8.2, "VM-Entry Controls for MSRs") to limit trace-packet generation to the guest environment.

For this usage, VM entry is programmed to enable trace packet generation, while VM exit is programmed to clear IA32_RTIT_CTL.TraceEn so as to disable trace-packet generation in the host. Further, if it is preferred that the

guest packet stream contain no indication that execution was in VMX non-root operation, the VMM should set to 1 all the VMX controls enumerated in Table 35-52.

35.5.2.3 Emulation of Intel PT Traced State

If a VMM emulates an element of processor state by taking a VM exit on reads and/or writes to that piece of state, and the state element impacts Intel PT packet generation or values, it may be incumbent upon the VMM to insert or modify the output trace data.

If a VM exit is taken on a guest write to CR3 (including “MOV CR3” as well as task switches), the PIP packet normally generated on the CR3 write will be missing.

To avoid decoder confusion when the guest trace is decoded, the VMM should emulate the missing PIP by writing it into the guest output buffer. If the guest CR3 value is manipulated, the VMM may also need to manipulate the IA32_RTIT_CR3_MATCH value, in order to ensure the trace behavior matches the guest's expectation.

Similarly, if a VMM emulates the TSC value by taking a VM exit on RDTSC, the TSC packets generated in the trace may mismatch the TSC values returned by the VMM on RDTSC. To ensure that the trace can be properly aligned with software logs based on RDTSC, the VMM should either make corresponding modifications to the TSC packet values in the guest trace, or use mechanisms such as TSC offsetting or TSC scaling in place of exiting.

35.5.2.4 TSC Scaling

When TSC scaling is enabled for a guest using Intel PT, the VMM should ensure that the value of Maximum Non-Turbo Ratio[15:8] in MSR_PLATFORM_INFO (MSR 0CEH) and the TSC/“core crystal clock” ratio (EBX/EAX) in CPUID.15H are set in a manner consistent with the resulting TSC rate that will be visible to the VM. This will allow the decoder to properly apply TSC packets, MTC packets (based on the core crystal clock or ART, whose frequency is indicated by CPUID.15H), and CBR packets (which indicate the ratio of the processor frequency to the Max Non-Turbo frequency). Absent this, or separate indication of the scaling factor, the decoder will be unable to properly track time in the trace. See Section 35.8.3 for details on tracking time within an Intel PT trace.

35.5.2.5 Failed VM Entry

The packets generated by a failed VM entry depend both on the VMCS configuration, as well as on the type of failure. The results to expect are summarized in the table below. Note that packets in *italics* may or may not be generated, depending on implementation choice, and the point of failure.

Table 35-54. Packets on a Failed VM Entry

Usage Model	Entry Configuration	Early Failure (fall through to next IP)	Late Failure (VM exit like)
System-Wide	No use of “Load IA32_RTIT_CTL” entry control or VM-entry MSR-load area	TIP (NextIP)	<i>CFE.VMENTRY, FUP(CLIP) if EventEn=1</i> <i>PIP(Guest CR3, NR=1), TraceEn 0→1 Packets (See Section 35.2.8.3), PIP(HostCR3, NR=0), TIP(HostIP)</i>
VMM Only	“Load IA32_RTIT_CTL” entry control or VM-entry MSR-load area used to clear TraceEn	TIP (NextIP)	<i>TraceEn 0→1 Packets (See Section 35.2.8.3), TIP(HostIP)</i>
VM Only	“Load IA32_RTIT_CTL” entry control or VM-entry MSR-load area used to set TraceEn	None	None

35.5.2.6 VMX Abort

VMX abort conditions take the processor into a shutdown state. On a VM exit that leads to VMX abort, some packets (FUP, PIP) may be generated, but any expected TIP, TIP.PGE, or TIP.PGD may be dropped.

35.6 TRACING AND SMM TRANSFER MONITOR (STM)

The SMM-transfer monitor (STM) is a VMM that operates inside SMM while in VMX root operation. An STM operates in conjunction with an executive monitor. The latter operates outside SMM and in VMX root operation. Transitions from the executive monitor or its VMs to the STM are called SMM VM exits. The STM returns from SMM via a VM entry to the VM in VMX non-root operation or the executive monitor in VMX root operation.

Intel PT supports tracing in an STM similar to tracing support for VMX operation as described above in Section 35.5. As a result, on a SMM VM exit resulting from #SMI, TraceEn is neither saved nor cleared by default. Software can save the state of the trace configuration MSRs and clear TraceEn using the MSR load/save lists.

Within Event Trace, SMM VM exits generate packets indicating both an #SMI and a VM exit. Similarly, VM entries that return from SMM generate packets that indicate both an RSM and a VM entry. SMM VM exits initiated by the VMCALL instruction do not generate any CFE packet, though the subsequent VM entry returning from SMM will generate a CFE.RSM.

35.7 PACKET GENERATION SCENARIOS

The following tables provides examples of packet generation for various operations. The following acronyms are used in the packet examples below:

- CLIP - Current LIP
- NLIP - Next Sequential LIP
- BLIP - Branch Target LIP

Table 35-55 illustrates the packets generated by a series of example operations, assuming that PacketEn (TriggerEn && ContextEn && FilterEn && BranchEn) is set before and after the operation.

Table 35-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
1	Normal non-jump operation		None
2	Conditional branch	6th branch in internal TNT buffer	TNT
3	Conditional branch	1 st ..5 th branch in internal TNT buffer	None
4	Near indirect JMP or CALL		TIP(BLIP)
5	Direct near JMP or CALL		None
6	Near RET	Uncompressed	TIP(BLIP)
7	Near RET	Compressed, 6th branch in internal TNT buffer	TNT
8	Far Branch	Assumes no update to CR3, CS.L, or CS.D	TIP(BLIP)
9	Far Branch	Assumes update to CR3	PIP(NewCR3), TIP(BLIP)
10	Far Branch	Assumes update to CR3 and CS.D/CS.L	PIP(NewCR3), MODE.Exec, TIP(BLIP)
11	External Interrupt or NMI	Assumes no update to CR3, CS.D, or CS.L	FUP(NLIP), TIP(BLIP)
12	External Interrupt or NMI	Assumes update to CR3 and CS.D/CS.L	FUP(NLIP), PIP(NewCR3), MODE.Exec, TIP(BLIP)
13	Exception/Fault or Software Interrupt	Assumes no update to CR3, CS.D, or CS.L	FUP(CLIP), TIP(BLIP)
14	MOV to CR3		PIP(NewCR3, NR)
15	VM exit	Assumes system-wide tracing, see Section 35.5.2.1	See Table 35-53
16	VM entry	Assumes system-wide tracing, see Section 35.5.2.1	See Table 35-53

Table 35-55. Packet Generation under Different Example Operations

Case	Operation	Details	Packets
17	ENCLU[EENTER] / ENCLU[ERESUME] / ENCLU[EEXIT] / AEX/EEE	Only debug enclaves allow PacketEn to be set during enclave execution. Assumes no change to CS.L or CS.D.	FUP(CLIP), TIP(BLIP)
18	XBEGIN/XACQUIRE/XEND/XRELEASE	Does not begin/end transactional execution	None
19	XBEGIN/XACQUIRE	Assumes beginning of transactional execution	MODE.TSX(InTX=1, TXAbort=0), FUP(CLIP)
20	XEND/XRELEASE	Completes transaction	MODE.TSX(InTX=0, TXAbort=0), FUP(CLIP)
21	XABORT or Asynchronous Abort	Aborts transactional execution	MODE.TSX(InTX=0, TXAbort=1), FUP(CLIP), TIP(BLIP)
22	INIT	On BSP. Assumes no CR3, CS.D, or CS.L update.	FUP(NLIP), TIP(ResetLIP)
23	INIT	On AP, goes to wait-for-SIPI. Assumes no CR3 update.	FUP(NLIP)
24	SIPI	Assumes no CS.D or CS.L update	TIP.PGE(SIPI.LIP)
25	Wake from state deeper than C0.1, P-state change, or other scenario where timing packets (MTC, CYC) may have ceased.	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR
26	UINTR	User interrupt handler entry.	FUP(NLIP)
27	UIRET	Exiting from user interrupt handler.	FUP(NLIP)

Table 35-56 illustrates the packets generated in example scenarios where the operation alters the value of PacketEn. Note that insertion of PSB+ is not included here, though it can be coincident with initial enabling of Intel PT. See Section 35.3.7 for details.

Table 35-56. Packet Generation with Operations That Alter the Value of PacketEn

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
1	WRMSR/XRSTORS that changes TraceEn 0 → 1	0	1	0	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec
2	WRMSR/XRSTORS that changes TraceEn 0 → 1	0	1	1	TSC if TSCEn=1; TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, MODE.Exec, TIP.PGE(NLIP)
3	WRMSR that changes TraceEn 1 → 0	1	0	D.C.		FUP(CLIP), TIP.PGD()
4	Taken Branch	1	0	1	Source is in IP filter region. Target is outside IP filter region.	TIP.PGD(BLIP)
5	Taken Branch, Interrupt, EEXIT, etc.	0	1	1	Source is outside IP filter region. Target is in IP filter region.	TIP.PGE(BLIP)
6	Far Branch, Interrupt, EENTER, etc.	1	0	0	Requires change to CPL or CR3, or entry to opt-out enclave.	TIP.PGD()
7	Trap-like event (external interrupt, NMI, VM exit/entry, etc.)	1	0	0	Requires change to CPL or CR3.	FUP(NLIP), TIP.PGD()
8	Fault-like event (exception/fault, software interrupt, VM exit/entry, etc.)	1	0	0	Requires change to CPL or CR3.	FUP(CLIP), TIP.PGD()

Table 35-56. Packet Generation with Operations That Alter the Value of PacketEn (Contd.)

Case	Operation	PktEn Before	PktEn After	CntxEn After	Details	Packets
9	SMI, VM exit/entry	1	0	0	TraceEn is cleared.	FUP(NLIP), TIP.PGD()
10	RSM, VM exit/entry	0	1	1	TraceEn is set.	See Case 2 for packets on enable. FUP/TIP.PGE IP is the BLIP.
11	VM Exit	1	0	0	Assumes guest-only tracing, see Section 35.5.2.2. TraceEn is cleared.	FUP(VMCSg.RIP), TIP.PGD()
12	VM entry	0	1	1	Assumes guest-only tracing, see Section 35.5.2.2. TraceEn is set.	TIP.PGE(VMCSg.RIP)

Table 35-57 illustrates examples of PTWRITE, assuming TriggerEn & PTWEn is true.

Table 35-57. Examples of PTWRITE when TriggerEn & PTWEn is True

Case	Operation	ContextEn	Details	Packets
1	MWAIT/UMWAIT gets fault or VM exit.	D.C.		None. Other trace sources may generate packets on fault or VM exit.
2	MWAIT/UMWAIT requests C0, or monitor not armed, or VMX virtual-interrupt delivery.	D.C.		None.
3	MWAIT/UMWAIT enters C-state deeper than C0.1.	0		PWRE(Cx), EXSTOP
4	MWAIT/UMWAIT enters C-state deeper than C0.1.	1		MWAIT(Cy), PWRE(Cx), EXSTOP(IP), FUP(CLIP)
5	HLT, Triple-fault shutdown, other operation that enters C1.	1		PWRE(C1), EXSTOP(IP), FUP(CLIP)
6	Hardware Duty Cycling (HDC).	1	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	PWRE(HW, C6), EXSTOP(IP), FUP(NLIP), TSC?, TMA?, CBR, PWRX(CC6, CC6, 0x8)
7	Wake event during Cx (x > 0).	D.C.	TSC if TSCEn=1 TMA if TSCEn=MTCEn=1	TSC?, TMA?, CBR, PWRX(LCC, DCC, 0x1) Other trace sources may generate packets for the wake operation (e.g., interrupt).

Table 35-58 illustrates examples of Power Event Trace, assuming TriggerEn & PwrEvtEn is true.

Table 35-58. Examples of Power Event Trace when TriggerEn & PwrEvtEn is True

Case	Operation	ContextEn & FilterEn	Details	Packets
1	PTWRITE rm32/64	0		None
2	PTWRITE rm32	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 4B, rm32_value), FUP(CLIP)?
3	PTWRITE rm64	1	FUP, PTW.IP=1 if FUPonPTW=1	PTW(IP=1?, 8B, rm64_value), FUP(CLIP)?

Table 35-59 illustrates examples of Event Trace, assuming TriggerEn && ContextEn && EventEn is true. In all cases, other trace sources (e.g., BranchEn), if enabled, may generate additional packets. For details, see the other tables in this section.

Table 35-59. Event Trace Examples when TriggerEn && ContextEn && EventEn is True

Case	Operation	ContextEn Before	ContextEn After	Details	Packets
1	IRET	1	D.C.		CFE.IRET(IP=1), FUP(CLIP)
2	IRET	0	1		CFE.IRET
3	External interrupt, including NMI	1	D.C.		CFE.INTR(IP=1, Vector), FUP(NLIP)
4	External interrupt, including NMI	1	1	Assumes BranchEn=1, illustrates the shared FUP.	CFE.INTR(IP=0, Vector), FUP(NLIP), TIP(BLIP)
5	SW Interrupt, Exception/Fault other than #PF	1	D.C.		CFE.INTR(IP=1, Vector), FUP(CLIP)
6	Page Fault (#PF)	1	D.C.		EVD.PFA, CFE.INTR(IP=1,14), FUP(CLIP)
7	Page Fault (#PF)	0	D.C.		None
10	SMI	1	D.C.		CFE.SMI(IP=1), FUP(NLIP)
11	RSM, TraceEn restored to 1	D.C.	1		CFE.RSM(IP=0)
12	Entry to Shutdown	1	D.C.		CFE.SHUTDOWN(IP=1), FUP(CLIP)
13	VM exit caused by interrupt, fault, or SMI	1	D.C.	Assumes "Conceal VMX in PT" exit control is 0.	EVD.VMXQ, EVD.VMXR, CFE.VMEXIT_INTR(IP=1, Vector), FUP(VMCSSg.LIP)
14	VM exit caused by other than interrupt, fault, or SMI	1	D.C.	Assumes "Conceal VMX in PT" exit control is 0.	EVD.VMXQ, EVD.VMXR, CFE.VMEXIT(IP=1), FUP(VMCSSg.LIP)
15	VM exit caused by other than interrupt, fault, or SMI	0	1	Assumes "Conceal VMX in PT" exit control is 0.	CFE.VMEXIT(IP=0)
16	VM entry	1	D.C.	Assumes "Conceal VMX in PT" entry control is 0.	CFE.VMENTRY(IP=1), FUP(VMCSSh.LIP)
17	AEX/EEE, from opt-out (non-debug) enclave	0	0		None
18	AEX/EEE, from opt-out (non-debug) enclave	0	1		CFE.INTR(IP=0)
19	AEX, from opt-in (debug) enclave	1	D.C.		CFE.INTR(IP=1, Vec), FUP(AEP LIP)
20	INIT	1	D.C.		CFE.INIT(IP=1), FUP(NLIP)
21	SIPI	1	D.C.		CFE.SIPI(IP=0)
22	STI/CLI/POPF	1	1	Assumes a change to RFLAGS.IF.	MODE.Exec, FUP(NLIP)

35.8 SOFTWARE CONSIDERATIONS

35.8.1 Tracing SMM Code

Nothing prevents an SMM handler from configuring and enabling packet generation for its own use. As described in Section 35.2.9.3, SMI will always clear TraceEn, so the SMM handler would have to set TraceEn in order to enable tracing. There are some unique aspects and guidelines involved with tracing SMM code, which follow:

1. SMM should save away the existing values of any configuration MSRs that SMM intends to modify for tracing. This will allow the non-SMM tracing context to be restored before RSM.
2. It is recommended that SMM wait until it sets CSbase to 0 before enabling packet generation, to avoid possible LIP vs RIP confusion.
3. Packet output cannot be directed to SMRR memory, even while tracing in SMM.
4. Before performing RSM, SMM should take care to restore modified configuration MSRs to the values they had immediately after #SMI. This involves first disabling packet generation by clearing TraceEn, then restoring any other configuration MSRs that were modified.
5. RSM
 - Software must ensure that TraceEn=0 at the time of RSM. Tracing RSM is not a supported usage model, and the packets generated by RSM are undefined.
 - For processors on which Intel PT and LBR use are mutually exclusive (see Section 35.3.1.2), any RSM during which TraceEn is restored to 1 will suspend any LBR or BTS logging.

35.8.2 Cooperative Transition of Multiple Trace Collection Agents

A third-party trace-collection tool should take into consideration the fact that it may be deployed on a processor that supports Intel PT but may run under any operating system.

In such a deployment scenario, Intel recommends that tool agents follow similar principles of cooperative transition of single-use hardware resources, similar to how performance monitoring tools handle performance monitoring hardware:

- Respect the “in-use” ownership of an agent who already configured the trace configuration MSRs, see architectural MSRs with the prefix “IA32_RTIT_” in Chapter 2, “Model-Specific Registers (MSRs),” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4, where “in-use” can be determined by reading the “enable bits” in the configuration MSRs.
- Relinquish ownership of the trace configuration MSRs by clearing the “enabled bits” of those configuration MSRs.

35.8.3 Tracking Time

This section describes the relationships of several clock counters whose update frequencies reside in different domains that feed into the timing packets. To track time, the decoder also needs to know the regularity or irregularity of the occurrences of various timing packets that store those clock counters.

Intel PT provides time information for three different but related domains:

- Processor timestamp counter

This counter increments at the max non-turbo or P1 frequency, and its value is returned on a RDTSC. Its frequency is fixed. The TSC packet holds the lower 7 bytes of the timestamp counter value. The TSC packet occurs occasionally and are much less frequent than the frequency of the time stamp counter. The timestamp counter will continue to increment when the processor is in deep C-States, with the exception of processors reporting CPUID.80000007H:EDX.TSC_INVARIANT[8] = 0.
- Core crystal clock

The ratio of the core crystal clock to timestamp counter frequency is known as P, and can be calculated as CPUID.15H:EBX[31:0] / CPUID.15H:EAX[31:0]. The frequency of the core crystal clock is fixed and lower than that of the timestamp counter. The periodic MTC packet is generated based on software-selected

multiples of the crystal clock frequency. The MTC packet is expected to occur more frequently than the TSC packet.

- Processor core clock

The processor core clock frequency can vary due to P-state and thermal conditions. The CYC packet provides elapsed time as measured in processor core clock cycles relative to the last CYC packet.

A decoder can use all or some combination of these packets to track time at different resolutions throughout the trace packets.

35.8.3.1 Time Domain Relationships

The three domains are related by the following formula:

$$\text{TimeStampValue} = (\text{CoreCrystalClockValue} * P) + \text{AdjustedProcessorCycles} + \text{Software\_Offset};$$

The CoreCrystalClockValue, also known as the Always Running Timer (ART) value, can provide the coarse-grained component of the TSC value. P, or the TSC/ART ratio, can be derived from CPUID.15H, as described in Section 35.8.3.

The AdjustedProcessorCycles component provides the fine-grained distance from the rising edge of the last core crystal clock. Specifically, it is a cycle count in the same frequency as the timestamp counter from the last crystal clock rising edge. The value is adjusted based on the ratio of the processor core clock frequency to the Maximum Non-Turbo (or P1) frequency.

The Software_Offsets component includes software offsets that are factored into the timestamp value, such as IA32_TSC_ADJUST.

35.8.3.2 Estimating TSC within Intel PT

For many usages, it may be useful to have an estimated timestamp value for all points in the trace. The formula provided in Section 35.8.3.1 above provides the framework for how such an estimate can be calculated from the various timing packets present in the trace.

The TSC packet provides the precise timestamp value at the time it is generated; however, TSC packets are infrequent, and estimates of the current timestamp value based purely on TSC packets are likely to be very inaccurate for this reason. In order to get more precise timing information between TSC packets, CYC packets and/or MTC packets should be enabled.

MTC packets provide incremental updates of the CoreCrystalClockValue. On processors that support CPUID.15H, the frequency of the timestamp counter and the core crystal clock is fixed, thus MTC packets provide a means to update the running timestamp estimate. Between two MTC packets A and B, the number of crystal clock cycles passed is calculated from the 8-bit payloads of respective MTC packets:

$$(\text{CTC}_B - \text{CTC}_A), \text{ where } \text{CTC}_i = \text{MTC}_i[15:8] \ll \text{IA32\_RTIT\_CTL.MTCFreq} \text{ and } i = A, B.$$

The time from a TSC packet to the subsequent MTC packet can be calculated using the TMA packet that follows the TSC packet. The TMA packet provides both the crystal clock value (lower 16 bits, in the CTC field) and the AdjustedProcessorCycles value (in the FastCounter field) that can be used in the calculation of the corresponding core crystal clock value of the TSC packet.

When the next MTC after a pair of TSC/TMA is seen, the number of crystal clocks passed since the TSC packet can be calculated by subtracting the TMA.CTC value from the time indicated by the MTC_{Next} packet by

$$\text{CTC}_{\text{Delta}}[15:0] = (\text{CTC}_{\text{Next}}[15:0] - \text{TMA.CTC}[15:0]), \text{ where } \text{CTC}_{\text{Next}} = \text{MTC}_{\text{Payload}} \ll \text{IA32\_RTIT\_CTL.MTCFreq}.$$

The TMA.FastCounter field provides the number of AdjustedProcessorCycles since the last crystal clock rising edge, from which it can be determined the percentage of the next crystal clock cycle that had passed at the time of the TSC packet.

CYC packets can provide further precision of an estimated timestamp value to many non-timing packets, by providing an indication of the time passed between other timing packets (MTCs or TSCs).

When enabled, CYC packets are sent preceding each CYC-eligible packet, and provide the number of processor core clock cycles that have passed since the last CYC packet. Thus between MTCs and TSCs, the accumulated CYC values can be used to estimate the AdjustedProcessorCycles component of the timestamp value. The accumulated CPU cycles will have to be adjusted to account for the difference in frequency between the processor core clock and

the P1 frequency. The necessary adjustment can be estimated using the core:bus ratio value given in the CBR packet, by multiplying the accumulated cycle count value by $P1/CBR_{\text{payload}}$.

Note that stand-alone TSC packets (that is, TSC packets that are not a part of a PSB+) are typically generated only when generation of other timing packets (MTCs and CYCs) has ceased for a period of time. Example scenarios include when Intel PT is re-enabled, or on wake after a sleep state. Thus any calculated estimate of the timestamp value leading up to a TSC packet will likely result in a discrepancy, which the TSC packet serves to correct.

A greater level of precision may be achieved by calculating the CPU clock frequency, see Section 35.8.3.4 below for a method to do so using Intel PT packets.

CYCs can be used to estimate time between TSCs even without MTCs, though this will likely result in a reduction in estimated TSC precision.

35.8.3.3 VMX TSC Manipulation

When software executes in non-Root operation, additional offset and scaling factors may be applied to the TSC value. These are optional, but may be enabled via VMCS controls on a per-VM basis. See Chapter 27, “VMX Non-Root Operation,” for details on VMX TSC offsetting and TSC scaling.

Like the value returned by RDTSC, TSC packets will include these adjustments, but other timing packets (such as MTC, CYC, and CBR) are not impacted. In order to use the algorithm above to estimate the TSC value when TSC scaling is in use, it will be necessary for software to account for the scaling factor. See Section 35.5.2.4 for details.

35.8.3.4 Calculating Frequency with Intel PT

Because Intel PT can provide both wall-clock time and processor clock cycle time, it can be used to measure the processor core clock frequency. Either TSC or MTC packets can be used to track the wall-clock time. By using CYC packets to count the number of processor core cycles that pass in between a pair of wall-clock time packets, the ratio between processor core clock frequency and TSC frequency can be derived. If the P1 frequency is known, it can be applied to determine the CPU frequency. See Section 35.8.3.1 above for details on the relationship between TSC, MTC, and CYC.

29. Updates to Chapter 37, Volume 3D

Change bars and violet text show changes to Chapter 37 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Chapter 37, "Enclave Access Control and Data Structures" was previously numbered as 36. With the insertion of the new SEAM chapter (chapter 34), this was subsequently re-numbered.
- In Section 37.9, "State Save Area (SSA) Frame", updated offset and size values for the GPRSGX in Table 37-8.

CHAPTER 37

ENCLAVE ACCESS CONTROL AND DATA STRUCTURES

37.1 OVERVIEW OF ENCLAVE EXECUTION ENVIRONMENT

When an enclave is created, it has a range of linear addresses to which the processor applies enhanced access control. This range is called the ELRANGE (see Section 36.3). When an enclave generates a memory access, the existing IA32 segmentation and paging architecture are applied. Additionally, linear addresses inside the ELRANGE must map to an EPC page otherwise when an enclave attempts to access that linear address a fault is generated.

The EPC pages need not be physically contiguous. System software allocates EPC pages to various enclaves. Enclaves must abide by OS/VMM imposed segmentation and paging policies. OS/VMM-managed page tables and extended page tables provide address translation for the enclave pages. Hardware requires that these pages are properly mapped to EPC (any failure generates an exception).

Enclave entry must happen through specific enclave instructions:

- ENCLU[EENTER], ENCLU[ERESUME].

Enclave exit must happen through specific enclave instructions or events:

- ENCLU[EEXIT], Asynchronous Enclave Exit (AEX).

Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations. As an example a read of an enclave page may result in the return of all one's or return of cyphertext of the cache line. Writing to an enclave page may result in a dropped write or a machine check at a later time. The processor will provide the protections as described in Section 37.4 and Section 37.5 on such accesses.

37.2 TERMINOLOGY

A memory access to the ELRANGE and initiated by an instruction executed by an enclave is called a Direct Enclave Access (Direct EA).

Memory accesses initiated by certain Intel® SGX instruction leaf functions such as ECREATE, EADD, EDBGRD, EDBGWR, ELDU/ELDB, EWB, EREMOVE, EENTER, and ERESUME to EPC pages are called Indirect Enclave Accesses (Indirect EA). Table 37-1 lists additional details of the indirect EA of SGX1 and SGX2 extensions.

Direct EAs and Indirect EAs together are called Enclave Accesses (EAs).

Any memory access that is not an Enclave Access is called a non-enclave access.

37.3 ACCESS-CONTROL REQUIREMENTS

Enclave accesses have the following access-control attributes:

- All memory accesses must conform to segmentation and paging protection mechanisms.
- Code fetches from inside an enclave to a linear address outside that enclave result in a #GP(0) exception.
- Shadow-stack-load or shadow-stack-store from inside an enclave to a linear address outside that enclave results in a #GP(0) exception.
- Non-enclave accesses to EPC memory result in undefined behavior. EPC memory is protected as described in Section 37.4 and Section 37.5 on such accesses.
- EPC pages of page types PT_REG, PT_TCS, and PT_TRIM must be mapped to ELRANGE at the linear address specified when the EPC page was allocated to the enclave using ENCLS[EADD] or ENCLS[EAUG] leaf functions. Enclave accesses through other linear address result in a #PF with the PFEC.SGX bit set.

- Direct EAs to any EPC pages must conform to the currently defined security attributes for that EPC page in the EPCM. These attributes may be defined at enclave creation time (EADD) or when the enclave sets them using SGX2 instructions. The failure of these checks results in a #PF with the PFEC.SGX bit set.
 - Target page must belong to the currently executing enclave.
 - Data may be written to an EPC page if the EPCM allow write access.
 - Data may be read from an EPC page if the EPCM allow read access.
 - Instruction fetches from an EPC page are allowed if the EPCM allows execute access.
 - Shadow-stack-load from an EPC page and shadow-stack-store to an EPC page are allowed only if the page type is PT_SS_FIRST or PT_SS_REST.
 - Data writes that are not shadow-stack-store are not allowed if the EPCM page type is PT_SS_FIRST or PT_SS_REST.
 - Target page must not have a restricted page type¹ (PT_SECS, PT_TCS, PT_VA, or PT_TRIM).
 - The EPC page must not be BLOCKED.
 - The EPC page must not be PENDING.
 - The EPC page must not be MODIFIED.

37.4 SEGMENT-BASED ACCESS CONTROL

Intel SGX architecture does not modify the segment checks performed by a logical processor. All memory accesses arising from a logical processor in protected mode (including enclave access) are subject to segmentation checks with the applicable segment register.

To ensure that outside entities do not modify the enclave's logical-to-linear address translation in an unexpected fashion, ENCLU[EENTER] and ENCLU[ERESUME] check that CS, DS, ES, and SS, if usable (i.e., not null), have segment base value of zero. A non-zero segment base value for these registers results in a #GP(0).

On enclave entry either via EENTER or ERESUME, the processor saves the contents of the external FS and GS registers, and loads these registers with values stored in the TCS at build time to enable the enclave's use of these registers for accessing the thread-local storage inside the enclave. On EEXIT and AEX, the contents at time of entry are restored. On AEX, the values of FS and GS are saved in the SSA frame. On ERESUME, FS and GS are restored from the SSA frame. The details of these operations can be found in the descriptions of EENTER, ERESUME, EEXIT, and AEX flows.

37.5 PAGE-BASED ACCESS CONTROL

37.5.1 Access-control for Accesses that Originate from Non-SGX Instructions

Intel SGX builds on the processor's paging mechanism to provide page-granular access-control for enclave pages. Enclave pages are designed to be accessible only from inside the currently executing enclave if they belong to that enclave. In addition, enclave accesses must conform to the access control requirements described in Section 37.3. or through certain Intel SGX instructions. Attempts to execute, read, or write to linear addresses mapped to EPC pages when not inside an enclave will result in the processor altering the access to preserve the confidentiality and integrity of the enclave. The exact behavior may be different between implementations.

37.5.2 Memory Accesses that Split Across ELRANGE

Memory data accesses are allowed to split across ELRANGE (i.e., a part of the access is inside ELRANGE and a part of the access is outside ELRANGE) while the processor is inside an enclave. If an access splits across ELRANGE, the

1. EPCM may allow write, read or execute access only for pages with page type PT_REG.

processor splits the access into two sub-accesses (one inside ELRANGE and the other outside ELRANGE), and each access is evaluated. A code-fetch access that splits across ELRANGE results in a #GP due to the portion that lies outside of the ELRANGE.

37.5.3 Implicit vs. Explicit Accesses

Memory accesses originating from Intel SGX instruction leaf functions are categorized as either explicit accesses or implicit accesses. Table 37-1 lists the implicit and explicit memory accesses made by Intel SGX leaf functions.

37.5.3.1 Explicit Accesses

Accesses to memory locations provided as explicit operands to Intel SGX instruction leaf functions, or their linked data structures are called explicit accesses.

Explicit accesses are always made using logical addresses. These accesses are subject to segmentation, paging, extended paging, and APIC-virtualization checks, and trigger any faults/exit associated with these checks when the access is made.

The interaction of explicit memory accesses with data breakpoints is leaf-function-specific, and is documented in Section 42.3.4.

37.5.3.2 Implicit Accesses

Accesses to data structures whose physical addresses are cached by the processor are called implicit accesses. These addresses are not passed as operands of the instruction but are implied by use of the instruction.

These accesses do not trigger any access-control faults/exits or data breakpoints. Table 37-1 lists memory objects that Intel SGX instruction leaf functions access either by explicit access or implicit access. The addresses of explicit access objects are passed via register operands with the second through fourth column of Table 37-1 matching implicitly encoded registers RBX, RCX, RDX.

Physical addresses used in different implicit accesses are cached via different instructions and for different durations. The physical address of SECS associated with each EPC page is cached at the time the page is added to the enclave via ENCLS[EADD] or ENCLS[EAUG], or when the page is loaded to EPC via ENCLS[ELDB] or ENCLS[ELDU]. This binding is severed when the corresponding page is removed from the EPC via ENCLS[EREMOVE] or ENCLS[EWB]. Physical addresses of TCS and SSA pages are cached at the time of most-recent enclave entry. Exit from an enclave (ENCLU[EEXIT] or AEX) flushes this caching. Details of Asynchronous Enclave Exit is described in Chapter 39.

The physical addresses that are cached for use by implicit accesses are derived from logical (or linear) addresses after checks such as segmentation, paging, EPT, and APIC virtualization checks. These checks may trigger exceptions or VM exits. Note, however, that such exception or VM exits may not occur after a physical address is cached and used for an implicit access.

Table 37-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EACCEPT	SGX2	SECINFO	EPCPAGE		SECS
EACCEPTCOPY	SGX2	SECINFO	EPCPAGE (Src)	EPCPAGE (Dst)	
EADD	SGX1	PAGEINFO and linked structures	EPCPAGE		
EAUG	SGX2	PAGEINFO and linked structures	EPCPAGE		SECS
EBLOCK	SGX1	EPCPAGE			SECS
ECREATE	SGX1	PAGEINFO and linked structures	EPCPAGE		
EDBGRD	SGX1	EPCADDR	Destination		SECS
EDBGWR	SGX1	EPCADDR	Source		SECS
EENTER	SGX1	TCS and linked SSA			SECS
EEXIT	SGX1				SECS, TCS

Table 37-1. List of Implicit and Explicit Memory Access by Intel® SGX Enclave Instructions (Contd.)

Instr. Leaf	Enum.	Explicit 1	Explicit 2	Explicit 3	Implicit
EEXTEND	SGX1	SECS	EPCPAGE		
EGETKEY	SGX1	KEYREQUEST	KEY		SECS
EINIT	SGX1	SIGSTRUCT	SECS	EINITTOKEN	
ELDB/ELDU	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	
EMODPE	SGX2	SECINFO	EPCPAGE		
EMODPR	SGX2	SECINFO	EPCPAGE		SECS
EMODT	SGX2	SECINFO	EPCPAGE		SECS
EPA	SGX1	EPCADDR			
EREMOVE	SGX1	EPCPAGE			SECS
EREPORT	SGX1	TARGETINFO	REPORTDATA	OUTPUTDATA	SECS
ERESUME	SGX1	TCS and linked SSA			SECS
ETRACK	SGX1	EPCPAGE			
EWB	SGX1	PAGEINFO and linked structures, PCMD	EPCPAGE	VAPAGE	SECS
Asynchronous Enclave Exit*					SECS, TCS, SSA
*Details of Asynchronous Enclave Exit (AEX) is described in Section 39.4					

37.6 INTEL® SGX DATA STRUCTURES OVERVIEW

Enclave operation is managed via a collection of data structures. Many of the top-level data structures contain sub-structures. The top-level data structures relate to parameters that may be used in enclave setup/maintenance, by Intel SGX instructions, or AEX event. The top-level data structures are:

- SGX Enclave Control Structure (SECS)
- Thread Control Structure (TCS)
- State Save Area (SSA)
- Page Information (PAGEINFO)
- Security Information (SECINFO)
- Paging Crypto MetaData (PCMD)
- Enclave Signature Structure (SIGSTRUCT)
- EINIT Token Structure (EINITTOKEN)
- Report Structure (REPORT)
- Report Target Info (TARGETINFO)
- Key Request (KEYREQUEST)
- Version Array (VA)
- Enclave Page Cache Map (EPCM)

Details of the top-level data structures and associated sub-structures are listed in Section 37.7 through Section 37.20.

37.7 SGX ENCLAVE CONTROL STRUCTURE (SECS)

The SECS data structure requires 4K-Bytes alignment.

Table 37-2. Layout of SGX Enclave Control Structure (SECS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
SIZE	0	8	Size of enclave in bytes; must be power of 2.
BASEADDR	8	8	Enclave Base Linear Address must be naturally aligned to size.
SSAFRAMESIZE	16	4	Size of one SSA frame in pages, including XSAVE, pad, GPR, and MISC (if CPUID.12H.00H:EBX != 0).
MISCSELECT	20	4	Bit vector specifying which extended features are saved to the MISC region (see Section 37.7.2) of the SSA frame when an AEX occurs.
CET_LEG_BITMAP_OFFSET	24	8	Page aligned offset of legacy code page bitmap from enclave base. Software is expected to program this offset such that the entire bitmap resides in the ELRANGE when legacy compatibility mode for indirect branch tracking is enabled. However this is not enforced by the hardware. This field exists when CPUID.07H.00H:EDX.CET_IBT[20] is enumerated as 1, else it is reserved.
CET_ATTRIBUTES	32	1	CET feature attributes of the enclave; see Table 37-5. This field exists when CPUID.12H.01H:EAX[6] is enumerated as 1, else it is reserved.
RESERVED	33	15	
ATTRIBUTES	48	16	Attributes of the Enclave, see Table 37-3.
MRENCLAVE	64	32	Measurement Register of enclave build process. See SIGSTRUCT for format.
RESERVED	96	32	
MRSIGNER	128	32	Measurement Register extended with the public key that verified the enclave. See SIGSTRUCT for format.
RESERVED	160	32	
CONFIGID	192	64	Post EINIT configuration identity.
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Post EINIT configuration security version number (SVN).
RESERVED	262	3834	The RESERVED field consists of the following: ▪ EID: An 8 byte Enclave Identifier. Its location is implementation specific. ▪ PAD: A 352 bytes padding pattern from the Signature (used for key derivation strings). It's location is implementation specific. ▪ ISVFAMILYID: A 16 byte value assigned to identify the family of products the enclave belongs to. ▪ ISVEXTPRODID: A 16 byte value assigned to identify the product identity of the enclave. ▪ The remaining 3226 bytes are reserved area. The entire 3834 byte field must be cleared prior to executing ECREATE.

37.7.1 ATTRIBUTES

The ATTRIBUTES data structure is comprised of bit-granular fields that are used in the SECS, the REPORT and the KEYREQUEST structures. CPUID.12H.01H enumerates a bitmap of permitted 1-setting of bits in ATTRIBUTES.

Table 37-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
INIT	0	This bit indicates if the enclave has been initialized by EINIT. It must be cleared when loaded as part of ECREATE. For EREPORT instruction, TARGET_INFO.ATTRIBUTES[ENIT] must always be 1 to match the state after EINIT has initialized the enclave.
DEBUG	1	If 1, the enclave permit debugger to read and write enclave data using EDBGD and EDBGWR.

Table 37-3. Layout of ATTRIBUTES Structure

Field	Bit Position	Description
MODE64BIT	2	Enclave runs in 64-bit mode.
RESERVED	3	Must be Zero.
PROVISIONKEY	4	Provisioning Key is available from EGETKEY.
EINITTOKEN_KEY	5	EINIT token key is available from EGETKEY.
CET	6	Enable CET attributes. When CPUID.12H.01H:EAX[6] is 0 this bit is reserved and must be 0.
KSS	7	Key Separation and Sharing Enabled.
RESERVED	9:8	Must be zero.
AEXNOTIFY	10	The bit indicates that threads within the enclave may receive AEX notifications.
RESERVED	63:11	Must be zero.
XFRM	127:64	XSAVE Feature Request Mask. See Section 41.7.

37.7.2 SECS.MISCSELECT Field

CPUID.12H.00H:EBX[31:0] enumerates which extended information that the processor can save into the MISC region of SSA when an AEX occurs. An enclave writer can specify via SIGSTRUCT how to set the SECS.MISCSELECT field. The bit vector of MISCSELECT selects which extended information is to be saved in the MISC region of the SSA frame when an AEX is generated. The bit vector definition of extended information is listed in Table 37-4.

If CPUID.12H.00H:EBX[31:0] = 0, SECS.MISCSELECT field must be all zeros.

The SECS.MISCSELECT field determines the size of MISC region of the SSA frame, see Section 37.9.2.

Table 37-4. Bit Vector Layout of MISCSELECT Field of Extended Information

Field	Bit Position	Description
EXINFO	0	Report information about page fault and general protection exception that occurred inside an enclave.
CPINFO	1	Report information about control protection exception that occurred inside an enclave. When CPUID.12H.00H:EBX[1] is 0, this bit is reserved.
Reserved	31:2	Reserved (0).

37.7.3 SECS.CET_ATTRIBUTES Field

The SECS.CET_ATTRIBUTES field can be used by the enclave writer to enable various CET attributes in an enclave. This field exists when CPUID.12H.01H:EAX[6] is enumerated as 1. Bits 1:0 are defined when CPUID.07H.00H:ECX.CET_SS is 1, and bits 5:2 are defined when CPUID.07H.00H:EDX.CET_IBT is 1.

Table 37-5. Bit Vector Layout of CET_ATTRIBUTES Field of Extended Information

Field	Bit Position	Description
SH_STK_EN	0	When set to 1, enable shadow stacks.
WR_SHSTK_EN	1	When set to 1, enables the WRSS{D,Q}W instructions.
ENDBR_EN	2	When set to 1, enables indirect branch tracking.
LEG_IW_EN	3	Enable legacy compatibility treatment for indirect branch tracking.
NO_TRACK_EN	4	When set to 1, enables use of no-track prefix for indirect branch tracking.
SUPPRESS_DIS	5	When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.
Reserved	7:6	Reserved (0).

37.8 THREAD CONTROL STRUCTURE (TCS)

Each executing thread in the enclave is associated with a Thread Control Structure. It requires 4K-Bytes alignment.

Table 37-6. Layout of Thread Control Structure (TCS)

Field	OFFSET (Bytes)	Size (Bytes)	Description
STAGE	0	8	Enclave execution state of the thread controlled by this TCS. A value of 0 indicates that this TCS is available for enclave entry. A value of 1 indicates that a logical processor is currently executing an enclave in the context of this TCS.
FLAGS	8	8	The thread's execution flags (see Section 37.8.1).
OSSA	16	8	Offset of the base of the State Save Area stack, relative to the enclave base. Must be page aligned.
CSSA	24	4	Current slot index of an SSA frame, cleared by EADD and EACCEPT.
NSSA	28	4	Number of available slots for SSA frames.
OENTRY	32	8	Offset in enclave to which control is transferred on EENTER relative to the base of the enclave.
AEP	40	8	The value of the Asynchronous Exit Pointer that was saved at EENTER time.
OFSBASE	48	8	Offset to add to the base address of the enclave for producing the base address of FS segment inside the enclave. Must be page aligned.
OGSBASE	56	8	Offset to add to the base address of the enclave for producing the base address of GS segment inside the enclave. Must be page aligned.
FSLIMIT	64	4	Size to become the new FS limit in 32-bit mode.
GSLIMIT	68	4	Size to become the new GS limit in 32-bit mode.
OCETSSA	72	8	When CPUID.12H.01H:EAX[6] is 1, this field provides the offset of the CET state save area from enclave base. When CPUID.12H.01H:EAX[6] is 0, this field is reserved and must be 0.
PREVSSP	80	8	When CPUID.07H.00H:ECX.CET_SS is 1, this field records the SSP at the time of AEX or EEXIT; used to setup SSP on entry. When CPUID.07H.00H:ECX.CET_SS is 0, this field is reserved and must be 0.
RESERVED	88	4008	Must be zero.

37.8.1 TCS.FLAGS

Table 37-7. Layout of TCS.FLAGS Field

Field	Bit Position	Description
DBGOPTIN	0	If set, allows debugging features (single-stepping, breakpoints, etc.) to be enabled and active while executing in the enclave on this TCS. Hardware clears this bit on EADD. A debugger may later modify it if the enclave's ATTRIBUTES.DEBUG is set.
AEXNOTIFY	1	A thread that enters the enclave cannot receive AEX notifications unless this flag is set to 1.
RESERVED	63:2	Must be zero.

37.8.2 State Save Area Offset (OSSA)

The OSSA points to a stack of State Save Area (SSA) frames (see Section 37.9) used to save the processor state when an interrupt or exception occurs while executing in the enclave.

37.8.3 Current State Save Area Frame (CSSA)

CSSA is the index of the current SSA frame that will be used by the processor to determine where to save the processor state on an interrupt or exception that occurs while executing in the enclave. It is an index into the array of frames addressed by OSSA. CSSA is incremented on an AEX and decremented on an ERESUME.

37.8.4 Number of State Save Area Frames (NSSA)

NSSA specifies the number of SSA frames available for this TCS. There must be at least one available SSA frame when EENTER-ing the enclave or the EENTER will fail.

37.9 STATE SAVE AREA (SSA) FRAME

When an AEX occurs while running in an enclave, the architectural state is saved in the thread's current SSA frame, which is pointed to by TCS.CSSA. An SSA frame must be page aligned, and contains the following regions:

- The XSAVE region starts at the base of the SSA frame, this region contains extended feature register state in an XSAVE/FXSAVE-compatible non-compacted format.
- A Pad region: software may choose to maintain a pad region separating the XSAVE region and the MISC region. Software choose the size of the pad region according to the sizes of the MISC and GPRSGX regions.
- The GPRSGX region. The GPRSGX region is the last region of an SSA frame (see Table 37-8). This is used to hold the processor general purpose registers (RAX ... R15), the RIP, the outside RSP and RBP, RFLAGS, and the AEX information.
- The MISC region (If CPUID.12H.00H:EBX[31:0] != 0). The MISC region is adjacent to the GRPSGX region, and may contain zero or more components of extended information that would be saved when an AEX occurs. If the MISC region is absent, the region between the GPRSGX and XSAVE regions is the pad region that software can use. If the MISC region is present, the region between the MISC and XSAVE regions is the pad region that software can use. See additional details in Section 37.9.2.

Table 37-8. Top-to-Bottom Layout of an SSA Frame

Region	Offset (Byte)	Size (Bytes)	Description
XSAVE	0	Calculate using CPUID leaf 0DH information	The size of XSAVE region in SSA is derived from the enclave's support of the collection of processor extended states that would be managed by XSAVE. The enablement of those processor extended state components in conjunction with CPUID.0DH information determines the XSAVE region size in SSA.
Pad	End of XSAVE region	Chosen by enclave writer	Ensure the end of GPRSGX region is aligned to the end of a 4KB page.
MISC	base of GPRSGX - sizeof(MISC)	Calculate from highest set bit of SECS.MISCSELECT	See Section 37.9.2.
GPRSGX	SSAFRAMESIZE - 184	184	See Table 37-9 for layout of the GPRSGX region.

37.9.1 GPRSGX Region

The layout of the GPRSGX region is shown in Table 37-9.

Table 37-9. Layout of GPRSGX Portion of the State Save Area

Field	OFFSET (Bytes)	Size (Bytes)	Description
RAX	0	8	
RCX	8	8	

Table 37-9. Layout of GPRSGX Portion of the State Save Area (Contd.)

Field	OFFSET (Bytes)	Size (Bytes)	Description
RDX	16	8	
RBX	24	8	
RSP	32	8	
RBP	40	8	
RSI	48	8	
RDI	56	8	
R8	64	8	
R9	72	8	
R10	80	8	
R11	88	8	
R12	96	8	
R13	104	8	
R14	112	8	
R15	120	8	
RFLAGS	128	8	Flag register.
RIP	136	8	Instruction pointer.
URSP	144	8	Non-Enclave (outside) stack pointer. Saved by EENTER, restored on AEX.
URBP	152	8	Non-Enclave (outside) RBP pointer. Saved by EENTER, restored on AEX.
EXITINFO	160	4	Contains information about exceptions that cause AEXs, which might be needed by enclave software (see Section 37.9.1.1).
RESERVED	164	3	
AEXNOTIFY	167	1	Bit 0: This bit allows enclave software to dynamically enable/disable AEX notifications. An enclave thread cannot receive AEX notifications unless this bit is set to 1 in the thread's current SSA frame. All other bits are reserved.
FSBASE	168	8	FS BASE.
GSBASE	176	8	GS BASE.

37.9.1.1 EXITINFO

EXITINFO contains the information used to report exit reasons to software inside the enclave. It is a 4 byte field laid out as in Table 37-10. The VALID bit is set only for the exceptions conditions which are reported inside an enclave. See Table 37-11 for which exceptions are reported inside the enclave. If the exception condition is not one reported inside the enclave then VECTOR and EXIT_TYPE are cleared.

When a higher priority event, such as SMI, and a pending debug exception occur at the same time when executing inside an enclave, the higher priority event has precedence. As an example for an SMI, the SSA exit info is zero. The debug exception will be delivered upon return from the SMI. In such cases, the EXITINFO field will not contain the information of a debug exception.

Table 37-10. Layout of EXITINFO Field

Field	Bit Position	Description
VECTOR	7:0	Exception number of exceptions reported inside enclave.
EXIT_TYPE	10:8	011b: Hardware exceptions. 110b: Software exceptions. Other values: Reserved.

Table 37-10. Layout of EXITINFO Field

Field	Bit Position	Description
RESERVED	30:11	Reserved as zero.
VALID	31	0: unsupported exceptions. 1: Supported exceptions. Includes two categories: - Unconditionally supported exceptions: #DE, #DB, #BP, #BR, #UD, #MF, #AC, #XM. - Conditionally supported exception: #PF, #GP if SECS.MISCSELECT.EXINFO = 1. #CP if SECS.MISCSELECT.CPINFO = 1.

37.9.1.2 VECTOR Field Definition

Table 37-11 contains the VECTOR field. This field contains information about some exceptions which occur inside the enclave. These vector values are the same as the values that would be used when vectoring into regular exception handlers. All values not shown are not reported inside an enclave.

Table 37-11. Exception Vectors

Name	Vector #	Description
#DE	0	Divider exception.
#DB	1	Debug exception.
#BP	3	Breakpoint exception.
#BR	5	Bound range exceeded exception.
#UD	6	Invalid opcode exception.
#GP	13	General protection exception. Only reported if SECS.MISCSELECT.EXINFO = 1.
#PF	14	Page fault exception. Only reported if SECS.MISCSELECT.EXINFO = 1.
#MF	16	x87 FPU floating-point error.
#AC	17	Alignment check exceptions.
#XM	19	SIMD floating-point exceptions.
#CP	21	Control protection exception. Only reported if SECS.MISCSELECT.CPINFO = 1.

37.9.2 MISC Region

The layout of the MISC region is shown in Table 37-12. The number of components that the processor supports in the MISC region corresponds to the bits of CPUID.12H.00H:EBX[31:0] set to 1. Each set bit in CPUID.12H.00H:EBX[31:0] has a defined size for the corresponding component, as shown in Table 37-12. Enclave writers needs to do the following:

- Decide which MISC region components will be supported for the enclave.
- Allocate an SSA frame large enough to hold the components chosen above.
- Instruct each enclave builder software to set the appropriate bits in SECS.MISCSELECT.

The first component, EXINFO, starts next to the GPRSGX region. Additional components in the MISC region grow in ascending order within the MISC region towards the XSAVE region.

The size of the MISC region is calculated as follows:

- If CPUID.12H.00H:EBX[31:0] = 0, MISC region is not supported.
- If CPUID.12H.00H:EBX[31:0] != 0, the size of MISC region is derived from sum of the highest bit set in SECS.MISCSELECT and the size of the MISC component corresponding to that bit. Offset and size information of currently defined MISC components are listed in Table 37-12. For example, if the highest bit set in SECS.MISCSELECT is bit 0, the MISC region offset is OFFSET(GPRSGX)-16 and size is 16 bytes.

- The processor saves a MISC component *i* in the MISC region if and only if SECS.MISCSELECT[*i*] is 1.

Table 37-12. Layout of MISC region of the State Save Area

MISC Components	OFFSET (Bytes)	Size (Bytes)	Description
EXINFO	Offset(GPRSGX) -16	16	If CPUID.12H.00H:EBX[0] = 1, exception information on #GP or #PF that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[0] = 1. If CPUID.12H.00H:EBX[1] = 1, exception information on #CP that occurred inside an enclave can be written to the EXINFO structure if specified by SECS.MISCSELECT[1] = 1.
Future Extension	Below EXINFO	TBD	Reserved. (Zero size if CPUID.12H.00H:EBX[31:1] = 0).

37.9.2.1 EXINFO Structure

Table 37-13 contains the layout of the EXINFO structure that provides additional information.

Table 37-13. Layout of EXINFO Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MADDR	0	8	If #PF: contains the page fault linear address that caused a page fault. If #GP: the field is cleared. If #CP: the field is cleared.
ERRCD	8	4	Exception error code for either #GP or #PF.
RESERVED	12	4	

37.9.2.2 Page Fault Error Code

A page-fault error code may be reported in EXINFO.ERRCD. The format of the error code is given in Figure 5-12, "Page-Fault Error Code."

Page faults that would report an error code clearing the U/S bit (indicating a supervisor-mode access) are not reported in EXINFO.

37.10 CET STATE SAVE AREA FRAME

The CET state save area consists of an array of CET state save frames. The number of CET state save frames is equal to the TCS.NSSA. The current CET SSA frame is indicated by TCS.CSSA. The offset of the CET state save area is specified by TCS.OCETSSA.

Table 37-14. Layout of CET State Save Area Frame

Field	Offset (Bytes)	Size (Bytes)	Description
SSP	0	8	Shadow Stack Pointer. This field is reserved when CPUID.07H.00H:ECX.CET_SS is 0.
IB_TRACK_STATE	8	8	Indirect branch tracker state: Bit 0: SUPPRESS - suppressed(1), tracking(0) Bit 1: TRACKER - IDLE (0), WAIT_FOR_ENDBRANCH (1) Bits 63:2 - Reserved This field is reserved when CPUID.07H.00H:EDX.CET_IBT is 0.

37.11 PAGE INFORMATION (PAGEINFO)

PAGEINFO is an architectural data structure that is used as a parameter to the EPC-management instructions. It requires 32-Byte alignment.

Table 37-15. Layout of PAGEINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
LINADDR	0	8	Enclave linear address.
SRCPGE	8	8	Effective address of the page where contents are located.
SECINFO/PCMD	16	8	Effective address of the SECINFO or PCMD (for ELDU, ELDB, EWB) structure for the page.
SECS	24	8	Effective address of EPC slot that currently contains the SECS.

37.12 SECURITY INFORMATION (SECINFO)

The SECINFO data structure holds meta-data about an enclave page.

Table 37-16. Layout of SECINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
FLAGS	0	8	Flags describing the state of the enclave page.
RESERVED	8	56	Must be zero.

37.12.1 SECINFO.FLAGS

The SECINFO.FLAGS are a set of fields describing the properties of an enclave page.

Table 37-17. Layout of SECINFO.FLAGS Field

Field	Bit Position	Description
R	0	If 1 indicates that the page can be read from inside the enclave; otherwise the page cannot be read from inside the enclave.
W	1	If 1 indicates that the page can be written from inside the enclave; otherwise the page cannot be written from inside the enclave.
X	2	If 1 indicates that the page can be executed from inside the enclave; otherwise the page cannot be executed from inside the enclave.
PENDING	3	If 1 indicates that the page is in the PENDING state; otherwise the page is not in the PENDING state.
MODIFIED	4	If 1 indicates that the page is in the MODIFIED state; otherwise the page is not in the MODIFIED state.
PR	5	If 1 indicates that a permission restriction operation on the page is in progress, otherwise a permission restriction operation is not in progress.
RESERVED	7:6	Must be zero.
PAGE_TYPE	15:8	The type of page that the SECINFO is associated with.
RESERVED	63:16	Must be zero.

37.12.2 PAGE_TYPE Field Definition

The SECINFO flags and EPC flags contain bits indicating the type of page.

Table 37-18. Supported PAGE_TYPE

TYPE	Value	Description
PT_SECS	0	Page is an SECS.
PT_TCS	1	Page is a TCS.
PT_REG	2	Page is a regular page.
PT_VA	3	Page is a Version Array.
PT_TRIM	4	Page is in trimmed state.
PT_SS_FIRST	5	When CPUID.12H.01H:EAX[6] is 1, Page is first page of a shadow stack. When CPUID.12H.01H:EAX[6] is 0, this value is reserved.
PT_SS_REST	6	When CPUID.12H.01H:EAX[6] is 1, Page is not first page of a shadow stack. When CPUID.12H.01H:EAX[6] is 0, this value is reserved.
	All others	Reserved.

37.13 PAGING CRYPTO METADATA (PCMD)

The PCMD structure is used to keep track of crypto meta-data associated with a paged-out page. Combined with PAGEINFO, it provides enough information for the processor to verify, decrypt, and reload a paged-out EPC page. The size of the PCMD structure (128 bytes) is architectural.

EWB calculates the Message Authentication Code (MAC) value and writes out the PCMD. ELDB/U reads the fields and checks the MAC.

The format of PCMD is as follows:

Table 37-19. Layout of PCMD Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
SECINFO	0	64	Flags describing the state of the enclave page; R/W by software.
ENCLAVEID	64	8	Enclave Identifier used to establish a cryptographic binding between paged-out page and the enclave.
RESERVED	72	40	Must be zero.
MAC	112	16	Message Authentication Code for the page, page meta-data and reserved field.

37.14 ENCLAVE SIGNATURE STRUCTURE (SIGSTRUCT)

SIGSTRUCT is a structure created and signed by the enclave developer that contains information about the enclave. SIGSTRUCT is processed by the EINIT leaf function to verify that the enclave was properly built.

SIGSTRUCT includes ENCLAVEHASH as SHA256 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32. Each of the 8 HASH dwords is stored in little-endian order.

SIGSTRUCT includes four 3072-bit integers (MODULUS, SIGNATURE, Q1, Q2). Each such integer is represented as a byte strings of length 384, with the most significant byte at the position "offset + 383", and the least significant byte at position "offset".

The (3072-bit integer) SIGNATURE should be an RSA signature, where: a) the RSA modulus (MODULUS) is a 3072-bit integer; b) the public exponent is set to 3; c) the signing procedure uses the EMSA-PKCS1-v1.5 format with DER encoding of the "DigestInfo" value as specified in of PKCS#1 v2.1/RFC 3447.

The 3072-bit integers Q1 and Q2 are defined by:

$q1 = \text{floor}(\text{Signature}^2 / \text{Modulus});$

$q2 = \text{floor}((\text{Signature}^3 - q1 * \text{Signature} * \text{Modulus}) / \text{Modulus});$

SIGSTRUCT must be page aligned

In column 5 of Table 37-20, 'Y' indicates that this field should be included in the signature generated by the developer.

Table 37-20. Layout of Enclave Signature Structure (SIGSTRUCT)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
HEADER	0	16	Must be byte stream 06000000E10000000000010000000000H	Y
VENDOR	16	4	Intel Enclave: 00008086H Non-Intel Enclave: 00000000H	Y
DATE	20	4	Build date is yyyyymmdd in hex: yyyy=4 digit year, mm=1-12, dd=1-31	Y
HEADER2	24	16	Must be byte stream 01010000600000006000000001000000H	Y
SWDEFINED	40	4	Available for software use.	Y
RESERVED	44	84	Must be zero.	Y
MODULUS	128	384	Module Public Key (keylength = 3072 bits).	N
EXPONENT	512	4	RSA Exponent = 3.	N
SIGNATURE	516	384	Signature over Header and Body.	N
MISCSELECT ¹	900	4	Bit vector specifying Extended SSA frame feature set to be used.	Y
MISCMASK	904	4	Bit vector mask of MISCSELECT to enforce.	Y
CET_ATTRIBUTES	908	1	When CPUID.12H.01H:EAX[6] is 1, this field provides the Enclave CET attributes that must be set. When CPUID.12H.01H:EAX[6] is 0, this field is reserved and must be 0.	Y
CET_ATTRIBUTES_MASK	909	1	When CPUID.12H.01H:EAX[6] is 1, this field provides the Mask of CET attributes to enforce. When CPUID.12H.01H:EAX[6] is 0, this field is reserved and must be 0.	Y
RESERVED	910	2	Must be zero.	Y
ISVFAMILYID	912	16	ISV assigned Product Family ID.	Y
ATTRIBUTES	928	16	Enclave Attributes that must be set.	Y
ATTRIBUTEMASK	944	16	Mask of Attributes to enforce.	Y
ENCLAVEHASH	960	32	MRENCLAVE of enclave this structure applies to.	Y
RESERVED	992	16	Must be zero.	Y
ISVEXTPRODID	1008	16	ISV assigned extended Product ID.	Y
ISVPRODID	1024	2	ISV assigned Product ID.	Y
ISVSVN	1026	2	ISV assigned SVN (security version number).	Y
RESERVED	1028	12	Must be zero.	N
Q1	1040	384	Q1 value for RSA Signature Verification.	N
Q2	1424	384	Q2 value for RSA Signature Verification.	N

NOTES:

1. If CPUID.12H.00H:EBX[31:0] = 0, MISCSELECT must be 0.

37.15 EINIT TOKEN STRUCTURE (EINITTOKEN)

The EINIT token is used by EINIT to verify that the enclave is permitted to launch. EINIT token is generated by an enclave in possession of the EINITTOKEN key (the Launch Enclave).

EINIT token must be 512-Byte aligned.

Table 37-21. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
Valid	0	4	Y	Bit 0: 1: Valid; 0: Invalid. All other bits reserved.
RESERVED	4	44	Y	Must be zero.
ATTRIBUTES	48	16	Y	ATTRIBUTES of the Enclave.
MRENCLAVE	64	32	Y	MRENCLAVE of the Enclave.
RESERVED	96	32	Y	Reserved.
MRSIGNER	128	32	Y	MRSIGNER of the Enclave.
RESERVED	160	32	Y	Reserved.
CPUSVNLE	192	16	N	Launch Enclave's CPUSVN.
ISVPRODIDLE	208	02	N	Launch Enclave's ISVPRODID.
ISVSVNLE	210	02	N	Launch Enclave's ISVSVN.
CET_MASKED_ATTRIBUTES_LE	212	1	N	When CPUID.12H.01H:EAX[6] is 1, this field provides the Launch enclaves masked CET attributes. This should be set to LE's CET_ATTRIBUTES masked with CET_ATTRIBUTES_MASK of the LE's KEYREQUEST. When CPUID.12H.01H:EAX[6] is 0, this field is reserved.
RESERVED	213	23	N	Reserved.
MASKEDMISCSELECTLE	236	4		Launch Enclave's MASKEDMISCSELECT: set by the LE to the resolved MISCSELECT value, used by EGETKEY (after applying KEYREQUEST's masking).
MASKEDATTRIBUTESLE	240	16	N	Launch Enclave's MASKEDATTRIBUTES: This should be set to the LE's ATTRIBUTES masked with ATTRIBUTEMASK of the LE's KEYREQUEST.
KEYID	256	32	N	Value for key wear-out protection.
MAC	288	16	N	Message Authentication Code on EINITTOKEN using EINITTOKEN_KEY.

37.16 REPORT (REPORT)

The REPORT structure is the output of the EREPORT instruction, and must be 512-Byte aligned.

Table 37-22. Layout of REPORT

Field	OFFSET (Bytes)	Size (Bytes)	Description
CPUSVN	0	16	The security version number of the processor.
MISCSELECT	16	4	Bit vector specifying which extended features are saved to the MISC region of the SSA frame when an AEX occurs.
CET_ATTRIBUTES	20	1	When CPUID.12H.01H:EAX[6] is 1, this field reports the CET_ATTRIBUTES of the Enclave. When CPUID.12H.01H:EAX[6] is 0, this field is reserved and must be 0.
RESERVED	21	11	Zero.
ISVEXTNPRODID	32	16	The value of SECS.ISVEXTPRODID.
ATTRIBUTES	48	16	ATTRIBUTES of the Enclave. See Section 37.7.1.
MRENCLAVE	64	32	The value of SECS.MRENCLAVE.
RESERVED	96	32	Zero.
MRSIGNER	128	32	The value of SECS.MRSIGNER.
RESERVED	160	32	Zero.
CONFIGID	192	64	Value provided by SW to identify enclave's post EINIT configuration.

Table 37-22. Layout of REPORT (Contd.)

Field	OFFSET (Bytes)	Size (Bytes)	Description
ISVPRODID	256	2	Product ID of enclave.
ISVSVN	258	2	Security version number (SVN) of the enclave.
CONFIGSVN	260	2	Value provided by SW to indicate expected SVN of enclave's post EINIT configuration.
RESERVED	262	42	Zero.
ISVFAMILYID	304	16	The value of SECS.ISVFAMILYID.
REPORTDATA	320	64	Data provided by the user and protected by the REPORT's MAC, see Section 37.16.1.
KEYID	384	32	Value for key wear-out protection.
MAC	416	16	Message Authentication Code on the report using report key.

37.16.1 REPORTDATA

REPORTDATA is a 64-Byte data structure that is provided by the enclave and included in the REPORT. It can be used to securely pass information from the enclave to the target enclave.

37.17 REPORT TARGET INFO (TARGETINFO)

This structure is an input parameter to the EREPORT leaf function. The address of TARGETINFO is specified as an effective address in RBX. It is used to identify the target enclave which will be able to cryptographically verify the REPORT structure returned by EREPORT. TARGETINFO must be 512-Byte aligned.

Table 37-23. Layout of TARGETINFO Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
MEASUREMENT	0	32	The MRENCLAVE of the target enclave.
ATTRIBUTES	32	16	The ATTRIBUTES field of the target enclave.
CET_ATTRIBUTES	48	1	When CPUID.12H.01H:EAX[6] is 1, this field provides the CET_ATTRIBUTES field of the target enclave. When CPUID.12H.01H:EAX[6] is 0, this field is reserved.
RESERVED	49	1	Must be zero.
CONFIGSVN	50	2	CONFIGSVN of the target enclave.
MISCSELECT	52	4	The MISCSELECT of the target enclave.
RESERVED	56	8	Must be zero.
CONFIGID	64	64	CONFIGID of target enclave.
RESERVED	128	384	Must be zero.

37.18 KEY REQUEST (KEYREQUEST)

This structure is an input parameter to the EGETKEY leaf function. It is passed in as an effective address in RBX and must be 512-Byte aligned. It is used for selecting the appropriate key and any additional parameters required in the derivation of that key.

Table 37-24. Layout of KEYREQUEST Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
KEYNAME	0	2	Identifies the Key Required.
KEYPOLICY	2	2	Identifies which inputs are required to be used in the key derivation.
ISVSVN	4	2	The ISV security version number that will be used in the key derivation.
CET_ATTRIBUTES_MASK	6	1	When CPUID.12H.01H:EAX[6] is 1, this field provides a mask that defines which CET_ATTRIBUTES bits will be included in key derivation. When CPUID.12H.01H:EAX[6] is 0, then this field is reserved and must be 0.
RESERVED	7	1	Must be zero.
CPUSVN	8	16	The security version number of the processor used in the key derivation.
ATTRIBUTEMASK	24	16	A mask defining which ATTRIBUTES bits will be included in key derivation.
KEYID	40	32	Value for key wear-out protection.
MISCMASK	72	4	A mask defining which MISCSELECT bits will be included in key derivation.
CONFIGSVN	76	2	Identifies which enclave Configuration's Security Version should be used in key derivation.
RESERVED	78	434	

37.18.1 KEY REQUEST KeyNames

Table 37-25. Supported KEYName Values

Key Name	Value	Description
EINIT_TOKEN_KEY	0	EINIT_TOKEN key
PROVISION_KEY	1	Provisioning Key
PROVISION_SEAL_KEY	2	Provisioning Seal Key
REPORT_KEY	3	Report Key
SEAL_KEY	4	Seal Key
	All others	Reserved

37.18.2 Key Request Policy Structure

Table 37-26. Layout of KEYPOLICY Field

Field	Bit Position	Description
MRENCLAVE	0	If 1, derive key using the enclave's MRENCLAVE measurement register.
MRSIGNER	1	If 1, derive key using the enclave's MRSIGNER measurement register.
NOISVPRODID	2	If 1, derive key WITHOUT using the enclave's ISVPRODID value.
CONFIGID	3	If 1, derive key using the enclave's CONFIGID value.
ISVFAMILYID	4	If 1, derive key using the enclave ISVFAMILYID value.
ISVEXTPRODID	5	If 1, derive key using enclave's ISVEXTPRODID value.
RESERVED	15:6	Must be zero.

37.19 VERSION ARRAY (VA)

In order to securely store the versions of evicted EPC pages, Intel SGX defines a special EPC page type called a Version Array (VA). Each VA page contains 512 slots, each of which can contain an 8-byte version number for a page evicted from the EPC. When an EPC page is evicted, software chooses an empty slot in a VA page; this slot receives the unique version number of the page being evicted. When the EPC page is reloaded, there must be a VA slot that must hold the version of the page. If the page is successfully reloaded, the version in the VA slot is cleared.

VA pages can be evicted, just like any other EPC page. When evicting a VA page, a version slot in some other VA page must be used to hold the version for the VA being evicted. A Version Array Page must be 4K-Bytes aligned.

Table 37-27. Layout of Version Array Data Structure

Field	OFFSET (Bytes)	Size (Bytes)	Description
Slot 0	0	8	Version Slot 0
Slot 1	8	8	Version Slot 1
...			
Slot 511	4088	8	Version Slot 511

37.20 ENCLAVE PAGE CACHE MAP (EPCM)

EPCM is a secure structure used by the processor to track the contents of the EPC. The EPCM holds exactly one entry for each page that is currently loaded into the EPC. EPCM is not accessible by software, and the layout of EPCM fields is implementation specific.

Table 37-28. Content of an Enclave Page Cache Map Entry

Field	Description
VALID	Indicates whether the EPCM entry is valid.
R	Read access; indicates whether enclave accesses for reads are allowed from the EPC page referenced by this entry.
W	Write access; indicates whether enclave accesses for writes are allowed to the EPC page referenced by this entry.
X	Execute access; indicates whether enclave accesses for instruction fetches are allowed from the EPC page referenced by this entry.
PT	EPCM page type (PT_SECS, PT_TCS, PT_REG, PT_VA, PT_TRIM, PT_SS_FIRST, PT_SS_REST).
ENCLAVESECS	SECS identifier of the enclave to which the EPC page belongs.
ENCLAVEADDRESS	Linear enclave address of the EPC page.
BLOCKED	Indicates whether the EPC page is in the blocked state.
PENDING	Indicates whether the EPC page is in the pending state.
MODIFIED	Indicates whether the EPC page is in the modified state.
PR	Indicates whether the EPC page is in a permission restriction state.

30. Updates Appendix A, Volume 3D

Change bars and violet text show changes to Appendix A of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- In Section A.1, "Basic VMX Information", updated footnote 2 in bit 48 description regarding processor's physical-address width.

APPENDIX A

VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by `CPUID.01H:ECX.VMX[5] = 1`. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 28 and other VMX chapters is determined by reading values from a set of capability MSR. These MSR are indexed starting at MSR address 480H. VMX capability MSR are read-only; an attempt to write them (with `WRMSR`) produces a general-protection exception (`#GP(0)`). They do not exist on processors that do not support VMX operation; an attempt to read them (with `RDMR`) on such processors produces a general-protection exception (`#GP(0)`).

A.1 BASIC VMX INFORMATION

The `IA32_VMX_BASIC` MSR (index 480H) consists of the following fields:

- Bits 30:0 contain the 31-bit VMCS revision identifier used by the processor. Processors that use the same VMCS revision identifier use the same size for VMCS regions (see subsequent item on bits 44:32).¹
- Bit 31 is always 0.
- Bits 44:32 report the number of bytes that software should allocate for the VMXON region and any VMCS region. It is a value greater than 0 and at most 4096 (bit 44 is set if and only if bits 43:32 are clear).
- Bit 48 indicates the width of the physical addresses that may be used for the VMXON region, each VMCS, and data structures referenced by pointers in a VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions). If the bit is 0, these addresses are limited to the processor's physical-address width.² If the bit is 1, these addresses are limited to 32 bits. This bit is always 0 for processors that support Intel 64 architecture.
- If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 33.15 for details of this treatment.
- Bits 53:50 report the memory type that should be used for the VMCS, for data structures referenced by pointers in the VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions), and for the MSEG header. If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported bits 53:50 in this MSR.³

As of this writing, all processors that support VMX operation indicate the write-back type. The values used are given in Table A-1.

Table A-1. Memory Types Recommended for VMCS and Related Data Structures

Value(s)	Field
0	Uncacheable (UC)
1-5	Not used
6	Write Back (WB)

1. Earlier versions of this manual specified that the VMCS revision identifier was a 32-bit field in bits 31:0 of this MSR. For all processors produced prior to this change, bit 31 of this MSR was read as 0.
2. On processors that support Intel 64 architecture, the pointer must not set bits beyond the processor's physical address width. This width is denoted `MAXPHYADDR` and is derived from the value enumerated in `CPUID.80000008H:EAX[7:0]`. If `IA32_TME_ACTIVATE[0] = 1` (indicating that TME has been configured), `MAXPHYADDR` is reduced by the value of `IA32_TME_ACTIVAT[39:36]` when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34); the value is not reduced in SEAM.
3. Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer.

Table A-1. Memory Types Recommended for VMCS and Related Data Structures (Contd.)

Value(s)	Field
7-15	Not used

- If bit 54 is read as 1, the processor reports information in the VM-exit instruction-information field on VM exits due to execution of the INS and OUTS instructions (see Section 29.2.5). This reporting is done only if this bit is read as 1.
- Bit 55 is read as 1 if any VMX controls that default to 1 may be cleared to 0. See Appendix A.2 for details. It also reports support for the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3.1, Appendix A.3.2, Appendix A.4, and Appendix A.5 for details.
- If bit 56 is read as 1, software can use VM entry to deliver a hardware exception with or without an error code, regardless of vector (see Section 28.2.1.3).
- The values of bits 47:45 and bits 63:57 are reserved and are read as 0.

A.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 28, “VM Entries,” certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**. Software can discover the default setting of a reserved control by consulting the appropriate VMX capability MSR (see Appendix A.3 through Appendix A.5).

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control’s new functionality should set the control to its default setting. For that reason, it is useful for software to know the default settings of the reserved controls.

Default settings partition the various controls into the following classes:

- **Always-flexible.** These have never been reserved.
- **Default0.** These are (or have been) reserved with a default setting of 0.
- **Default1.** They are (or have been) reserved with a default setting of 1.

As noted in Appendix A.1, a logical processor uses bit 55 of the IA32_VMX_BASIC MSR to indicate whether any of the default1 controls may be 0:

- If bit 55 of the IA32_VMX_BASIC MSR is read as 0, all the default1 controls are reserved and must be 1. VM entry will fail if any of these controls are 0 (see Section 28.2.1).
- If bit 55 of the IA32_VMX_BASIC MSR is read as 1, not all the default1 controls are reserved, and some (but not necessarily all) may be 0. The CPU supports four (4) new VMX capability MSRs: IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3 through Appendix A.5 for details. (These MSRs are not supported if bit 55 of the IA32_VMX_BASIC MSR is read as 0.)

A.3 VM-EXECUTION CONTROLS

There are separate capability MSRs for the pin-based VM-execution controls, the primary processor-based VM-execution controls, the secondary processor-based VM-execution controls, and the tertiary processor-based VM-execution controls. These are described in Appendix A.3.1, Appendix A.3.2, Appendix A.3.3, and Appendix A.3.4, respectively.

A.3.1 Pin-Based VM-Execution Controls

The IA32_VMX_PINBASED_CTLS MSR (index 481H) reports on the allowed settings of **most** of the pin-based VM-execution controls (see Section 26.6.1):

- Bits 31:0 indicate the **allowed 0-settings** of these controls. VM entry allows control X (bit X of the pin-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the pin-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 2, and 4; the corresponding bits of the IA32_VMX_PINBASED_CTLMSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any pin-based VM-execution control in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLMSR (see below) reports which of the pin-based VM-execution controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the **allowed 1-settings** of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLMSR (index 48DH) reports on the allowed settings of **all** of the pin-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the pin-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_PINBASED_CTLMSR. (The IA32_VMX_TRUE_PINBASED_CTLMSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLMSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 2, and 4, there is no need for software to consult the IA32_VMX_PINBASED_CTLMSR.

A.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLMSR (index 482H) reports on the allowed settings of **most** of the primary processor-based VM-execution controls (see Section 26.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary processor-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary processor-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 4–6, 8, 13–16, and 26; the corresponding bits of the IA32_VMX_PROCBASED_CTLMSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any of the primary processor-based VM-execution controls in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLMSR (see below) reports which of the primary processor-based VM-execution controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLMSR (index 48EH) reports on the allowed settings of **all** of the primary processor-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary processor-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_PROCBASED_CTLMSR. (The IA32_VMX_TRUE_PROCBASED_CTLMSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLMSR. Assuming that software knows that the default1 class of primary processor-based VM-execution controls contains bits 1, 4–6, 8, 13–16, and 26, there is no need for software to consult the IA32_VMX_PROCBASED_CTLMSR.

A.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLMSR2 (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 26.6.2). The following items provide details, including enforcement by VM entry:

- Bits 31:0 indicate the allowed 0-settings of these controls. These bits are always 0. This fact indicates that VM entry allows each bit of the secondary processor-based VM-execution controls to be 0 (reserved bits must be 0)
- Bits 63:32 indicate the allowed 1-settings of these controls; the 1-setting is not allowed for any reserved bit. VM entry allows control X (bit X of the secondary processor-based VM-execution controls) to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X and the “activate secondary controls” primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLMSR2 MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLMSR is 1).

A.3.4 Tertiary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLMSR3 (index 492H) reports on the allowed 1-settings of the tertiary processor-based VM-execution controls (see Section 26.6.2); the 1-setting is not allowed for any reserved bit.

VM entry allows control X (bit X of the tertiary processor-based VM-execution controls) to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLMSR3 MSR exists only on processors that support the 1-setting of the “activate tertiary controls” VM-execution control (only if bit 49 of the IA32_VMX_PROCBASED_CTLMSR is 1).

Notice that the organization of this MSR differs from that of IA32_VMX_PROCBASED_CTLMSR2 (Appendix A.3.3). This is because there are 64 tertiary processor-based VM-execution controls, while there were only 32 secondary processor-based VM-execution controls.

A.4 VM-EXIT CONTROLS

There are separate capability MSRs for the primary VM-exit controls and the secondary VM-exit controls. These are described in Appendix A.4.1 and Appendix A.4.2, respectively.

A.4.1 Primary VM-Exit Controls

The IA32_VMX_EXIT_CTLMSR (index 483H) reports on the allowed settings of **most** of the primary VM-exit controls (see Section 26.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary VM-exit controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary VM-exit controls in the default1 class (see Appendix A.2). These are bits 0–8, 10, 11, 13, 14, 16, and 17; the corresponding bits of the IA32_VMX_EXIT_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any primary VM-exit control in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (see below) reports which of the primary VM-exit controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (index 48FH) reports on the allowed settings of **all** of the primary VM-exit controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary VM-exit controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary VM-exit controls is contained in the IA32_VMX_EXIT_CTLS MSR. (The IA32_VMX_TRUE_EXIT_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the primary VM-exit controls is contained in the IA32_VMX_TRUE_EXIT_CTLS MSR. Assuming that software knows that the default1 class of primary VM-exit controls contains bits 0–8, 10, 11, 13, 14, 16, and 17, there is no need for software to consult the IA32_VMX_EXIT_CTLS MSR.

A.4.2 Secondary VM-Exit Controls

The IA32_VMX_EXIT_CTLS2 MSR (index 493H) reports on the allowed 1-settings of the secondary VM-exit controls (see Section 26.7.1); the 1-setting is not allowed for any reserved bit.

VM entry allows control X (bit X of the secondary VM-exit controls) to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if control X and the “activate secondary controls” primary VM-exit control are both 1.

The IA32_VMX_EXIT_CTLS2 MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-exit control (only if bit 63 of the IA32_VMX_EXIT_CTLS MSR is 1).

Notice that the organization of this MSR differs from that of IA32_VMX_EXIT_CTLS (Appendix A.4.1). This is because there are 64 secondary VM-exit controls, while there were only 32 primary VM-exit controls.

A.5 VM-ENTRY CONTROLS

The IA32_VMX_ENTRY_CTLS MSR (index 484H) reports on the allowed settings of **most** of the VM-entry controls (see Section 26.8.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-entry controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. Exceptions are made for the VM-entry controls in the default1 class (see Appendix A.2). These are bits 0–8 and 12; the corresponding bits of the IA32_VMX_ENTRY_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-entry control in the default1 class is 0.
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLMS MSR (see below) reports which of the VM-entry controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLMS MSR (index 490H) reports on the allowed settings of **all** of the VM-entry controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-entry controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_ENTRY_CTLMS MSR. (The IA32_VMX_TRUE_ENTRY_CTLMS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLMS MSR. Assuming that software knows that the default1 class of VM-entry controls contains bits 0–8 and 12, there is no need for software to consult the IA32_VMX_ENTRY_CTLMS MSR.

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
- If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control; see Section 29.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.
- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
 - Bit 6 reports (if set) the support for activity state 1 (HLT).
 - Bit 7 reports (if set) the support for activity state 2 (shutdown).
 - Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).

- If bit 14 is read as 1, Intel® Processor Trace (Intel PT) can be used in VMX operation. If the processor supports Intel PT but does not allow it to be used in VMX operation, execution of VMXON clears IA32_RTIT_CTL.TraceEn (see “VMXON—Enter VMX Operation” in Chapter 32); any attempt to write IA32_RTIT_CTL while in VMX operation (including VMX root operation) causes a general-protection exception.
- If bit 15 is read as 1, the RDMSR instruction can be used in system-management mode (SMM) to read the IA32_SMBASE MSR (MSR address 9EH). See Section 33.15.6.3.
- Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).
- Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of IA32_VMX_MISC is N, then $512 * (N + 1)$ is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).

- If bit 28 is read as 1, bit 2 of the `IA32_SMM_MONITOR_CTL` can be set to 1. `VMXOFF` unblocks SMIs unless `IA32_SMM_MONITOR_CTL[bit 2]` is 1 (see Section 33.14.4).
- If bit 29 is read as 1, software can use `VMWRITE` to write to any supported field in the VMCS; otherwise, `VMWRITE` cannot be used to modify VM-exit information fields.
- If bit 30 is read as 1, VM entry allows injection of a software interrupt, software exception, or privileged software exception with an instruction length of 0.
- Bits 63:32 report the 32-bit MSEG revision identifier used by the processor.
- Bits 13:9 and bit 31 are reserved and are read as 0.

A.7 VMX-FIXED BITS IN CR0

The `IA32_VMX_CR0_FIXED0` MSR (index 486H) and `IA32_VMX_CR0_FIXED1` MSR (index 487H) indicate how bits in CR0 may be set in VMX operation. They report on bits in CR0 that are allowed to be 0 and to be 1, respectively, in VMX operation. If bit X is 1 in `IA32_VMX_CR0_FIXED0`, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in `IA32_VMX_CR0_FIXED1`, then that bit of CR0 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in `IA32_VMX_CR0_FIXED0`, then that bit is also 1 in `IA32_VMX_CR0_FIXED1`; if bit X is 0 in `IA32_VMX_CR0_FIXED1`, then that bit is also 0 in `IA32_VMX_CR0_FIXED0`. Thus, each bit in CR0 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in `IA32_VMX_CR0_FIXED0` and 1 in `IA32_VMX_CR0_FIXED1`).

A.8 VMX-FIXED BITS IN CR4

The `IA32_VMX_CR4_FIXED0` MSR (index 488H) and `IA32_VMX_CR4_FIXED1` MSR (index 489H) indicate how bits in CR4 may be set in VMX operation. They report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in `IA32_VMX_CR4_FIXED0`, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in `IA32_VMX_CR4_FIXED1`, then that bit of CR4 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in `IA32_VMX_CR4_FIXED0`, then that bit is also 1 in `IA32_VMX_CR4_FIXED1`; if bit X is 0 in `IA32_VMX_CR4_FIXED1`, then that bit is also 0 in `IA32_VMX_CR4_FIXED0`. Thus, each bit in CR4 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in `IA32_VMX_CR4_FIXED0` and 1 in `IA32_VMX_CR4_FIXED1`).

A.9 VMCS ENUMERATION

The `IA32_VMX_VMCS_ENUM` MSR (index 48AH) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 26.11.2, each field in the VMCS is associated with a 32-bit encoding which is structured as follows:

- Bits 31:15 are reserved (must be 0).
- Bits 14:13 indicate the field's width.
- Bit 12 is reserved (must be 0).
- Bits 11:10 indicate the field's type.
- Bits 9:1 is an index field that distinguishes different fields with the same width and type.
- Bit 0 indicates access type.

`IA32_VMX_VMCS_ENUM` indicates to software the highest index value used in the encoding of any field supported by the processor:

- Bits 9:1 contain the highest index value used for any VMCS encoding.
- Bit 0 and bits 63:10 are reserved and are read as 0.

A.10 VPID AND EPT CAPABILITIES

The IA32_VMX_EPT_VPID_CAP MSR (index 48CH) reports information about the capabilities of the logical processor with regard to virtual-processor identifiers (VPIDs, Section 30.1) and extended page tables (EPT, Section 30.3):

- If bit 0 is read as 1, the processor supports execute-only translations by EPT. This support allows software to configure EPT paging-structure entries in which bits 1:0 are clear (indicating that data accesses are not allowed) and bit 2 is set (indicating that instruction fetches are allowed).¹
- Bit 6 indicates support for a page-walk length of 4.
- Bit 7 indicates support for a page-walk length of 5.
- If bit 8 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be uncacheable (UC); see Section 26.6.11.
- If bit 14 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be write-back (WB).
- If bit 16 is read as 1, the logical processor allows software to configure a EPT PDE to map a 2-Mbyte page (by setting bit 7 in the EPT PDE).
- If bit 17 is read as 1, the logical processor allows software to configure a EPT PDPTE to map a 1-Gbyte page (by setting bit 7 in the EPT PDPTE).
- Support for the INVEPT instruction (see Chapter 32 and Section 30.4.3.1):
 - If bit 20 is read as 1, the INVEPT instruction is supported.
 - If bit 25 is read as 1, the single-context INVEPT type is supported.
 - If bit 26 is read as 1, the all-context INVEPT type is supported.
- If bit 21 is read as 1, accessed and dirty flags for EPT are supported (see Section 30.3.5).
- If bit 22 is read as 1, the processor reports advanced VM-exit information for EPT violations (see Section 29.2.1). This reporting is done only if this bit is read as 1.
- If bit 23 is read as 1, supervisor shadow-stack control is supported (see Section 30.3.3.2).
- Support for the INVVPID instruction (see Chapter 32 and Section 30.4.3.1):
 - If bit 32 is read as 1, the INVVPID instruction is supported.
 - If bit 40 is read as 1, the individual-address INVVPID type is supported.
 - If bit 41 is read as 1, the single-context INVVPID type is supported.
 - If bit 42 is read as 1, the all-context INVVPID type is supported.
 - If bit 43 is read as 1, the single-context-retaining-globals INVVPID type is supported.
- Bits 53:48 enumerate the maximum HLAT prefix size. It is expected that any processor that supports the 1-setting of the “enable HLAT” VM-execution control will enumerate this value as 1. See Section 5.5.1.
- Bits 5:1, bits 13:9, bit 15, bits 19:18, bit 24, bits 31:27, bits 39:33, bits 47:44, and bits 63:54 are reserved and are read as 0.

The IA32_VMX_EPT_VPID_CAP MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLMSR is 1) and that support either the 1-setting of the “enable EPT” VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLMSR2 is 1) or the 1-setting of the “enable VPID” VM-execution control (only if bit 37 of the IA32_VMX_PROCBASED_CTLMSR2 is 1).

1. If the “mode-based execute control for EPT” VM-execution control is 1, setting bit 0 indicates also that software may also configure EPT paging-structure entries in which bits 1:0 are both clear and in which bit 10 is set (indicating a translation that can be used to fetch instructions from a supervisor-mode linear address or a user-mode linear address).

A.11 VM FUNCTIONS

The IA32_VMX_VMFUNC MSR (index 491H) reports on the allowed settings of the VM-function controls (see Section 26.6.14). VM entry allows bit X of the VM-function controls to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if bit X of the VM-function controls, the “activate secondary controls” primary processor-based VM-execution control, and the “enable VM functions” secondary processor-based VM-execution control are all 1.

The IA32_VMX_VMFUNC MSR exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTL1 MSR is 1) and the 1-setting of the “enable VM functions” secondary processor-based VM-execution control (only if bit 45 of the IA32_VMX_PROCBASED_CTL2 MSR is 1).

31. Updates Appendix B, Volume 3D

Change bars and violet text show changes to Appendix B of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Added SEAM shared EPT pointer to Table B-4 in Section B.2.1, "64-Bit Control Fields".
- Added SEAM-guest KeyID to Table B-8 in Section B.3.1, "32-Bit Control Fields".

APPENDIX B FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 26.11.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.).

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 26.11.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-1 enumerates the 16-bit control fields.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_0xx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID) ¹	000000000B	00000000H
Posted-interrupt notification vector ²	000000001B	00000002H
EPTP index ³	000000010B	00000004H
HLAT prefix size ⁴	000000011B	00000006H
Last PID-pointer index ⁵	000000100B	00000008H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable VPID” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “IPI virtualization” VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_0xx0B)

Field Name	Index	Encoding
Guest ES selector	000000000B	00000800H
Guest CS selector	000000001B	00000802H
Guest SS selector	000000010B	00000804H
Guest DS selector	000000011B	00000806H
Guest FS selector	000000100B	00000808H

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH
Guest TR selector	000000111B	0000080EH
Guest interrupt status ¹	000001000B	00000810H
PML index ²	000001001B	00000812H
Guest UINV ³	000001010B	00000814H

NOTES:

1. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of either the “clear UINV” VM-exit control or the “load UINV” VM-entry control.

B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 26.11.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)		00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)		00002003H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Address of MSR bitmaps (full) ¹	000000010B	00002004H
Address of MSR bitmaps (high) ¹		00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)		00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)		00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)		0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)		0000200DH
PML address (full) ²	000000111B	0000200EH
PML address (high) ²		0000200FH
TSC offset (full)	000001000B	00002010H
TSC offset (high)		00002011H
Virtual-APIC address (full) ³	000001001B	00002012H
Virtual-APIC address (high) ³		00002013H
APIC-access address (full) ⁴	000001010B	00002014H
APIC-access address (high) ⁴		00002015H
Posted-interrupt descriptor address (full) ⁵	000001011B	00002016H
Posted-interrupt descriptor address (high) ⁵		00002017H
VM-function controls (full) ⁶	000001100B	00002018H
VM-function controls (high) ⁶		00002019H
EPT pointer (EPTP; full) ⁷	000001101B	0000201AH
EPT pointer (EPTP; high) ⁷		0000201BH
EOI-exit bitmap 0 (EOI_EXIT0; full) ⁸	000001110B	0000201CH
EOI-exit bitmap 0 (EOI_EXIT0; high) ⁸		0000201DH
EOI-exit bitmap 1 (EOI_EXIT1; full) ⁸	000001111B	0000201EH
EOI-exit bitmap 1 (EOI_EXIT1; high) ⁸		0000201FH
EOI-exit bitmap 2 (EOI_EXIT2; full) ⁸	000010000B	00002020H
EOI-exit bitmap 2 (EOI_EXIT2; high) ⁸		00002021H
EOI-exit bitmap 3 (EOI_EXIT3; full) ⁸	000010001B	00002022H
EOI-exit bitmap 3 (EOI_EXIT3; high) ⁸		00002023H
EPTP-list address (full) ⁹	000010010B	00002024H
EPTP-list address (high) ⁹		00002025H
VMREAD-bitmap address (full) ¹⁰	000010011B	00002026H
VMREAD-bitmap address (high) ¹⁰		00002027H
VMWRITE-bitmap address (full) ¹⁰	000010100B	00002028H
VMWRITE-bitmap address (high) ¹⁰		00002029H

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Virtualization-exception information address (full) ¹¹	000010101B	0000202AH
Virtualization-exception information address (high) ¹¹		0000202BH
XSS-exiting bitmap (full) ¹²	000010110B	0000202CH
XSS-exiting bitmap (high) ¹²		0000202DH
ENCLS-exiting bitmap (full) ¹³	000010111B	0000202EH
ENCLS-exiting bitmap (high) ¹³		0000202FH
Sub-page-permission-table pointer (full) ¹⁴	000011000B	00002030H
Sub-page-permission-table pointer (high) ¹⁴		00002031H
TSC multiplier (full) ¹⁵	000011001B	00002032H
TSC multiplier (high) ¹⁵		00002033H
Tertiary processor-based VM-execution controls (full) ¹⁶	000011010B	00002034H
Tertiary processor-based VM-execution controls (high) ¹⁶		00002035H
Low PASID directory address (full) ¹⁷	000011100B	00002038H
Low PASID directory address (high) ¹⁷		00002039H
High PASID directory address (full) ¹⁷	000011101B	0000203AH
High PASID directory address (high) ¹⁷		0000203BH
SEAM shared EPT pointer (full) ¹⁸	000011110B	0000203CH
SEAM shared EPT pointer (high) ¹⁸		0000203DH
PCONFIG-exiting bitmap (full) ¹⁹	000011111B	0000203EH
PCONFIG-exiting bitmap (high) ¹⁹		0000203FH
Hypervisor-managed linear-address translation pointer (HLATP; full) ²⁰	000100000B	00002040H
HLATP (high) ²⁰		00002041H
PID-pointer table address (full) ²¹	000100001B	00002042H
PID-pointer table address (high) ²¹		00002043H
Secondary VM-exit controls (full) ²²	000100010B	00002044H
Secondary VM-exit controls (high) ²²		00002045H
IA32_SPEC_CTRL mask (full) ²³	000100101B	0000204AH
IA32_SPEC_CTRL mask (high) ²³		0000204BH
IA32_SPEC_CTRL shadow (full) ²³	000100110B	0000204CH
IA32_SPEC_CTRL shadow (high) ²³		0000204DH

NOTES:

1. This field exists only on processors that support the 1-setting of the “use MSR bitmaps” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable PML” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “virtualize APIC accesses” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “process posted interrupts” VM-execution control.
6. This field exists only on processors that support the 1-setting of the “enable VM functions” VM-execution control.
7. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
8. This field exists only on processors that support the 1-setting of the “virtual-interrupt delivery” VM-execution control.
9. This field exists only on processors that support the 1-setting of the “EPTP switching” VM-function control.

10. This field exists only on processors that support the 1-setting of the “VMCS shadowing” VM-execution control.
11. This field exists only on processors that support the 1-setting of the “EPT-violation #VE” VM-execution control.
12. This field exists only on processors that support the 1-setting of the “enable XSAVES/XRSTORS” VM-execution control.
13. This field exists only on processors that support the 1-setting of the “enable ENCLS exiting” VM-execution control.
14. This field exists only on processors that support the 1-setting of the “sub-page write permissions for EPT” VM-execution control.
15. This field exists only on processors that support the 1-setting of the “use TSC scaling” VM-execution control.
16. This field exists only on processors that support the 1-setting of the “activate tertiary controls” VM-execution control.
17. This field exists only on processors that support the 1-setting of the “PASID translation” VM-execution control.
18. This field exists only on processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control.
19. This field exists only on processors that support the 1-setting of the “enable PCONFIG” VM-execution control.
20. This field exists only on processors that support the 1-setting of the “enable HLAT” VM-execution control.
21. This field exists only on processors that support the 1-setting of the “IPI virtualization” VM-execution control.
22. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
23. This field exists only on processors that support the 1-setting of the “virtualize IA32_SPEC_CTRL” VM-execution control.

B.2.2 64-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-5 enumerates the 64-bit read-only data fields.

Table B-5. Encodings for 64-Bit Read-Only Data Fields (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full) ¹	000000000B	00002400H
Guest-physical address (high) ¹		00002401H
MSR data (full) ²	000000001B	00002402H
MSR data (high) ²		00002403H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable EPT” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “enable MSR-list instructions” VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)		00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H
Guest IA32_DEBUGCTL (high)		00002803H
Guest IA32_PAT (full) ¹	000000010B	00002804H
Guest IA32_PAT (high) ¹		00002805H
Guest IA32_EFER (full) ²	000000011B	00002806H
Guest IA32_EFER (high) ²		00002807H

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Guest IA32_PERF_GLOBAL_CTRL (full) ³	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high) ³		00002809H
Guest PDPTE0 (full) ⁴	000000101B	0000280AH
Guest PDPTE0 (high) ⁴		0000280BH
Guest PDPTE1 (full) ⁴	000000110B	0000280CH
Guest PDPTE1 (high) ⁴		0000280DH
Guest PDPTE2 (full) ⁴	000000111B	0000280EH
Guest PDPTE2 (high) ⁴		0000280FH
Guest PDPTE3 (full) ⁴	000001000B	00002810H
Guest PDPTE3 (high) ⁴		00002811H
Guest IA32_BNDCFGS (full) ⁵	000001001B	00002812H
Guest IA32_BNDCFGS (high) ⁵		00002813H
Guest IA32_RTIT_CTL (full) ⁶	000001010B	00002814H
Guest IA32_RTIT_CTL (high) ⁶		00002815H
Guest IA32_LBR_CTL (full) ⁷	000001011B	00002816H
Guest IA32_LBR_CTL (high) ⁷		00002817H
Guest IA32_PKRS (full) ⁸	000001100B	00002818H
Guest IA32_PKRS (high) ⁸		00002819H

NOTES:

1. This field exists only on processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
2. This field exists only on processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-entry control.
4. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.
5. This field exists only on processors that support either the 1-setting of the "load IA32_BNDCFGS" VM-entry control or that of the "clear IA32_BNDCFGS" VM-exit control.
6. This field exists only on processors that support either the 1-setting of the "load IA32_RTIT_CTL" VM-entry control or that of the "clear IA32_RTIT_CTL" VM-exit control.
7. This field exists only on processors that support either the 1-setting of the "load IA32_LBR_CTL" VM-entry control or that of the "clear IA32_LBR_CTL" VM-exit control.
8. This field exists only on processors that support the 1-setting of the "load PKRS" VM-entry control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full) ¹	000000000B	00002C00H
Host IA32_PAT (high) ¹		00002C01H

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb) (Contd.)

Field Name	Index	Encoding
Host IA32_EFER (full) ²	000000001B	00002C02H
Host IA32_EFER (high) ²		00002C03H
Host IA32_PERF_GLOBAL_CTRL (full) ³	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high) ³		00002C05H
Host IA32_PKRS (full) ⁴	000000011B	00002C06H
Host IA32_PKRS (high) ⁴		00002C07H

NOTES:

1. This field exists only on processors that support the 1-setting of the "load IA32_PAT" VM-exit control.
2. This field exists only on processors that support the 1-setting of the "load IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-exit control.
4. This field exists only on processors that support the 1-setting of the "load PKRS" VM-exit control.

B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 26.11.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	000000000B	00004000H
Primary processor-based VM-execution controls	000000001B	00004002H
Exception bitmap	000000010B	00004004H
Page-fault error-code mask	000000011B	00004006H
Page-fault error-code match	000000100B	00004008H
CR3-target count	000000101B	0000400AH
Primary VM-exit controls	000000110B	0000400CH
VM-exit MSR-store count	000000111B	0000400EH
VM-exit MSR-load count	000001000B	00004010H
VM-entry controls	000001001B	00004012H
VM-entry MSR-load count	000001010B	00004014H
VM-entry interruption-information field	000001011B	00004016H
VM-entry exception error code	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold ¹	000001110B	0000401CH
Secondary processor-based VM-execution controls ²	000001111B	0000401EH
PLE_Gap ³	000010000B	00004020H

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
PLE_Window ³	000010001B	00004022H
Instruction-timeout control ⁴	000010010B	00004024H
SEAM-guest KeyID ⁵	000010011B	00004026H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control.
4. This field exists only on processors that support the 1-setting of the “instruction timeout” VM-execution control.
5. This field exists only on processors that support the 1-setting of the “SEAM guest-physical address width” VM-execution control.

B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	000000000B	00004400H
Exit reason	000000001B	00004402H
VM-exit interruption information	000000010B	00004404H
VM-exit interruption error code	000000011B	00004406H
IDT-vectoring information field	000000100B	00004408H
IDT-vectoring error code	000000101B	0000440AH
VM-exit instruction length	000000110B	0000440CH
VM-exit instruction information	000000111B	0000440EH

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	000000000B	00004800H
Guest CS limit	000000001B	00004802H
Guest SS limit	000000010B	00004804H
Guest DS limit	000000011B	00004806H
Guest FS limit	000000100B	00004808H
Guest GS limit	000000101B	0000480AH
Guest LDTR limit	000000110B	0000480CH
Guest TR limit	000000111B	0000480EH
Guest GDTR limit	000001000B	00004810H

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest IDTR limit	000001001B	00004812H
Guest ES access rights	000001010B	00004814H
Guest CS access rights	000001011B	00004816H
Guest SS access rights	000001100B	00004818H
Guest DS access rights	000001101B	0000481AH
Guest FS access rights	000001110B	0000481CH
Guest GS access rights	000001111B	0000481EH
Guest LDTR access rights	000010000B	00004820H
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value ¹	000010111B	0000482EH

NOTES:

1. This field exists only on processors that support the 1-setting of the “activate VMX-preemption timer” VM-execution control.

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 26.11.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
CR0 read shadow	000000010B	00006004H
CR4 read shadow	000000011B	00006006H
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3 ¹	000000111B	0000600EH

NOTES:

1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	000000000B	00006400H
I/O RCX	000000001B	00006402H
I/O RSI	000000010B	00006404H
I/O RDI	000000011B	00006406H
I/O RIP	000000100B	00006408H
Guest-linear address	000000101B	0000640AH

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	000000000B	00006800H
Guest CR3	000000001B	00006802H
Guest CR4	000000010B	00006804H
Guest ES base	000000011B	00006806H
Guest CS base	000000100B	00006808H
Guest SS base	000000101B	0000680AH
Guest DS base	000000110B	0000680CH
Guest FS base	000000111B	0000680EH
Guest GS base	000001000B	00006810H
Guest LDTR base	000001001B	00006812H

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest TR base	000001010B	00006814H
Guest GDTR base	000001011B	00006816H
Guest IDTR base	000001100B	00006818H
Guest DR7	000001101B	0000681AH
Guest RSP	000001110B	0000681CH
Guest RIP	000001111B	0000681EH
Guest RFLAGS	000010000B	00006820H
Guest pending debug exceptions	000010001B	00006822H
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H
Guest IA32_S_CET ¹	000010100B	00006828H
Guest SSP ¹	000010101B	0000682AH
Guest IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000010110B	0000682CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-entry control.

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	000000000B	00006C00H
Host CR3	000000001B	00006C02H
Host CR4	000000010B	00006C04H
Host FS base	000000011B	00006C06H
Host GS base	000000100B	00006C08H
Host TR base	000000101B	00006C0AH
Host GDTR base	000000110B	00006C0CH
Host IDTR base	000000111B	00006C0EH
Host IA32_SYSENTER_ESP	000001000B	00006C10H
Host IA32_SYSENTER_EIP	000001001B	00006C12H
Host RSP	000001010B	00006C14H
Host RIP	000001011B	00006C16H
Host IA32_S_CET ¹	000001100B	00006C18H
Host SSP ¹	000001101B	00006C1AH

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Host IA32_INTERRUPT_SSP_TABLE_ADDR ¹	000001110B	00006C1CH

NOTES:

1. This field is supported only on processors that support the 1-setting of the “load CET state” VM-exit control.

32. Updates Appendix C, Volume 3D

Change bars and violet text show changes to Appendix C of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3D: System Programming Guide, Part 4*.

Changes to this chapter:

- Added UDB to Basic Exit Reason 0 in Table C-1.

APPENDIX C

VMX BASIC EXIT REASONS

Every VM exit writes a 32-bit exit reason to the VMCS (see Section 26.9.1). Certain VM-entry failures also do this (see Section 28.8). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table C-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

Table C-1. Basic Exit Reasons

Basic Exit Reason	Description
0	Exception or non-maskable interrupt (NMI). Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. This case includes executions of BOUND that cause #BR, executions of INT1 (they cause #DB), executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD0, UD1, UD2, and UDB (they cause #UD). 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1.
1	External interrupt. An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	Triple fault. The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	INIT signal. An INIT signal arrived
4	Start-up IPI (SIPI). A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	I/O system-management interrupt (SMI). An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 33.15.2).
6	Other SMI. An SMI arrived and caused an SMM VM exit (see Section 33.15.2) but not immediately after retirement of an I/O instruction.
7	Interrupt window. At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	NMI window. At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	Task switch. Guest software attempted a task switch.
10	CPUID. Guest software attempted to execute CPUID.
11	GETSEC. Guest software attempted to execute GETSEC.
12	HLT. Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.
13	INVD. Guest software attempted to execute INVD.
14	INVLPG. Guest software attempted to execute INVLPG and the "INVLPG exiting" VM-execution control was 1.
15	RDPMC. Guest software attempted to execute RDPMC and the "RDPMC exiting" VM-execution control was 1.
16	RDTSC. Guest software attempted to execute RDTSC and the "RDTSC exiting" VM-execution control was 1.
17	RSM. Guest software attempted to execute RSM in SMM.
18	VMCALL. VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 33.15.2).
19	VMCLEAR. Guest software attempted to execute VMCLEAR.
20	VMLAUNCH. Guest software attempted to execute VMLAUNCH.
21	VMPTRLD. Guest software attempted to execute VMPTRLD.
22	VMPTRST. Guest software attempted to execute VMPTRST.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
23	VMREAD. Guest software attempted to execute VMREAD.
24	VMRESUME. Guest software attempted to execute VMRESUME.
25	VMWRITE. Guest software attempted to execute VMWRITE.
26	VMXOFF. Guest software attempted to execute VMXOFF.
27	VMXON. Guest software attempted to execute VMXON.
28	Control-register accesses. Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 27.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1. Such VM exits instead use basic exit reason 43.
29	MOV DR. Guest software attempted a MOV to or from a debug register and the “MOV-DR exiting” VM-execution control was 1.
30	I/O instruction. Guest software attempted to execute an I/O instruction and either: 1: The “use I/O bitmaps” VM-execution control was 0 and the “unconditional I/O exiting” VM-execution control was 1. 2: The “use I/O bitmaps” VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.
31	RDMSR. Guest software attempted to execute RDMSR and one of the following holds: ▪ The “use MSR bitmaps” VM-execution control was 0. ▪ The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. ▪ The value of RCX is in the range 00000000H – 00001FFFH and the n^{th} bit in read bitmap for low MSRs is 1, where n is the value of RCX. ▪ The value of RCX is in the range C0000000H – C0001FFFH and the n^{th} bit in read bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
32	WRMSR or WRMSRNS. Guest software attempted to execute either WRMSR or WRMSRNS and either: 1: The “use MSR bitmaps” VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX is in the range 00000000H – 00001FFFH and the n^{th} bit in write bitmap for low MSRs is 1, where n is the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the n^{th} bit in write bitmap for high MSRs is 1, where n is the value of RCX & 00001FFFH.
33	VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 28.3.1.
34	VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs. See Section 28.4.
36	MWAIT. Guest software attempted to execute MWAIT and the “MWAIT exiting” VM-execution control was 1.
37	Monitor trap flag. A VM exit occurred due to the 1-setting of the “monitor trap flag” VM-execution control (see Section 27.5.2) or VM entry injected a pending MTF VM exit as part of VM entry (see Section 28.6.2).
39	MONITOR. Guest software attempted to execute MONITOR and the “MONITOR exiting” VM-execution control was 1.
40	PAUSE. Either guest software attempted to execute PAUSE and the “PAUSE exiting” VM-execution control was 1 or the “PAUSE-loop exiting” VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 27.1.3).
41	VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 28.9).
43	TPR below threshold. The logical processor determined that the value of bits 7:4 of the byte at offset 080H on the virtual-APIC page was below that of the TPR threshold VM-execution control field while the “use TPR shadow” VM-execution control was 1 either as part of TPR virtualization (Section 31.1.2) or VM entry (Section 28.7.7).
44	APIC access. Guest software attempted to access memory at a physical address on the APIC-access page and the “virtualize APIC accesses” VM-execution control was 1 (see Section 31.4).
45	Virtualized EOI. EOI virtualization was performed for a virtual interrupt whose vector indexed a bit set in the EOI-exit bitmap.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
46	Access to GDTR or IDTR. Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the “descriptor-table exiting” VM-execution control was 1.
47	Access to LDTR or TR. Guest software attempted to execute LLDT, LTR, SLDT, or STR and the “descriptor-table exiting” VM-execution control was 1.
48	EPT violation. An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	EPT misconfiguration. An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	INVEPT. Guest software attempted to execute INVEPT.
51	RDTSMP. Guest software attempted to execute RDTSMP and the “enable RDTSMP” and “RDTSMP exiting” VM-execution controls were both 1.
52	VMX-preemption timer expired. The preemption timer counted down to zero.
53	INVVPID. Guest software attempted to execute INVVPID.
54	WBINVD or WBNOINVD. Guest software attempted to execute WBINVD or WBNOINVD and the “WBINVD exiting” VM-execution control was 1.
55	XSETBV. Guest software attempted to execute XSETBV.
56	APIC write. Guest software completed a write to the virtual-APIC page that must be virtualized by VMM software (see Section 31.4.3.3).
57	RDRAND. Guest software attempted to execute RDRAND and the “RDRAND exiting” VM-execution control was 1.
58	INVPCID. Guest software attempted to execute INVPCID and the “enable INVPCID” and “INVLPG exiting” VM-execution controls were both 1.
59	VMFUNC. Guest software invoked a VM function with the VMFUNC instruction and the VM function either was not enabled or generated a function-specific condition causing a VM exit.
60	ENCLS. Guest software attempted to execute ENCLS, “enable ENCLS exiting” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the ENCLS-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the ENCLS-exiting bitmap is 1.
61	RDSEED. Guest software attempted to execute RDSEED and the “RDSEED exiting” VM-execution control was 1.
62	Page-modification log full. The processor attempted to create a page-modification log entry and the value of the PML index was not in the range 0–511.
63	XSAVES. Guest software attempted to execute XSAVES, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
64	XRSTORS. Guest software attempted to execute XRSTORS, the “enable XSAVES/XRSTORS” was 1, and a bit was set in the logical-AND of the following three values: EDX:EAX, the IA32_XSS MSR, and the XSS-exiting bitmap.
65	PCONFIG. Guest software attempted to execute PCONFIG, “enable PCONFIG” VM-execution control was 1, and either (1) EAX < 63 and the corresponding bit in the PCONFIG-exiting bitmap is 1; or (2) EAX ≥ 63 and bit 63 in the PCONFIG-exiting bitmap is 1.
66	SPP-related event. The processor attempted to determine an access’s sub-page write permission and encountered an SPP miss or an SPP misconfiguration. See Section 30.3.4.2.
67	UMWAIT. Guest software attempted to execute UMWAIT and the “enable user wait and pause” and “RDTSMP exiting” VM-execution controls were both 1.
68	TPAUSE. Guest software attempted to execute TPAUSE and the “enable user wait and pause” and “RDTSMP exiting” VM-execution controls were both 1.
69	LOADIWKEY. Guest software attempted to execute LOADIWKEY and the “LOADIWKEY exiting” VM-execution control was 1.
72	ENQCMD PASID translation failure. A VM exit occurred during PASID translation because the present bit was clear in a PASID-directory entry, the valid bit was clear in a PASID-table entry, or one of the entries set a reserved bit.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
73	ENQCMDS PASID translation failure. A VM exit occurred during PASID translation because the present bit was clear in a PASID-directory entry, the valid bit was clear in a PASID-table entry, or one of the entries set a reserved bit.
74	Bus lock. The processor asserted a bus lock while the “bus-lock detection” VM-execution control was 1. (Such VM exits will also set bit 26 of the exit-reason field.)
75	Instruction timeout. The “instruction timeout” VM-execution control was 1 and certain operations prevented the processor from reaching an instruction boundary within the amount of time specified by the instruction-timeout control.
76	SEAMCALL. Guest software attempted to execute SEAMCALL.
77	TDCALL. Guest software attempted to execute TDCALL.
78	RDMSRLIST. Guest software attempted to execute RDMSRLIST and either the “use MSR bitmaps” VM-execution control was 0 or any of the following holds for the index an MSR being accessed: ▪ The index is neither in the range 00000000H – 00001FFFFH nor in the range C0000000H – C0001FFFFH. ▪ The index is in the range 00000000H – 00001FFFFH and the n^{th} bit in read bitmap for low MSRs is 1, where n is the index. ▪ The index is in the range C0000000H – C0001FFFFH and the n^{th} bit in read bitmap for high MSRs is 1, where n is the logical AND of the index and the value 00001FFFFH.
79	WRMSRLIST. Guest software attempted to execute WRMSRLIST and either the “use MSR bitmaps” VM-execution control was 0 or any of the following holds for the index an MSR being accessed: ▪ The index is neither in the range 00000000H – 00001FFFFH nor in the range C0000000H – C0001FFFFH. ▪ The index is in the range 00000000H – 00001FFFFH and the n^{th} bit in write bitmap for low MSRs is 1, where n is the index. ▪ The index is in the range C0000000H – C0001FFFFH and the n^{th} bit in write bitmap for high MSRs is 1, where n is the logical AND of the index and the value 00001FFFFH.

33. Updates to Chapter 2, Volume 4

Change bars and violet text show changes to Chapter 2 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 4: Model-Specific Registers*.

Changes to this chapter:

- Removed CPUID Signature DisplayFamily_DisplayModel value of 06_7DH from 10th generation Intel® Core™ processors based on Ice Lake microarchitecture.
- In Section 2.1, added detail for MAXPHYADDR.
- In Table 2-2, the following MSRs were updated:
 - IA32_APIC_BASE (address 1BH)
 - IA32_MKTME_KEYID_PARTITIONING (address 87H)
 - IA32_MTRRCAP (address FEH), bit 15
 - IA32_ARCH_CAPABILITIES (address 10AH), bit 17
 - IA32_MCG_CAP (address 179H), bit 12
 - IA32_MCG_STATUS (address 17AH), bits 3 and 12
 - IA32_RTIT_OUTPUT_BASE (address 560H)
 - IA32_TME_CAPABILITY (address 981H), bits 3 and 28
 - IA32_TME_ACTIVATE (address 982H), bits 7:4, 31, 39:36, and 51
 - IA32_TME_EXCLUDE_MASK (address 983H)
 - IA32_TME_EXCLUDE_BASE (address 984H)
 - IA32_SEAMRR_BASE (address 1400H)
 - IA32_SEAMRR_MASK (address 1401H)
- In Table 2-34, updated MSR IA32_RTIT_OUTPUT_BASE (address 560H).
- In Section 2.17.8, added paragraph for MAXPHYADDR.
- In Table 2-53, Table 2-56, and Table 2-57, updated MSR_CORE_MKTME_ACTIVATE (address 9FFH), bits 39:36.

CHAPTER 2 MODEL-SPECIFIC REGISTERS (MSRS)

This chapter lists MSRs across Intel processor families. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions. The scope of an MSR defines the set of processors that access the same MSR with RDMSR and WRMSR. Thread-scope MSRs are unique to every logical processor. Core-scope MSRs are shared by the threads in the same core; similarly for module-scope, die-scope, and package-scope.

When a processor package contains a single die, die-scope and package-scope are synonymous. When a package contains multiple die, they are distinct.

NOTE

For information on hierarchical level types supported, refer to the CPUID.1FH definition for the actual level type numbers: "V2 Extended Topology Enumeration Leaf" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A. Also see Section 10.9.1, "Hierarchical Mapping of Shared Resources," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A). Table 2-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_BDH	Intel® Series 2 Core™ Ultra processors supporting Lunar Lake performance hybrid architecture
06_ADH, 06_AEH	Intel® Xeon® 6 P-core processors based on Granite Rapids microarchitecture
06_AFH	Intel® Xeon® 6 E-core processors based on Sierra Forest microarchitecture
06_AAH	Intel® Core™ Ultra 7 processors supporting Meteor Lake performance hybrid architecture
06_CFH	5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture
06_8FH	4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture
06_BAH, 06_B7H, 06_BFH	13th generation Intel® Core™ processors supporting Raptor Lake performance hybrid architecture
06_97H, 06_9AH	12th generation Intel® Core™ processors supporting Alder Lake performance hybrid architecture
06_8CH, 06_8DH	11th generation Intel® Core™ processors based on Tiger Lake microarchitecture
06_A7H	11th generation Intel® Core™ processors based on Rocket Lake microarchitecture
06_7EH	10th generation Intel® Core™ processors based on Ice Lake microarchitecture
06_A5H, 06_A6H	10th generation Intel® Core™ processors based on Comet Lake microarchitecture
06_66H	Intel® Core™ processors based on Cannon Lake microarchitecture
06_8EH, 06_9EH	7th generation Intel® Core™ processors based on Kaby Lake microarchitecture, 8th and 9th generation Intel® Core™ processors based on Coffee Lake microarchitecture, Intel® Xeon® E processors based on Coffee Lake microarchitecture
06_6AH, 06_6CH	3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_55H	Intel® Xeon® Scalable Processor Family based on Skylake microarchitecture, 2nd generation Intel® Xeon® Scalable Processor Family based on Cascade Lake product, and 3rd generation Intel® Xeon® Scalable Processor Family based on Cooper Lake product
06_4EH, 06_5EH	6th generation Intel Core processors and Intel Xeon processor E3-1500m v5 product family and E3-1200 v5 product family based on Skylake microarchitecture
06_85H	Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series based on Knights Mill microarchitecture
06_57H	Intel® Xeon Phi™ Processor 3200, 5200, 7200 Series based on Knights Landing microarchitecture
06_56H	Intel Xeon processor D-1500 product family based on Broadwell microarchitecture
06_4FH	Intel Xeon processor E5 v4 Family based on Broadwell microarchitecture, Intel Xeon processor E7 v4 Family, Intel Core i7-69xx Processor Extreme Edition
06_47H	5th generation Intel Core processors, Intel Xeon processor E3-1200 v4 product family based on Broadwell microarchitecture
06_3DH	Intel Core M-5xxx Processor, 5th generation Intel Core processors based on Broadwell microarchitecture
06_3FH	Intel Xeon processor E5-4600/2600/1600 v3 product families, Intel Xeon processor E7 v3 product families based on Haswell-E microarchitecture, Intel Core i7-59xx Processor Extreme Edition
06_3CH, 06_45H, 06_46H	4th Generation Intel Core processor and Intel Xeon processor E3-1200 v3 product family based on Haswell microarchitecture
06_3EH	Intel Xeon processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture
06_3EH	Intel Xeon processor E5-2600/1600 v2 product families and Intel Xeon processor E5-2400 v2 product family based on Ivy Bridge-E microarchitecture, Intel Core i7-49xx Processor Extreme Edition
06_3AH	3rd Generation Intel Core Processor and Intel Xeon processor E3-1200 v2 product family based on Ivy Bridge microarchitecture
06_2DH	Intel Xeon processor E5 Family based on Sandy Bridge microarchitecture, Intel Core i7-39xx Processor Extreme Edition
06_2FH	Intel Xeon Processor E7 Family
06_2AH	Intel Xeon processor E3-1200 product family; 2nd Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5, and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon processor 3400, 3500, 5500 series
06_1DH	Intel Xeon processor MP 7400 series
06_17H	Intel Xeon processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series
06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_86H, 06_96H, 06_9CH	Intel Atom® processors, Intel® Celeron® processors, Intel® Pentium® processors, and Intel® Pentium® Silver processors based on Tremont Microarchitecture
06_7AH	Intel Atom processors based on Goldmont Plus microarchitecture
06_5FH	Intel Atom processors based on Goldmont microarchitecture (Denverton)
06_5CH	Intel Atom processors based on Goldmont microarchitecture

Table 2-1. CPUID Signature Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_4CH	Intel Atom processor X7-Z8000 and X5-Z8000 series based on Airmont microarchitecture
06_5DH	Intel Atom processor X3-C3000 based on Silvermont microarchitecture
06_5AH	Intel Atom processor Z3500 series
06_4AH	Intel Atom processor Z3400 series
06_37H	Intel Atom processor E3000 series, Z3600 series, Z3700 series
06_4DH	Intel Atom processor C2000 series
06_36H	Intel Atom processor S1000 Series
06_1CH, 06_26H, 06_27H, 06_35H, 06_36H	Intel Atom processor family, Intel Atom processor D2000, N2000, E2000, Z2000, C1000 series
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon processor, Intel Xeon processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon processor, Intel Pentium III processor
06_03H, 06_05H	Intel Pentium II Xeon processor, Intel Pentium II processor
06_01H	Intel Pentium Pro processor
05_01H, 05_02H, 05_04H	Intel Pentium processor, Intel Pentium processor with MMX Technology
06_BEH	Intel® Processor and Intel® Core™ i3 and Intel® Core™ 3 N-Series Processors and Intel Atom® x7000 Processor Series based on Gracemont microarchitecture

The Intel® Quark™ SoC X1000 processor can be identified by the signature of DisplayFamily_DisplayModel = 05_09H and SteppingID = 0

2.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 2-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 2-2 and certain bit fields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a model-specific MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 2-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of “DF_DM” (see Table 2-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-2. MAXPHYADDR is derived from the value enumerated in CPUID.80000008H:EAX[7:0] (this width is at most 52). However, if IA32_TME_ACTIVATE[0] = 1 (indicating that TME has been configured), MAXPHYADDR is reduced by the value of IA32_TME_ACTIVATE[39:36] when a logical processor is outside secure arbitration mode (SEAM; see Chapter 34 of the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3); the value is not reduced in SEAM.¹

MSR address range between 40000000H - 4000FFFFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 2-2. IA-32 Architectural MSRs

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
Register Address: 0H, 0		IA32_P5_MC_ADDR (P5_MC_ADDR)	
See Section 2.23, "MSRs in Pentium Processors."			Pentium Processor (05_01H)
Register Address: 1H, 1		IA32_P5_MC_TYPE (P5_MC_TYPE)	
See Section 2.23, "MSRs in Pentium Processors."			DF_DM = 05_01H
Register Address: 6H, 6		IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, "Monitor/Mwait Address Range Determination."			0F_03H
Register Address: 10H, 16		IA32_TIME_STAMP_COUNTER (TSC)	
See Section 19.17, "Time-Stamp Counter."			05_01H
Register Address: 17H, 23		IA32_PLATFORM_ID (MSR_PLATFORM_ID)	
Platform ID (R/O) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.			06_01H
49:0	Reserved.		
52:50	Platform ID (R/O) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7		
63:53	Reserved.		
Register Address: 1BH, 27		IA32_APIC_BASE (APIC_BASE)	
This register holds the APIC base address, permitting the relocation of the APIC memory map. See Section 12.4.4, "Local APIC Status and Location," and Section 12.4.5, "Relocating the Local APIC Registers."			06_01H
7:0	Reserved.		
8	BSP Flag (R/W)		
9	Reserved.		
10	Enable x2APIC mode.		06_1AH
11	APIC Global Enable (R/W)		

1. IA32_TME_ACTIVATE[39:36] is the number of physical-address bits reserved to encode TDX-private key identifiers. This number is never greater than IA32_TME_ACTIVATE[35:32], which is the number physical-address bits used for key identifiers generally.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MAXAPICADDR -1:12	APIC Base (R/W)	MAXAPICADDR is normally CPUID.80000008H:EAX[7:0] for processors that support CPUID.80000008H and 36 otherwise. If IA32_TME_ACTIVATE[39:36] > 0, MAXAPICADDR is reduced by IA32_TME_ACTIVATE[35:32].
63: MAXAPICADDR	Reserved.	
Register Address: 2FH, 47		IA32_BARRIER
IA32_BARRIER (R/O) The IA32_BARRIER MSR ensures ordered execution by acting like LFENCE, controlling the sequencing of subsequent MSR reads after prior MSR reads and instructions.		CPUID.07H.01H:EAX[27] = 1
31:0	DATA Reserved. Always 0.	
63:32	Reserved.	
Register Address: 3AH, 58		IA32_FEATURE_CONTROL
Control Features in Intel 64 Processor (R/W)		If any one enumeration condition for defined bit field holds.
0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written; writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL contents are preserved across RESET when PWRGOOD is not deasserted.	If any one enumeration condition for defined bit field position greater than bit 0 holds.
1	Enable VMX inside SMX operation (R/WL) This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when CPUID.01H:ECX[6:5] returns 11b (SMX and VMX respectively).	If CPUID.01H:ECX[5] = 1 && CPUID.01H:ECX[6] = 1
2	Enable VMX outside SMX operation (R/WL) This bit enables VMX for a system executive that does not require SMX. BIOS must set this bit only when CPUID.01H:ECX.VMX[5] is set.	If CPUID.01H:ECX[5] = 1
7:3	Reserved.	
14:8	SENTER Local Function Enables (R/WL) When set, each bit in the field represents an enable control for a corresponding SENTER function. This field is supported only if CPUID.01H:ECX[6] is set.	If CPUID.01H:ECX[6] = 1
15	SENTER Global Enable (R/WL) This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.01H:ECX[6] is set.	If CPUID.01H:ECX[6] = 1
16	Reserved.	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime re-configuration of SGX Launch Control via the IA32_SGXLEPUBKEYHASHn MSR.	If CPUID.07H.00H:ECX[30] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
18	SGX Global Enable (R/WL) This bit must be set to enable SGX leaf functions.	If CPUID.07H.00H:EBX[2] = 1
19	Reserved.	
20	LMCE On (R/WL) When set, system software can program the MSRs associated with LMCE to configure delivery of some machine check exceptions to a single logical processor.	If IA32_MCG_CAP[27] = 1
63:21	Reserved.	
Register Address: 3BH, 59		IA32_TSC_ADJUST
Per Logical Processor TSC Adjust (R/Write to clear)		If CPUID.07H.00H:EBX[1] = 1
63:0	THREAD_ADJUST Local offset value of the IA32_TSC for a logical processor. Reset value is zero. A write to IA32_TSC will modify the local offset in IA32_TSC_ADJUST and the content of IA32_TSC, but does not affect the internal invariant TSC hardware.	
Register Address: 48H, 72		IA32_SPEC_CTRL
Speculation Control (R/W) The MSR bits are defined as logical processor scope. On some core implementations, the bits may impact sibling logical processors on the same core. This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.		If any one of the enumeration conditions for defined bit field positions holds.
0	Indirect Branch Restricted Speculation (IBRS). Restricts speculation of indirect branch.	If CPUID.07H.00H:EDX[26] = 1
1	Single Thread Indirect Branch Predictors (STIBP). Prevents indirect branch predictions on all logical processors on the core from being controlled by any sibling logical processor in the same core.	If CPUID.07H.00H:EDX[27] = 1
2	Speculative Store Bypass Disable (SSBD) delays speculative execution of a load until the addresses for all older stores are known.	If CPUID.07H.00H:EDX[31] = 1
3	IPRED_DIS_U If 1, enables IPRED_DIS control for CPL3.	If CPUID.07H.02H:EDX[1] = 1
4	IPRED_DIS_S If 1, enables IPRED_DIS control for CPL0/1/2.	If CPUID.07H.02H:EDX[1] = 1
5	RRSBA_DIS_U If 1, disables RRSBA behavior for CPL3.	If CPUID.07H.02H:EDX[2] = 1
6	RRSBA_DIS_S If 1, disables RRSBA behavior for CPL0/1/2.	If CPUID.07H.02H:EDX[2] = 1
7	PSFD If 1, disables Fast Store Forwarding Predictor. Note that setting bit 2 (SSBD) also disables this.	If CPUID.07H.02H:EDX[0] = 1
8	DDPD_U If 1, disables the Data Dependent Prefetcher that examines data values in memory while CPL = 3. Note that setting bit 2 (SSBD) also disables this.	If CPUID.07H.02H:EDX[3] = 1
9	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
10	BHI_DIS_S When '1, enables BHI_DIS_S behavior.	If CPUID.07H.02H:EDX[4] = 1
63:11	Reserved.	
Register Address: 49H, 73		IA32_PRED_CMD
Prediction Command (W0) Gives software a way to issue commands that affect the state of predictors.		If any one of the enumeration conditions for defined bit field positions holds.
0	Indirect Branch Prediction Barrier (IBPB)	If CPUID.07H.00H:EDX[26] = 1
63:1	Reserved.	
Register Address: 4EH, 78		IA32_PPIN_CTL
Protected Processor Inventory Number Enable Control (R/W)		If CPUID.07H.01H:EBX[0] = 1 ¹
0	LockOut (R/W0) If 0, indicates that further writes to IA32_PPIN_CTL is allowed. If 1, indicates that further writes to IA32_PPIN_CTL is disallowed. Writing 1 to this bit is only permitted if the Enable_PPIN bit is clear. The Privileged System Software Inventory Agent should read IA32_PPIN_CTL[bit 1] to determine if IA32_PPIN is accessible. The Privileged System Software Inventory Agent is not expected to write to this MSR.	
1	Enable_PPIN (R/W) If 1, indicates that IA32_PPIN is accessible using RDMSR. If 0, indicates that IA32_PPIN is inaccessible using RDMSR. Any attempt to read IA32_PPIN will cause #GP.	
63:2	Reserved.	
Register Address: 4FH, 79		IA32_PPIN
Protected Processor Inventory Number (R/O)		If CPUID.07H.01H:EBX[0] = 1 ¹
63:0	Protected Processor Inventory Number (R/O) A unique value within a given CPUID family/model/stepping signature that a privileged inventory initialization agent can access to identify each physical processor, when access to IA32_PPIN is enabled. Access to IA32_PPIN is permitted only if IA32_PPIN_CTL[bits 1:0] = '10b'.	
Register Address: 79H, 121		IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)
BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 11.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.		06_01H
Register Address: 7AH, 122		IA32_FEATURE_ACTIVATION
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread.		
0	SE Secure Enclaves feature activation.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	KL Keylocker feature activation.	
63:2	Reserved.	
Register Address: 7BH, 123		IA32_MCU_ENUMERATION
IA32_MCU_ENUMERATION (R/O) Enumeration of architectural features.		
0	UNIFORM_MCU_AVAIL When set to 1, uniform microcode update is available, and UNIFORM_MCU_SCOPE (bits [10:8]) indicates the scope of writes to IA32_BIOS_UPDT_TRIG. When set to 0, uniform microcode update is not available, and writes to IA32_BIOS_UPDT_TRIG are core scoped.	
1	UNIFORM_MCU_CONFIG_REQD When set to 1, indicates that configuration is required to ensure that all MCU components are updated on WRMSR 79H, and UNIFORM_MCU_CONFIG_COMPLETE (bit 2) should be checked to determine whether the necessary configuration has been completed. When set to 0, indicates that no configuration is required, and UNIFORM_MCU_CONFIG_COMPLETE should be ignored.	
2	UNIFORM_MCU_CONFIG_COMPLETE If UNIFORM_MCU_CONFIG_REQD (bit 1) is 0, then this bit should be ignored. If UNIFORM_MCU_CONFIG_REQD is 1, then this bit indicates whether all necessary configurations have been completed to ensure that all MCU components will be updated on WRMSR 79H.	
3	ARCH_ROLLBACK_SVN_COMMIT When set to 1, indicates support for the MCU deferred SVN architecture, SVN reporting architecture, and MCU rollback architecture.	
4	MCU_STAGING When set to 1, indicates that the microcode update staging capability is supported by the processor. When supported, the use of the MCU staging capability is recommended to reduce the latency of the IA32_BIOS_UPDT_TRIG operation.	
7:5	Reserved for future use.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:8	UNIFORM_MCU_SCOPE Indicates the current* uniform microcode update scope: ▪ 0x02: Core Scoped ▪ 0x03: Module Scoped** ▪ 0x04: Tile Scoped** ▪ 0x05: Die Scoped** ▪ 0x80: Package Scoped ▪ 0xC0: Platform Scoped All others: Reserved for future use * The value of this field reflects the state of platform configuration and may change as the configuration changes during the boot process. Once configuration is complete, it is not expected to change during runtime. ** If these domains are enumerated by CPUID.1FH, then this field may also report them as appropriate.	
63:16	Reserved for future use.	
Register Address: 7CH, 124		IA32_MCU_STATUS
MCU Status (R/O) Communicates results from the previous patch loads.		
0	MCU_PARTIAL_UPDATE When set to 1, indicates that the most recent write to IA32_BIOS_UPDT_TRIG resulted in a partial update. This means that microcode update components were only partially updated after some portion of the MCU had already been committed and the Revision ID had been updated.	
1	AUTH_FAIL_ON_MCU_COMPONENT When set to 1, indicates that an authentication failure occurred on some portion of the MCU after another portion of the MCU had already been committed and the Revision ID had already been updated on the most recent write to IA32_BIOS_UPDT_TRIG.	
2	Reserved for future use.	
3	POST_BIOS_MCU When set to 1, indicates that an update was successfully loaded via IA32_BIOS_UPDT_TRIG after bit 0 of MSR_BIOS_DONE (address 151H) was set to 1.	
63:4	Reserved for future use.	
Register Address: 82H, 130		IA32_FZM_RANGE_INDEX
IA32_FZM_RANGE_INDEX (R/W) Index and Domain handle for a valid FZM region. Programmed by software and used by other FRM MSRs FZM Range Index register to R/W Domain Index.		
3:0	REGION_INDEX Holds the Index of domain.	
7:4	Reserved.	
12:8	DOMAIN_HANDLE Holds the Domain Handle.	
63:13	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 83H, 131		IA32_FZM_DOMAIN_CONFIG
IA32_FZM_DOMAIN_CONFIG (R/O) Bit mask of valid regions within the domain identified by FZM_RANGE_INDEX.		
63:0	REGION_BITMAP Bitmap of valid regions for a given domain.	
Register Address: 84H, 132		IA32_FZM_RANGE_STARTADDR
IA32_FZM_RANGE_STARTADDR (R/O) Start address of the FZM range pointed to by FZM_RANGE_INDEX.		
51:0	START_ADDR Start address of the specified domain in FZM_RANGE_INDEX.	
63:52	Reserved.	
Register Address: 85H, 133		IA32_FZM_RANGE_ENDADDR
IA32_FZM_RANGE_ENDADDR (R/O) End address of the specified domain in FZM_RANGE_INDEX.		
51:0	END_ADDR End address of the specified domain in FZM_RANGE_INDEX.	
63:52	Reserved.	
Register Address: 86H, 134		IA32_FZM_RANGE_WRITESTATUS
IA32_FZM_RANGE_WRITESTATUS (R/O) Write status of the FZM range pointed to by FZM_RANGE_INDEX.		
0	WRITE_STATUS Write status of the specified domain in FZM_RANGE_INDEX.	
1	READ_STATUS Read status of the specified domain in FZM_RANGE_INDEX.	
63:2	Reserved.	
Register Address: 87H, 135		IA32_MKTME_KEYID_PARTITIONING
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX.		
31:0	NUM_MKTME_KEYIDS Number of activated Intel TME-MK KeyIDs. This field is supported on all parts that enumerate support for Intel Total Memory Encryption - Multi-Key (Intel TME-MK). If IA32_TME_ACTIVATE.LOCK is 1, this field reports the number of shared KeyIDs supported by the processor (all KeyIDs other than SEAM-private KeyIDs and KeyID 0); otherwise, it reports 0. Intel TME-MK KeyIDs will always span the KeyID range [1 ... NUM_MKTME_KEYIDS]. <i>Note: KeyID 0 is reserved for TME and will not be included.</i>	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:32	NUM_TDX_PRIV_KEYIDS Number of activated SEAM-private KeyIDs. This field is supported on all parts that enumerate support for SEAM mode. If IA32_TME_ACTIVATE.LOCK is 1, this field reports the number of SEAM-private KeyIDs supported by the processor (all KeyIDs, except shared KeyIDs and KeyID 0); otherwise, it reports 0. SEAM-private KeyIDs will always span the range [NUM_MKTME_KEYIDS+1... (NUM_MKTME_KEYIDS + NUM_TDX_PRIV_KEYIDS)]. Note: KeyID 0 is reserved for TME and will not be included.	
Register Address: 8BH, 139		IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)
BIOS Update Signature (R/W) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.		06_01H
31:0	Reserved.	
63:32	PATCH_SIGN_ID It is recommended that this field be preloaded with zero prior to executing CPUID. If the field remains zero following the execution of CPUID, this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature patch signature ID.	
Register Address: 8CH, 140		IA32_SGXLEPUBKEYHASH0
IA32_SGXLEPUBKEYHASH[63:0] (R/W) Bits 63:0 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Read permitted if CPUID.12H.00H:EAX[0] = 1 && CPUID.07H.00H:ECX[30] = 1. Write permitted if CPUID.12H.00H:EAX[0] = 1 && IA32_FEATURE_CONTROL[17] = 1 && IA32_FEATURE_CONTROL[0] = 1.
Register Address: 8DH, 141		IA32_SGXLEPUBKEYHASH1
IA32_SGXLEPUBKEYHASH[127:64] (R/W) Bits 127:64 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 8EH, 142		IA32_SGXLEPUBKEYHASH2
IA32_SGXLEPUBKEYHASH[191:128] (R/W) Bits 191:128 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 8FH, 143		IA32_SGXLEPUBKEYHASH3
IA32_SGXLEPUBKEYHASH[255:192] (R/W) Bits 255:192 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 90H, 144		IA32_SGXLEPUBKEYHASH4
IA32_SGXLEPUBKEYHASH[319:256] (R/W) Bits 319:256 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 91H, 145		IA32_SGXLEPUBKEYHASH5
IA32_SGXLEPUBKEYHASH[383:320] (R/W) Bits 383:320 of the SHA256 digest of the SIGSTRUCT.MODULUS for SGX Launch Enclave. On reset, the default value is the digest of Intel's signing key.		Same comment in MSR listing for IA32_SGXLEPUBKEYHASH0 (MSR address 8CH, 140) applies here.
Register Address: 9BH, 155		IA32_SMM_MONITOR_CTL
SMM Monitor Configuration (R/W)		If CPUID.01H:ECX[5] = 1 CPUID.01H:ECX[6] = 1
0	Valid (R/W)	
1	Reserved.	
2	Controls SMI unblocking by VMXOFF (see Section 33.14.4).	If IA32_VMX_MISC[28]
11:3	Reserved.	
31:12	MSEG Base (R/W)	
63:32	Reserved.	
Register Address: 9EH, 158		IA32_SMBASE
Base address of the logical processor's SMRAM image (R/O, SMM only).		If IA32_VMX_MISC[15]
Register Address: BCH, 188		IA32_MISC_PACKAGE_CTL5
Power Filtering Control (R/W) This MSR has a value of 0 after reset and is unaffected by INIT# or SIPI#.		If IA32_ARCH_CAPABILITIES [10] = 1
0	ENERGY_FILTERING_ENABLE (R/W) If set, RAPL MSRs report filtered processor power consumption data. This bit can be changed from 0 to 1, but cannot be changed from 1 to 0. After setting, all attempts to clear it are ignored until the next processor reset.	If IA32_ARCH_CAPABILITIES [11] = 1
63:1	Reserved.	
Register Address: BDH, 189		IA32_XAPIC_DISABLE_STATUS
xAPIC Disable Status (R/O)		If CPUID.07H:00H:EDX[29] = 1 and IA32_ARCH_CAPABILITIES [21] = 1
0	LEGACY_XAPIC_DISABLED When set, indicates that the local APIC is in x2APIC mode (IA32_APIC_BASE.EXTD = 1) and that attempts to clear IA32_APIC_BASE.EXTD will fail (e.g., WRMSR will #GP).	
63:1	Reserved.	
Register Address: C1H, 193		IA32_PMC0 (PERFCTR0)
General Performance Counter 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0
Register Address: C2H, 194		IA32_PMC1 (PERFCTR1)
General Performance Counter 1 (R/W)		If CPUID.0AH:EAX[15:8] > 1
Register Address: C3H, 195		IA32_PMC2
General Performance Counter 2 (R/W)		If CPUID.0AH:EAX[15:8] > 2
Register Address: C4H, 196		IA32_PMC3
General Performance Counter 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: C5H, 197		IA32_PMC4
General Performance Counter 4 (R/W)		If CPUID.0AH:EAX[15:8] > 4
Register Address: C6H, 198		IA32_PMC5
General Performance Counter 5 (R/W)		If CPUID.0AH:EAX[15:8] > 5
Register Address: C7H, 199		IA32_PMC6
General Performance Counter 6 (R/W)		If CPUID.0AH:EAX[15:8] > 6
Register Address: C8H, 200		IA32_PMC7
General Performance Counter 7 (R/W)		If CPUID.0AH:EAX[15:8] > 7
Register Address: C9H, 201		IA32_PMC8
General Performance Counter 8 (R/W)		If CPUID.0AH:EAX[15:8] > 8
Register Address: CAH, 202		IA32_PMC9
General Performance Counter 9 (R/W)		If CPUID.0AH:EAX[15:8] > 9
Register Address: CFH, 207		IA32_CORE_CAPABILITIES
IA32 Core Capabilities Register		If CPUID.07H.00H:EDX[30] = 1
63:0	Reserved.	No architecturally defined bits.
Register Address: E1H, 225		IA32_UMWAIT_CONTROL
UMWAIT Control (R/W)		
0	C0.2 is not allowed by the OS. Value of "1" means all C0.2 requests revert to C0.1.	
1	Reserved.	
31:2	Determines the maximum time in TSC-quanta that the processor can reside in either C0.1 or C0.2. A zero value indicates no maximum time. The maximum time value is a 32-bit value where the upper 30 bits come from this field and the lower two bits are zero.	
Register Address: E7H, 231		IA32_MPERF
TSC Frequency Clock Counter (R/Write to clear)		If CPUID.06H:ECX[0] = 1
63:0	CO_MCNT: C0 TSC Frequency Clock Count Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
Register Address: E8H, 232		IA32_APERF
Actual Performance Clock Counter (R/Write to clear)		If CPUID.06H:ECX[0] = 1
63:0	CO_ACNT: C0 Actual Frequency Clock Count Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
Register Address: FEH, 254		IA32_MTRRCAP (MTRRcap)
MTRR Capability (R/O) See Section 13.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		06_01H
7:0	VCNT: The number of variable memory type ranges in the processor.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
8	Fixed range MTRRs are supported when set.	
9	Reserved.	
10	WC Supported when set.	
11	SMRR Supported when set.	
12	PRMRR supported when set.	
14:13	Reserved.	
15	SEAMRR supported when set.	
63:16	Reserved.	
Register Address: 10AH, 266		IA32_ARCH_CAPABILITIES
Enumeration of Architectural Features (R/O)		If CPUID.07H.00H:EDX[29] = 1
0	RDCL_NO: The processor is not susceptible to Rogue Data Cache Load (RDCL).	
1	IBRS_ALL: The processor supports enhanced IBRS.	
2	RSBA: The processor supports RSB Alternate. Alternative branch predictors may be used by RET instructions when the RSB is empty. SW using retpoline may be affected by this behavior.	
3	SKIP_L1DFL_VMENTRY: A value of 1 indicates the hypervisor need not flush the L1D on VM entry.	
4	SSB_NO: Processor is not susceptible to Speculative Store Bypass.	
5	MDS_NO: Processor is not susceptible to Microarchitectural Data Sampling (MDS).	
6	IF_PSCCHANGE_MC_NO: The processor is not susceptible to a machine check error due to modifying the size of a code page without TLB invalidation.	
7	TSX_CTRL: If 1, indicates presence of IA32_TSX_CTRL MSR.	
8	TAA_NO: If 1, processor is not affected by TAA.	
9	MCU_CONTROL: If 1, the processor supports the IA32_MCU_CONTROL MSR.	
10	MISC_PACKAGE_CTL: The processor supports IA32_MISC_PACKAGE_CTL MSR.	
11	ENERGY_FILTERING_CTL: The processor supports setting and reading the IA32_MISC_PACKAGE_CTL[0] (ENERGY_FILTERING_ENABLE) bit.	
12	DOITM: If 1, the processor supports Data Operand Independent Timing Mode.	
13	SBDR_SSDP_NO: The processor is not affected by either the Shared Buffers Data Read (SBDR) vulnerability or the Sideband Stale Data Propagator (SSDP).	
14	FBSDP_NO: The processor is not affected by the Fill Buffer Stale Data Propagator (FBSDP).	
15	PSDP_NO: The processor is not affected by vulnerabilities involving the Primary Stale Data Propagator (PSDP).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
16	MCU_ENUMERATION: If 1, the processor supports the IA32_MCU_ENUMERATION and IA32_MCU_STATUS MSRs.	
17	FB_CLEAR: If 1, the processor supports overwrite of fill buffer values as part of MD_CLEAR operations with the VERW instruction. On these processors, L1D_FLUSH does not overwrite fill buffer values.	
18	FB_CLEAR_CTRL: If 1, the processor supports the IA32_MCU_OPT_CTRL MSR and allows software to set bit 3 of that MSR (FB_CLEAR_DIS).	
19	RRSBA: A value of 1 indicates the processor may have the RRSBA alternate prediction behavior, if not disabled by RRSBA_DIS_U or RRSBA_DIS_S.	
20	BHI_NO: A value of 1 indicates BHI_NO branch prediction behavior, regardless of the value of IA32_SPEC_CTRL[BHI_DIS_S] MSR bit.	
21	XAPIC_DISABLE_STATUS: Enumerates that the IA32_XAPIC_DISABLE_STATUS MSR exists, and that bit 0 specifies whether the legacy xAPIC is disabled and APIC state is locked to x2APIC.	
22	MCU_EXTENDED_SERVICE: If 1, the processor supports MCU Extended servicing - IA32_MCU_EXT_SERVICE MSR.	
23	OVERCLOCKING_STATUS: If set, the IA32_OVERCLOCKING_STATUS MSR exists.	
24	PBSRB_NO: If 1, the processor is not affected by issues related to Post-Barrier Return Stack Buffer Predictions.	
25	GDS_CTRL: If 1, the processor supports the GDS_MITG_DIS and GDS_MITG_LOCK bits of the IA32_MCU_OPT_CTRL MSR.	
26	GDS_NO: If 1, the processor is not affected by Gather Data Sampling.	
27	RFDS_NO: If 1, the processor is not affected by Register File Data Sampling.	
28	RFDS_CLEAR: If 1, when VERW is executed the processor will clear stale data from register files affected by Register File Data Sampling.	
29	IGN_UMONITOR_SUPPORT If 0, IA32_MCU_OPT_CTRL bit 6 (IGN_UMONITOR) is not supported. If 1, it indicates support of IA32_MCU_OPT_CTRL bit 6 (IGN_UMONITOR).	
30	MON_UMON_MITG_SUPPORT If 0, IA32_MCU_OPT_CTRL bit 7 (MON_UMON_MITG) is not supported. If 1, it indicates support of IA32_MCU_OPT_CTRL bit 7 (MON_UMON_MITG).	
31	Reserved.	
32	PBOPT_SUPPORT If 0, IA32_PBOPT_CTRL bit 0 (Prediction Barrier Option [PBOPT]) is not supported. If 1, IA32_PBOPT_CTRL bit 0 (Prediction Barrier Option [PBOPT]) is supported.	
61:33	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
62	ITS_NO If 0, the hypervisor indicates that the system is not affected by Indirect Target Selection. If 1, the hypervisor indicates that the system may be affected by Indirect Target Selection.		
63	Reserved.		
Register Address: 10BH, 267		IA32_FLUSH_CMD	
Flush Command (W0) Gives software a way to invalidate structures with finer granularity than other architectural methods.			If any one of the enumeration conditions for defined bit field positions holds.
0	L1D_FLUSH Writeback and invalidate the L1 data cache.		If CPUID.07H.00H:EDX[28] = 1
63:1	Reserved.		
Register Address: 10FH, 271		IA32_TSX_FORCE_ABORT	
TSX Force Abort			If CPUID.07H.00H:EDX[13] = 1
0	RTM_FORCE_ABORT If 1, all RTM transactions abort with EAX code 0.		R/W, Default: 0 If CPUID.07H.00H:EDX[11] = 1, bit 0 is always 1 and writes to change it are ignored. If SDV_ENABLE_RTM is 1, bit 0 is always 0 and writes to change it are ignored.
1	TSX_CPUID_CLEAR When set, CPUID.07H.00H:EBX[11] = 0 and CPUID.07H.00H:EBX[4] = 0.		R/W, Default: 0 Can be set only if CPUID.07H.00H:EDX[11] = 1 or if SDV_ENABLE_RTM is 1.
2	SDV_ENABLE_RTM When set, CPUID.07H.00H:EDX[11] = 0 and the processor may not force abort RTM. This unsupported mode should only be used for software development and not for production usage.		R/W, Default: 0 If 0, can be set only if CPUID.07H.00H:EDX[11] = 1.
63:3	Reserved.		
Register Address: 122H, 290		IA32_TSX_CTRL	
IA32_TSX_CTRL (R/W)			Thread scope. Not architecturally serializing. Available when CPUID.07H.00H:EDX.ARCH_CAPABILITIES[29] = 1 and IA32_ARCH_CAPABILITIES.bit 7 = 1.
0	RTM_DISABLE When set to 1, XBEGIN will always abort with EAX code 0.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	TSX_CPUID_CLEAR When set to 1, CPUID.07H.00H:EBX.RTM[11] and CPUID.07H.00H:EBX.HLE[4] report 0. When set to 0 and the SKU supports TSX, these bits will return 1.	
63:2	Reserved.	
Register Address: 123H, 291		IA32_MCU_OPT_CTRL
Microcode Update Option Control (R/W)		If CPUID.07H.00H:EDX[9] = 1 or CPUID.07H.00H:EDX[11] = 1 IA32_ARCH_CAPABILITIES [18] = 1 or IA32_ARCH_CAPABILITIES [25] = 1 or IA32_ARCH_CAPABILITIES [29] = 1 or IA32_ARCH_CAPABILITIES [30] = 1
0	RNGDS_MITG_DIS (R/W) If 0 (default), SRBDS mitigation is enabled for RDRAND and RDSEED. If 1, SRBDS mitigation is disabled for RDRAND and RDSEED executed outside of Intel SGX enclaves.	If CPUID.07H.00H:EDX[9] = 1
1	RTM_ALLOW If 0, XBEGIN will always abort with EAX code 0. If 1, XBEGIN behavior depends on the value of IA32_TSX_CTRL[RTM_DISABLE].	If CPUID.07H.00H:EDX[11] = 1 Read/Write Setting RTM_LOCKED prevents writes to this bit.
2	RTM_LOCKED When 1, RTM_ALLOW is locked at zero, writes to RTM_ALLOW will be ignored.	If CPUID.07H.00H:EDX[11] = 1 Read-Only status bit.
3	FB_CLEAR_DIS If 1, prevents the VERW instruction from performing an FB_CLEAR action.	If IA32_ARCH_CAPABILITIES [18] = 1
4	GDS_MITG_DIS If 0, the Gather Data Sampling mitigation is enabled (patch load time default). If 1 on all threads for a given core, the Gather Data Sampling mitigation is disabled.	If IA32_ARCH_CAPABILITIES [25] = 1
5	GDS_MITG_LOCK If 0, not locked, and GDS_MITG_DIS is under OS control. If 1, locked and GDS_MITG_DIS is forced to 0 (writes are ignored).	If IA32_ARCH_CAPABILITIES [25] = 1
6	IGN_UMONITOR If 0, enable CPL0-3 software to use the UMONITOR/UMWAIT instructions. If 1 (default), disable UMONITOR functionality. CPL0-3 software will be able to call the UMONITOR instruction without causing a fault, however the address monitoring hardware will not be armed. When UMWAIT is called, it will not enter an implementation-dependent optimized state.	If IA32_ARCH_CAPABILITIES [29] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
7	MON_UMON_MITG If 0 (default), disabled. If 1, enable: Flush the thread's previously monitored address from the CPU caches as part of the (U)MONITOR instruction. Additionally, for every 4th (U)MONITOR instruction within a core, flush the peer hyperthread's monitored address from the CPU caches as well. This will increase the latency of the instruction. This may have a minor impact on workloads using the (U)MONITOR instruction.	If IA32_ARCH_CAPABILITIES [30] = 1
63:8	Reserved.	
Register Address: 174H, 372		IA32_SYSENTER_CS
SYSENTER_CS_MSR (R/W)		06_01H
15:0	CS Selector.	
31:16	Not used.	Can be read and written.
63:32	Not used.	Writes ignored; reads return zero.
Register Address: 175H, 373		IA32_SYSENTER_ESP
SYSENTER_ESP_MSR (R/W)		06_01H
Register Address: 176H, 374		IA32_SYSENTER_EIP
SYSENTER_EIP_MSR (R/W)		06_01H
Register Address: 179H, 377		IA32_MCG_CAP (MCG_CAP)
Global Machine Check Capability (R/O)		06_01H
7:0	Count: Number of reporting banks.	
8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set.	
9	MCG_EXT_P: Extended machine check state registers are present if this bit is set.	
10	MCP_CMCI_P: Support for corrected MC error event is present.	06_01H
11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
12	MCG_SEAM_NR_P: Indicates that the processor supports SEAM non-root operation indication in IA32_MCG_STATUS.	
15:13	Reserved.	
23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	
24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
25	Reserved.	
26	MCG_ELOG_P: Indicates that the processor allows platform firmware to be invoked when an error is detected so that it may provide additional platform specific information in an ACPI format "Generic Error Data Entry" that augments the data included in machine check bank registers.	06_3EH
27	MCG_LMCE_P: Indicates that the processor supports extended state in IA32_MCG_STATUS and associated MSR necessary to configure Local Machine Check Exception (LMCE).	06_3EH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:28	Reserved.	
Register Address: 17AH, 378		IA32_MCG_STATUS (MCG_STATUS)
Global Machine Check Status (R/W)		06_01H
0	RIPV. Restart IP valid.	06_01H
1	EIPV. Error IP valid.	06_01H
2	MCIP. Machine check in progress.	06_01H
3	LMCE_S. Local machine check.	If IA32_MCG_CAP.LMCE_P[27] = 1
11:4	Reserved.	
12	SEAM_NR. SEAM non-root operation.	If IA32_MCG_CAP.SEAM_NR_P[12] = 1
63:13	Reserved.	
Register Address: 17BH, 379		IA32_MCG_CTL (MCG_CTL)
Global Machine Check Control (R/W)		If IA32_MCG_CAP.CTL_P[8] = 1
Register Address: 180H–185H, 384–389		N/A
Reserved		06_0EH ²
Register Address: 186H, 390		IA32_PERFVTSEL0 (PERFVTSEL0)
Performance Event Select Register 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0
7:0	Event Select: Selects a performance event logic unit.	
15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
16	USR: Counts while in privilege level is not ring 0.	
17	OS: Counts while in privilege level is ring 0.	
18	Edge: Enables edge detection if set.	
19	PC: Enables pin control.	
20	INT: Enables interrupt on counter overflow.	
21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
22	EN: Enables the corresponding performance counter to commence counting when this bit is set.	
23	INV: Invert the CMASK.	
31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
63:32	Reserved.	
Register Address: 187H, 391		IA32_PERFVTSEL1 (PERFVTSEL1)
Performance Event Select Register 1 (R/W)		If CPUID.0AH:EAX[15:8] > 1
Register Address: 188H, 392		IA32_PERFVTSEL2
Performance Event Select Register 2 (R/W)		If CPUID.0AH:EAX[15:8] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 189H, 393		IA32_PERFEVTSEL3
Performance Event Select Register 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3
Register Address: 18AH, 394		IA32_PERFEVTSEL4
Performance Event Select Register 4 (R/W)		If CPUID.0AH:EAX[15:8] > 4
Register Address: 18BH, 395		IA32_PERFEVTSEL5
Performance Event Select Register 5 (R/W)		If CPUID.0AH:EAX[15:8] > 5
Register Address: 18CH, 396		IA32_PERFEVTSEL6
Performance Event Select Register 6 (R/W)		If CPUID.0AH:EAX[15:8] > 6
Register Address: 18DH, 397		IA32_PERFEVTSEL7
Performance Event Select Register 7 (R/W)		If CPUID.0AH:EAX[15:8] > 7
Register Address: 18EH, 398		IA32_PERFEVTSEL8
Performance Event Select Register 8 (R/W)		If CPUID.0AH:EAX[15:8] > 8
Register Address: 18FH, 399		IA32_PERFEVTSEL9
Performance Event Select Register 9 (R/W)		If CPUID.0AH:EAX[15:8] > 9
Register Address: 18AH–194H, 394–404		N/A
Reserved.		06_0EH ³
Register Address: 195H, 405		IA32_OVERCLOCKING_STATUS
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR.		
0	Overclocking Utilized Indicates if specific forms of overclocking have been enabled on this boot or reset cycle: 0 indicates no, 1 indicates yes.	
1	Undervolt Protection Indicates if the “Dynamic OC Undervolt Protection” security feature is active: 0 indicates disabled, 1 indicates enabled.	
2	Overclocking Secure Status Indicates that overclocking capabilities have been unlocked by BIOS, with or without overclocking: 0 indicates Not Secured, 1 indicates Secure.	
63:3	Reserved.	
Register Address: 196H–197H, 406–407		N/A
Reserved.		06_0EH ³
Register Address: 198H, 408		IA32_PERF_STATUS
Current Performance Status (R/O) See Section 16.1.1, “Software Interface For Initiating Performance State Transitions.”		0F_03H
15:0	Current Performance State Value.	
63:16	Reserved.	
Register Address: 199H, 409		IA32_PERF_CTL

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Performance Control MSR (R/W) Software makes a request for a new Performance state (P-State) by writing this MSR. See Section 16.1.1, "Software Interface For Initiating Performance State Transitions."		0F_03H
15:0	Target performance State Value.	
31:16	Reserved.	
32	Intel® Dynamic Acceleration Technology Engage (R/W) When set to 1: Disengages Intel Dynamic Acceleration Technology.	06_0FH (Mobile only)
63:33	Reserved.	
Register Address: 19AH, 410		IA32_CLOCK_MODULATION
Clock Modulation Control (R/W) See Section 16.8.3, "Software Controlled Clock Modulation."		If CPUID.01H:EDX[22] = 1
0	Extended On-Demand Clock Modulation Duty Cycle.	If CPUID.06H:EAX[5] = 1
3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	If CPUID.01H:EDX[22] = 1
4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	If CPUID.01H:EDX[22] = 1
63:5	Reserved.	
Register Address: 19BH, 411		IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 16.8.2, "Thermal Monitor."		If CPUID.01H:EDX[22] = 1
0	High-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
1	Low-Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
2	PROCHOT# Interrupt Enable	If CPUID.01H:EDX[22] = 1
3	FORCEPR# Interrupt Enable	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Interrupt Enable	If CPUID.01H:EDX[22] = 1
7:5	Reserved.	
14:8	Threshold #1 Value	If CPUID.01H:EDX[22] = 1
15	Threshold #1 Interrupt Enable	If CPUID.01H:EDX[22] = 1
22:16	Threshold #2 Value	If CPUID.01H:EDX[22] = 1
23	Threshold #2 Interrupt Enable	If CPUID.01H:EDX[22] = 1
24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
63:25	Reserved.	
Register Address: 19CH, 412		IA32_THERM_STATUS
Thermal Status Information (R/O) Contains status information about the processor's thermal sensor and automatic thermal monitoring facilities. See Section 16.8.2, "Thermal Monitor."		If CPUID.01H:EDX[22] = 1
0	Thermal Status (R/O)	If CPUID.01H:EDX[22] = 1
1	Thermal Status Log (R/W)	If CPUID.01H:EDX[22] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
2	PROCHOT # or FORCEPR# event (R/O)	If CPUID.01H:EDX[22] = 1
3	PROCHOT # or FORCEPR# log (R/WCO)	If CPUID.01H:EDX[22] = 1
4	Critical Temperature Status (R/O)	If CPUID.01H:EDX[22] = 1
5	Critical Temperature Status log (R/WCO)	If CPUID.01H:EDX[22] = 1
6	Thermal Threshold #1 Status (R/O)	If CPUID.01H:ECX[8] = 1
7	Thermal Threshold #1 log (R/WCO)	If CPUID.01H:ECX[8] = 1
8	Thermal Threshold #2 Status (R/O)	If CPUID.01H:ECX[8] = 1
9	Thermal Threshold #2 log (R/WCO)	If CPUID.01H:ECX[8] = 1
10	Power Limitation Status (R/O)	If CPUID.06H:EAX[4] = 1
11	Power Limitation log (R/WCO)	If CPUID.06H:EAX[4] = 1
12	Current Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
13	Current Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
14	Cross Domain Limit Status (R/O)	If CPUID.06H:EAX[7] = 1
15	Cross Domain Limit log (R/WCO)	If CPUID.06H:EAX[7] = 1
22:16	Digital Readout (R/O)	If CPUID.06H:EAX[0] = 1
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O)	If CPUID.06H:EAX[0] = 1
31	Reading Valid (R/O)	If CPUID.06H:EAX[0] = 1
63:32	Reserved.	
Register Address: 1A0H, 416		IA32_MISC_ENABLE
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable When set, the fast-strings feature (for REP MOVSB and REP STOSB) is enabled (default). When clear, fast-strings are disabled.	OF_OH
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled. Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2, and adaptive thermal throttling will still be activated. The default value of this field varies with product. See respective tables where default value is listed.	OF_OH
6:4	Reserved.	
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.	OF_OH

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS). 0 = BTS is supported.	0F_0H
12	Processor Event Based Sampling (PEBS) Unavailable (R/O) 1 = PEBS is not supported. 0 = PEBS is supported.	06_0FH
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 0 = Enhanced Intel SpeedStep Technology disabled. 1 = Enhanced Intel SpeedStep Technology enabled.	If CPUID.01H:ECX[7] = 1
17	Reserved.	
18	ENABLE MONITOR FSM (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 2.	CPUID.00H:EAX > 2 and CPUID.07H:01H:EBX. CPUIDMAXVAL_LIM_RMV[3] = 0
23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	If CPUID.01H:ECX[14] = 1
63:24	Reserved. Note: Some older processors defined one of these bits as a disable for the execute-disable feature of paging. If a processor supports this bit, this information is provided in the model-specific tables. See Table 2-3 for the definition of this bit.	
Register Address: 1B0H, 432		IA32_ENERGY_PERF_BIAS
Performance Energy Bias Hint (R/W)		If CPUID.06H:ECX[3] = 1
3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
63:4	Reserved.	
Register Address: 1B1H, 433		IA32_PACKAGE_THERM_STATUS

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Package Thermal Status Information (R/O) Contains status information about the package's thermal sensor. See Section 16.9, "Package Level Thermal Management."		If CPUID.06H:EAX[6] = 1
0	Pkg Thermal Status (R/O)	
1	Pkg Thermal Status Log (R/W)	
2	Pkg PROCHOT # event. (R/O)	
3	Pkg PROCHOT # log. (R/WC0)	
4	Pkg Critical Temperature Status. (R/O)	
5	Pkg Critical Temperature Status Log. (R/WC0)	
6	Pkg Thermal Threshold #1 Status. (R/O)	
7	Pkg Thermal Threshold #1 Log. (R/WC0)	
8	Pkg Thermal Threshold #2 Status. (R/O)	
9	Pkg Thermal Threshold #1 Log. (R/WC0)	
10	Pkg Power Limitation Status. (R/O)	
11	Pkg Power Limitation Log. (R/WC0)	
15:12	Reserved.	
22:16	Pkg Digital Readout. (R/O)	
25:23	Reserved.	
26	Hardware Feedback Interface Structure Change Status.	If CPUID.06H:EAX[19] = 1
63:27	Reserved.	
Register Address: 1B2H, 434		IA32_PACKAGE_THERM_INTERRUPT
Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 16.9, "Package Level Thermal Management."		If CPUID.06H:EAX[6] = 1
0	Pkg High-Temperature Interrupt Enable.	
1	Pkg Low-Temperature Interrupt Enable.	
2	Pkg PROCHOT# Interrupt Enable.	
3	Reserved.	
4	Pkg Overheat Interrupt Enable.	
7:5	Reserved.	
14:8	Pkg Threshold #1 Value.	
15	Pkg Threshold #1 Interrupt Enable.	
22:16	Pkg Threshold #2 Value.	
23	Pkg Threshold #2 Interrupt Enable.	
24	Pkg Power Limit Notification Enable.	
25	Hardware Feedback Interrupt Enable.	If CPUID.06H:EAX[19] = 1
63:26	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 1C4H, 452		IA32_XFD
Extended Feature Disable Control (R/W) Controls which XSAVE-enabled features are temporarily disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.		If CPUID.0DH.01H:EAX[4] = 1
Register Address: 1C5H, 453		IA32_XFD_ERR
Extended Feature Disable Error Code (R/W) Reports which XSAVE-enabled features caused a fault due to being disabled. See Section 13.14 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.		If CPUID.0DH.01H:EAX[4] = 1
Register Address: 1D9H, 473		IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)
Trace/Profile Resource Control (R/W)		06_0EH
0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
2	BLD: Enable OS bus-lock detection. See Section 19.3.1.6 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.	If (CPUID.07H.00H:ECX[24] = 1)
5:3	Reserved.	
6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH
11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H:ECX[15] = 1 && CPUID.0AH:EAX[7:0] > 1
12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 38FH) on a PMI request.	If CPUID.01H:ECX[15] = 1 && CPUID.0AH:EAX[7:0] > 1
13	ENABLE_UNCORE_PMI: When set, enables the logical processor to receive and generate PMI on behalf of the uncore.	06_1AH
14	FREEZE_WHILE_SMM: When set, freezes PerfMon and trace messages while in SMM.	If IA32_PERF_CAPABILITIES[12] = 1
15	RTM_DEBUG: When set, enables DR7 debug bit on XBEGIN.	If (CPUID.07H.00H:EBX[11] = 1)
63:16	Reserved.	
Register Address: 1DDH, 477		IA32_LER_FROM_IP
Last Event Record Source IP Register (R/W)		
63:0	FROM_IP The source IP of the recorded branch or event, in canonical form.	Reset Value: 0
Register Address: 1DEH, 478		IA32_LER_TO_IP

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Last Event Record Destination IP Register (R/W)		
63:0	TO_IP The destination IP of the recorded branch or event, in canonical form.	Reset Value: 0
Register Address: 1E0H, 480		IA32_LER_INFO
Last Event Record Info Register (R/W)		
55:0	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
59:56	BR_TYPE The branch type recorded by this LBR. Encodings match those of IA32_LBR_x_INFO.	Reset Value: 0
60	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel® TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel® TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
63	MISPRED The recorded branch taken/not-taken resolution (for conditional branches) or target (for any indirect branch, including RETs) was mispredicted.	Reset Value: 0
Register Address: 1F2H, 498		IA32_SMRR_PHYSBASE
SMRR Base Address (Writeable only in SMM) Base address of SMM memory range.		If IA32_MTRRCAP.SMRR[11] = 1
7:0	Type. Specifies memory type of the range.	
11:8	Reserved.	
31:12	PhysBase SMRR physical Base Address.	
63:32	Reserved.	
Register Address: 1F3H, 499		IA32_SMRR_PHYSMASK
SMRR Range Mask (Writeable only in SMM) Range Mask of SMM memory range.		If IA32_MTRRCAP[SMRR] = 1
10:0	Reserved.	
11	Valid Enable range mask.	
31:12	PhysMask SMRR address range mask.	
63:32	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 1F8H, 504		IA32_PLATFORM_DCA_CAP
DCA Capability (R)		If CPUID.01H:ECX[18] = 1
Register Address: 1F9H, 505		IA32_CPU_DCA_CAP
If set, CPU supports Prefetch-Hint type.		If CPUID.01H:ECX[18] = 1
Register Address: 1FAH, 506		IA32_DCA_0_CAP
DCA type 0 Status and Control register.		If CPUID.01H:ECX[18] = 1
0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	
2:1	TRANSACTION	
6:3	DCA_TYPE	
10:7	DCA_QUEUE_SIZE	
12:11	Reserved.	
16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	
23:17	Reserved.	
24	SW_BLOCK: SW can request DCA block by setting this bit.	
25	Reserved.	
26	HW_BLOCK: Set when DCA is blocked by HW (e.g., CR0.CD = 1).	
31:27	Reserved.	
Register Address: 200H, 512		IA32_MTRR_PHYSBASE0 (MTRRphysBase0)
See Section 13.11.2.3, "Variable Range MTRRs."		If IA32_MTRRCAP[7:0] > 0
Register Address: 201H, 513		IA32_MTRR_PHYSMASK0
MTRRphysMask0		If IA32_MTRRCAP[7:0] > 0
Register Address: 202H, 514		IA32_MTRR_PHYSBASE1
MTRRphysBase1		If IA32_MTRRCAP[7:0] > 1
Register Address: 203H, 515		IA32_MTRR_PHYSMASK1
MTRRphysMask1		If IA32_MTRRCAP[7:0] > 1
Register Address: 204H, 516		IA32_MTRR_PHYSBASE2
MTRRphysBase2		If IA32_MTRRCAP[7:0] > 2
Register Address: 205H, 517		IA32_MTRR_PHYSMASK2
MTRRphysMask2		If IA32_MTRRCAP[7:0] > 2
Register Address: 206H, 518		IA32_MTRR_PHYSBASE3
MTRRphysBase3		If IA32_MTRRCAP[7:0] > 3
Register Address: 207H, 519		IA32_MTRR_PHYSMASK3
MTRRphysMask3		If IA32_MTRRCAP[7:0] > 3
Register Address: 208H, 520		IA32_MTRR_PHYSBASE4
MTRRphysBase4		If IA32_MTRRCAP[7:0] > 4
Register Address: 209H, 521		IA32_MTRR_PHYSMASK4
MTRRphysMask4		If IA32_MTRRCAP[7:0] > 4

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
MTRRphysBase5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
MTRRphysMask5		If IA32_MTRRCAP[7:0] > 5
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
MTRRphysBase6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
MTRRphysMask6		If IA32_MTRRCAP[7:0] > 6
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
MTRRphysBase7		If IA32_MTRRCAP[7:0] > 7
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
MTRRphysMask7		If IA32_MTRRCAP[7:0] > 7
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
MTRRphysBase8		If IA32_MTRRCAP[7:0] > 8
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
MTRRphysMask8		If IA32_MTRRCAP[7:0] > 8
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
MTRRphysBase9		If IA32_MTRRCAP[7:0] > 9
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
MTRRphysMask9		If IA32_MTRRCAP[7:0] > 9
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
MTRRfix64K_00000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
MTRRfix16K_80000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
MTRRfix16K_A0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	
See Section 13.11.2.2, "Fixed Range MTRRs."		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
MTRRfix4K_C8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
MTRRfix4K_D0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
MTRRfix4K_D8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
MTRRfix4K_E0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MTRRfix4K_E8000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26EH, 622		IA32_MTRR_FIX4K_F0000
MTRRfix4K_F0000		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 26FH, 623		IA32_MTRR_FIX4K_F8000
MTRRfix4K_F8000.		If CPUID.01H:EDX.MTRR[12] = 1
Register Address: 277H, 631		IA32_PAT
IA32_PAT (R/W)		If CPUID.01H:EDX.PAT[16] = 1
2:0	PA0	
7:3	Reserved.	
10:8	PA1	
15:11	Reserved.	
18:16	PA2	
23:19	Reserved.	
26:24	PA3	
31:27	Reserved.	
34:32	PA4	
39:35	Reserved.	
42:40	PA5	
47:43	Reserved.	
50:48	PA6	
55:51	Reserved.	
58:56	PA7	
63:59	Reserved.	
Register Address: 280H, 640		IA32_MCO_CTL2
MSR to enable/disable CMCi capability for bank 0. (R/W) See Section 17.3.2.5, "IA32_MCi_CTL2 MSRs."		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 0
14:0	Corrected error count threshold.	
29:15	Reserved.	
30	CMCI_EN	
63:31	Reserved.	
Register Address: 281H, 641		IA32_MC1_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 1
Register Address: 282H, 642		IA32_MC2_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 2
Register Address: 283H, 643		IA32_MC3_CTL2
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 284H, 644	IA32_MC4_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 4
Register Address: 285H, 645	IA32_MC5_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 5
Register Address: 286H, 646	IA32_MC6_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 6
Register Address: 287H, 647	IA32_MC7_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 7
Register Address: 288H, 648	IA32_MC8_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 8
Register Address: 289H, 649	IA32_MC9_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 9
Register Address: 28AH, 650	IA32_MC10_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 10
Register Address: 28BH, 651	IA32_MC11_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 11
Register Address: 28CH, 652	IA32_MC12_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 12
Register Address: 28DH, 653	IA32_MC13_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 13
Register Address: 28EH, 654	IA32_MC14_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 14
Register Address: 28FH, 655	IA32_MC15_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 15
Register Address: 290H, 656	IA32_MC16_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 16
Register Address: 291H, 657	IA32_MC17_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 17

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 292H, 658	IA32_MC18_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 18
Register Address: 293H, 659	IA32_MC19_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 19
Register Address: 294H, 660	IA32_MC20_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 20
Register Address: 295H, 661	IA32_MC21_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 21
Register Address: 296H, 662	IA32_MC22_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 22
Register Address: 297H, 663	IA32_MC23_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 23
Register Address: 298H, 664	IA32_MC24_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 24
Register Address: 299H, 665	IA32_MC25_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 25
Register Address: 29AH, 666	IA32_MC26_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 26
Register Address: 29BH, 667	IA32_MC27_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 27
Register Address: 29CH, 668	IA32_MC28_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 28
Register Address: 29DH, 669	IA32_MC29_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 29
Register Address: 29EH, 670	IA32_MC30_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 30
Register Address: 29FH, 671	IA32_MC31_CTL2	
Same fields as IA32_MCO_CTL2. (R/W)		If IA32_MCG_CAP[10] = 1 && IA32_MCG_CAP[7:0] > 31

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 2DCH, 732		IA32_INTEGRITY_STATUS
IA32_INTEGRITY_STATUS (R/O) Provides status information for integrity features.		If any one enumeration condition for defined bit field holds.
0	STATIC_LOCKSTEP 0: Static Lockstep Mode is not active on this logical processor. 1: Static Lockstep Mode is active on this logical processor.	If CPUID.07H.01H:EDX[24] = 1
63:1	Reserved.	
Register Address: 2FFH, 767		IA32_MTRR_DEF_TYPE
MTRRdefType (R/W)		If CPUID.01H:EDX.MTRR[12] = 1
2:0	Default Memory Type	
9:3	Reserved.	
10	Fixed Range MTRR Enable	
11	MTRR Enable	
63:12	Reserved.	
Register Address: 309H, 777		IA32_FIXED_CTR0
Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any.		If CPUID.0AH:EDX[4:0] > 0 CPUID.0AH:ECX[0] = 1 CPUID.23H.01H:EBX[0] = 1
Register Address: 30AH, 778		IA32_FIXED_CTR1
Fixed-Function Performance Counter 1 (R/W): Counts CPU_CLK_Unhalted.Core.		If CPUID.0AH:EDX[4:0] > 1 CPUID.0AH:ECX[1] = 1 CPUID.23H.01H:EBX[1] = 1
Register Address: 30BH, 779		IA32_FIXED_CTR2
Fixed-Function Performance Counter 2 (R/W): Counts CPU_CLK_Unhalted.Ref.		If CPUID.0AH:EDX[4:0] > 2 CPUID.0AH:ECX[2] = 1 CPUID.23H.01H:EBX[2] = 1
Register Address: 30CH, 780		IA32_FIXED_CTR3
Fixed-Function Performance Counter 3 (R/W): Top-down Microarchitecture Analysis unhalted number of available slots.		If CPUID.0AH:EDX[4:0] > 3 CPUID.0AH:ECX[3] = 1 CPUID.23H.01H:EBX[3] = 1
Register Address: 30DH, 781		IA32_FIXED_CTR4
Fixed-Function Performance Counter 4 (R/W): Top-down bad speculation.		If CPUID.0AH:EDX[4:0] > 4 CPUID.0AH:ECX[4] = 1 CPUID.23H.01H:EBX[4] = 1
47:0	FIXED_COUNTER Top-down bad speculation counter.	
63:46	Reserved.	
Register Address: 30EH, 782		IA32_FIXED_CTR5
Fixed-Function Performance Counter 5 (R/W): Top-down Frontend Bound.		If CPUID.0AH:EDX[4:0] > 5 CPUID.0AH:ECX[5] = 1 CPUID.23H.01H:EBX[5] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:0	FIXED_COUNTER Top-down Frontend Bound counter.	
63:46	Reserved.	
Register Address: 30FH, 783		IA32_FIXED_CTR6
Fixed-Function Performance Counter 6 (R/W): Top-down retiring.		If CPUID.0AH:EDX[4:0] > 6 CPUID.0AH:ECX[6] = 1 CPUID.23H.01H:EBX[6] = 1
47:0	FIXED_COUNTER Top-down Retiring counter.	
63:46	Reserved.	
Register Address: 345H, 837		IA32_PERF_CAPABILITIES
Read Only MSR that enumerates the existence of performance monitoring features. (R/O)		If CPUID.01H:ECX[15] = 1
5:0	LBR format	
6	PEBS Trap	
7	PEBSSaveArchRegs	
11:8	PEBS Record Format	
12	1: Freeze while SMM is supported.	
13	1: Full width of counter writable via IA32_A_PMCx.	
14	PEBS_BASELINE	
15	1: Performance metrics available.	
16	1: PEBS output will be written into the Intel PT trace stream.	If CPUID.07H.00H:EBX[25] = 1
17	1: Indicates support for PEBS Retire Latency output.	
18	TSX_ADDRESS	
19	RDPMC_METRICS_CLEAR	
63:20	Reserved.	
Register Address: 38DH, 909		IA32_FIXED_CTR_CTRL
Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.		If CPUID.0AH:EAX[7:0] > 1
0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0.	
1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0.	
2	AnyThr0: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
3	EN0_PMI: Enable PMI when fixed counter 0 overflows.	
4	EN1_OS: Enable Fixed Counter 1 to count while CPL = 0.	
5	EN1_Usr: Enable Fixed Counter 1 to count while CPL > 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
6	AnyThr1: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
7	EN1_PMI: Enable PMI when fixed counter 1 overflows.	
8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0.	
9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0.	
10	AnyThr2: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH:EAX[7:0] > 2 && CPUID.0AH:EDX[15] = 0
11	EN2_PMI: Enable PMI when fixed counter 2 overflows.	
12	EN3_OS: Enable Fixed Counter 3 to count while CPL = 0.	
13	EN3_Usr: Enable Fixed Counter 3 to count while CPL > 0.	
14	Reserved.	
15	EN3_PMI: Enable PMI when fixed counter 3 overflows.	
63:16	Reserved.	
Register Address: 38EH, 910		IA32_PERF_GLOBAL_STATUS
Global Performance Counter Status (R/O)		If CPUID.0AH:EAX[7:0] > 0 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Ovf_PMC0: Overflow status of IA32_PMC0.	If CPUID.0AH:EAX[15:8] > 0
1	Ovf_PMC1: Overflow status of IA32_PMC1.	If CPUID.0AH:EAX[15:8] > 1
2	Ovf_PMC2: Overflow status of IA32_PMC2.	If CPUID.0AH:EAX[15:8] > 2
3	Ovf_PMC3: Overflow status of IA32_PMC3.	If CPUID.0AH:EAX[15:8] > 3
n	Ovf_PMCn: Overflow status of IA32_PMCn.	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0.	If CPUID.0AH:EAX[7:0] > 1
33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1.	If CPUID.0AH:EAX[7:0] > 1
34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2.	If CPUID.0AH:EAX[7:0] > 1
32+m	Ovf_FixedCtrm: Overflow status of IA32_FIXED_CTRm.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	OVF_PERF_METRICS: If this bit is set, it indicates that PERF_METRIC counter has overflowed and a PMI is triggered; however, an overflow of fixed counter 3 should normally happen first. If this bit is clear no overflow occurred.	
54:49	Reserved.	
55	Trace_ToPA_PMI: A PMI occurred due to a ToPA entry memory buffer that was completely filled.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
58	LBR_Frz. LBRs are frozen due to: - IA32_DEBUGCTL.FREEZE_LBR_ON_PMI=1. - The LBR stack overflowed.	If CPUID.0AH:EAX[7:0] > 3
59	CTR_Frz. Performance counters in the core PMU are frozen due to: - IA32_DEBUGCTL.FREEZE_PERFMON_ON_PMI=1. - One or more core PMU counters overflowed.	If CPUID.0AH:EAX[7:0] > 3
60	ASCI: Data in the performance counters in the core PMU may include contributions from the direct or indirect operation Intel SGX to protect an enclave.	If the processor supports Intel® SGX.
61	Ovf_Uncore: Uncore counter overflow status.	If CPUID.0AH:EAX[7:0] > 2
62	OvfBuf: DS SAVE area Buffer overflow status.	If CPUID.0AH:EAX[7:0] > 0
63	CondChgd: Status bits of this register have changed.	If CPUID.0AH:EAX[7:0] > 0
Register Address: 38FH, 911		IA32_PERF_GLOBAL_CTRL
Global Performance Counter Control (R/W) Counter increments while the result of ANDing the respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.		If CPUID.0AH:EAX[7:0] > 0
0	EN_PMC0	If CPUID.0AH:EAX[15:8] > 0
1	EN_PMC1	If CPUID.0AH:EAX[15:8] > 1
2	EN_PMC2	If CPUID.0AH:EAX[15:8] > 2
n	EN_PMCn	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	EN_FIXED_CTR0	If CPUID.0AH:EDX[4:0] > 0
33	EN_FIXED_CTR1	If CPUID.0AH:EDX[4:0] > 1
34	EN_FIXED_CTR2	If CPUID.0AH:EDX[4:0] > 2
32+m	EN_FIXED_CTRm	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	EN_PERF_METRICS: If this bit is set and fixed counter 3 is effectively enabled, built-in performance metrics are enabled.	
63:49	Reserved.	
Register Address: 390H, 912		IA32_PERF_GLOBAL_STATUS_RESET
Global Performance Counter Overflow Reset Control (R/W)		If CPUID.0AH:EAX[7:0] > 3 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Set 1 to Clear Ovf_PMC0 bit.	If CPUID.0AH:EAX[15:8] > 0
1	Set 1 to Clear Ovf_PMC1 bit.	If CPUID.0AH:EAX[15:8] > 1
2	Set 1 to Clear Ovf_PMC2 bit.	If CPUID.0AH:EAX[15:8] > 2
n	Set 1 to Clear Ovf_PMCn bit.	If CPUID.0AH:EAX[15:8] > n
31:n	Reserved.	
32	Set 1 to Clear Ovf_FIXED_CTR0 bit.	If CPUID.0AH:EDX[4:0] > 0
33	Set 1 to Clear Ovf_FIXED_CTR1 bit.	If CPUID.0AH:EDX[4:0] > 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
34	Set 1 to Clear Ovf_FIXED_CTR2 bit.	If CPUID.0AH:EDX[4:0] > 2
32+m	Set 1 to Clear Ovf_FIXED_CTRm bit.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	RESET_OVF_PERF_METRICS: If this bit is set, it will clear the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
54:49	Reserved.	
55	Set 1 to Clear Trace_ToPA_PMI bit.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	Set 1 to Clear LBR_Frz bit.	If CPUID.0AH:EAX[7:0] > 3
59	Set 1 to Clear CTR_Frz bit.	If CPUID.0AH:EAX[7:0] > 3
60	Set 1 to Clear ASCI bit.	If the processor supports Intel® SGX.
61	Set 1 to Clear Ovf_Uncore bit.	06_2EH
62	Set 1 to Clear OvfBuf bit.	If CPUID.0AH:EAX[7:0] > 0
63	Set 1 to clear CondChgd bit.	If CPUID.0AH:EAX[7:0] > 0
Register Address: 391H, 913		IA32_PERF_GLOBAL_STATUS_SET
Global Performance Counter Overflow Set Control (R/W)		If CPUID.0AH:EAX[7:0] > 3 (CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1)
0	Set 1 to cause Ovf_PMC0 = 1.	If CPUID.0AH:EAX[7:0] > 3
1	Set 1 to cause Ovf_PMC1 = 1.	If CPUID.0AH:EAX[15:8] > 1
2	Set 1 to cause Ovf_PMC2 = 1.	If CPUID.0AH:EAX[15:8] > 2
n	Set 1 to cause Ovf_PMCn = 1.	If CPUID.0AH:EAX[15:8] > n
31:n	Reserved.	
32	Set 1 to cause Ovf_FIXED_CTR0 = 1.	If CPUID.0AH:EAX[7:0] > 3
33	Set 1 to cause Ovf_FIXED_CTR1 = 1.	If CPUID.0AH:EAX[7:0] > 3
34	Set 1 to cause Ovf_FIXED_CTR2 = 1.	If CPUID.0AH:EAX[7:0] > 3
32+m	Set 1 to cause Ovf_FIXED_CTRm = 1.	If CPUID.0AH:ECX[m] == 1 CPUID.0AH:EDX[4:0] > m
47:33+m	Reserved.	
48	SET_OVF_PERF_METRICS: If this bit is set, it will set the status bit in the IA32_PERF_GLOBAL_STATUS register for the PERF_METRICS counters.	
54:49	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	If CPUID.07H.00H:EBX[25] = 1 && CPUID.14H.00H:ECX[0] = 1
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	If CPUID.0AH:EAX[7:0] > 3
59	Set 1 to cause CTR_Frz = 1.	If CPUID.0AH:EAX[7:0] > 3

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
60	Set 1 to cause ASCI = 1.	If the processor supports Intel® SGX.
61	Set 1 to cause Ovf_Uncore = 1.	If CPUID.0AH:EAX[7:0] > 3
62	Set 1 to cause OvfBuf = 1.	If CPUID.0AH:EAX[7:0] > 3
63	Reserved.	
Register Address: 392H, 914		IA32_PERF_GLOBAL_INUSE
Indicator that core PerfMon interface is in use. (R/O)		If CPUID.0AH:EAX[7:0] > 3
0	IA32_PERFEVTSEL0 in use.	
1	IA32_PERFEVTSEL1 in use.	If CPUID.0AH:EAX[15:8] > 1
2	IA32_PERFEVTSEL2 in use.	If CPUID.0AH:EAX[15:8] > 2
n	IA32_PERFEVTSELn in use.	If CPUID.0AH:EAX[15:8] > n
31:n+1	Reserved.	
32	IA32_FIXED_CTR0 in use.	
33	IA32_FIXED_CTR1 in use.	
34	IA32_FIXED_CTR2 in use.	
32+m	IA32_FIXED_CTRm in use.	
62:33+m	Reserved or model specific.	
63	PMI in use.	
Register Address: 3F1H, 1009		IA32_PEBE_ENABLE
PEBS Control (R/W)		
0	Enable PEBS on IA32_PMC0.	06_0FH
3:1	Reserved or model specific.	
31:4	Reserved.	
35:32	Reserved or model specific.	
63:36	Reserved.	
Register Address: 400H, 1024		IA32_MCO_CTL
MCO_CTL		If IA32_MCG_CAP.CNT > 0
Register Address: 401H, 1025		IA32_MCO_STATUS
MCO_STATUS		If IA32_MCG_CAP.CNT > 0
Register Address: 402H, 1026		IA32_MCO_ADDR ¹
MCO_ADDR		If IA32_MCG_CAP.CNT > 0
Register Address: 403H, 1027		IA32_MCO_MISC
MCO_MISC		If IA32_MCG_CAP.CNT > 0
Register Address: 404H, 1028		IA32_MC1_CTL
MC1_CTL		If IA32_MCG_CAP.CNT > 1
Register Address: 405H, 1029		IA32_MC1_STATUS
MC1_STATUS		If IA32_MCG_CAP.CNT > 1
Register Address: 406H, 1030		IA32_MC1_ADDR ²

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC1_ADDR		If IA32_MCG_CAP.CNT > 1
Register Address: 407H, 1031	IA32_MC1_MISC	
MC1_MISC		If IA32_MCG_CAP.CNT > 1
Register Address: 408H, 1032	IA32_MC2_CTL	
MC2_CTL		If IA32_MCG_CAP.CNT > 2
Register Address: 409H, 1033	IA32_MC2_STATUS	
MC2_STATUS		If IA32_MCG_CAP.CNT > 2
Register Address: 40AH, 1034	IA32_MC2_ADDR ¹	
MC2_ADDR		If IA32_MCG_CAP.CNT > 2
Register Address: 40BH, 1035	IA32_MC2_MISC	
MC2_MISC		If IA32_MCG_CAP.CNT > 2
Register Address: 40CH, 1036	IA32_MC3_CTL	
MC3_CTL		If IA32_MCG_CAP.CNT > 3
Register Address: 40DH, 1037	IA32_MC3_STATUS	
MC3_STATUS		If IA32_MCG_CAP.CNT > 3
Register Address: 40EH, 1038	IA32_MC3_ADDR ¹	
MC3_ADDR		If IA32_MCG_CAP.CNT > 3
Register Address: 40FH, 1039	IA32_MC3_MISC	
MC3_MISC		If IA32_MCG_CAP.CNT > 3
Register Address: 410H, 1040	IA32_MC4_CTL	
MC4_CTL		If IA32_MCG_CAP.CNT > 4
Register Address: 411H, 1041	IA32_MC4_STATUS	
MC4_STATUS		If IA32_MCG_CAP.CNT > 4
Register Address: 412H, 1042	IA32_MC4_ADDR ¹	
MC4_ADDR		If IA32_MCG_CAP.CNT > 4
Register Address: 413H, 1043	IA32_MC4_MISC	
MC4_MISC		If IA32_MCG_CAP.CNT > 4
Register Address: 414H, 1044	IA32_MC5_CTL	
MC5_CTL		If IA32_MCG_CAP.CNT > 5
Register Address: 415H, 1045	IA32_MC5_STATUS	
MC5_STATUS		If IA32_MCG_CAP.CNT > 5
Register Address: 416H, 1046	IA32_MC5_ADDR ¹	
MC5_ADDR		If IA32_MCG_CAP.CNT > 5
Register Address: 417H, 1047	IA32_MC5_MISC	
MC5_MISC		If IA32_MCG_CAP.CNT > 5
Register Address: 418H, 1048	IA32_MC6_CTL	
MC6_CTL		If IA32_MCG_CAP.CNT > 6

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 419H, 1049	IA32_MC6_STATUS	
MC6_STATUS		If IA32_MCG_CAP.CNT > 6
Register Address: 41AH, 1050	IA32_MC6_ADDR ¹	
MC6_ADDR		If IA32_MCG_> 6CAP.CNT > 6
Register Address: 41BH, 1051	IA32_MC6_MISC	
MC6_MISC		If IA32_MCG_CAP.CNT > 6
Register Address: 41CH, 1052	IA32_MC7_CTL	
MC7_CTL		If IA32_MCG_CAP.CNT > 7
Register Address: 41DH, 1053	IA32_MC7_STATUS	
MC7_STATUS		If IA32_MCG_CAP.CNT > 7
Register Address: 41EH, 1054	IA32_MC7_ADDR ¹	
MC7_ADDR		If IA32_MCG_CAP.CNT > 7
Register Address: 41FH, 1055	IA32_MC7_MISC	
MC7_MISC		If IA32_MCG_CAP.CNT > 7
Register Address: 420H, 1056	IA32_MC8_CTL	
MC8_CTL		If IA32_MCG_CAP.CNT > 8
Register Address: 421H, 1057	IA32_MC8_STATUS	
MC8_STATUS		If IA32_MCG_CAP.CNT > 8
Register Address: 422H, 1058	IA32_MC8_ADDR ¹	
MC8_ADDR		If IA32_MCG_CAP.CNT > 8
Register Address: 423H, 1059	IA32_MC8_MISC	
MC8_MISC		If IA32_MCG_CAP.CNT > 8
Register Address: 424H, 1060	IA32_MC9_CTL	
MC9_CTL		If IA32_MCG_CAP.CNT > 9
Register Address: 425H, 1061	IA32_MC9_STATUS	
MC9_STATUS		If IA32_MCG_CAP.CNT > 9
Register Address: 426H, 1062	IA32_MC9_ADDR ¹	
MC9_ADDR		If IA32_MCG_CAP.CNT > 9
Register Address: 427H, 1063	IA32_MC9_MISC	
MC9_MISC		If IA32_MCG_CAP.CNT > 9
Register Address: 428H, 1064	IA32_MC10_CTL	
MC10_CTL		If IA32_MCG_CAP.CNT > 10
Register Address: 429H, 1065	IA32_MC10_STATUS	
MC10_STATUS		If IA32_MCG_CAP.CNT > 10
Register Address: 42AH, 1066	IA32_MC10_ADDR ¹	
MC10_ADDR		If IA32_MCG_CAP.CNT > 10
Register Address: 42BH, 1067	IA32_MC10_MISC	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC10_MISC		If IA32_MCG_CAP.CNT > 10
Register Address: 42CH, 1068	IA32_MC11_CTL	
MC11_CTL		If IA32_MCG_CAP.CNT > 11
Register Address: 42DH, 1069	IA32_MC11_STATUS	
MC11_STATUS		If IA32_MCG_CAP.CNT > 11
Register Address: 42EH, 1070	IA32_MC11_ADDR ¹	
MC11_ADDR		If IA32_MCG_CAP.CNT > 11
Register Address: 42FH, 1071	IA32_MC11_MISC	
MC11_MISC		If IA32_MCG_CAP.CNT > 11
Register Address: 430H, 1072	IA32_MC12_CTL	
MC12_CTL		If IA32_MCG_CAP.CNT > 12
Register Address: 431H, 1073	IA32_MC12_STATUS	
MC12_STATUS		If IA32_MCG_CAP.CNT > 12
Register Address: 432H, 1074	IA32_MC12_ADDR ¹	
MC12_ADDR		If IA32_MCG_CAP.CNT > 12
Register Address: 433H, 1075	IA32_MC12_MISC	
MC12_MISC		If IA32_MCG_CAP.CNT > 12
Register Address: 434H, 1076	IA32_MC13_CTL	
MC13_CTL		If IA32_MCG_CAP.CNT > 13
Register Address: 435H, 1077	IA32_MC13_STATUS	
MC13_STATUS		If IA32_MCG_CAP.CNT > 13
Register Address: 436H, 1078	IA32_MC13_ADDR ¹	
MC13_ADDR		If IA32_MCG_CAP.CNT > 13
Register Address: 437H, 1079	IA32_MC13_MISC	
MC13_MISC		If IA32_MCG_CAP.CNT > 13
Register Address: 438H, 1080	IA32_MC14_CTL	
MC14_CTL		If IA32_MCG_CAP.CNT > 14
Register Address: 439H, 1081	IA32_MC14_STATUS	
MC14_STATUS		If IA32_MCG_CAP.CNT > 14
Register Address: 43AH, 1082	IA32_MC14_ADDR ¹	
MC14_ADDR		If IA32_MCG_CAP.CNT > 14
Register Address: 43BH, 1083	IA32_MC14_MISC	
MC14_MISC		If IA32_MCG_CAP.CNT > 14
Register Address: 43CH, 1084	IA32_MC15_CTL	
MC15_CTL		If IA32_MCG_CAP.CNT > 15
Register Address: 43DH, 1085	IA32_MC15_STATUS	
MC15_STATUS		If IA32_MCG_CAP.CNT > 15

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 43EH, 1086	IA32_MC15_ADDR ¹	
MC15_ADDR		If IA32_MCG_CAP.CNT > 15
Register Address: 43FH, 1087	IA32_MC15_MISC	
MC15_MISC		If IA32_MCG_CAP.CNT > 15
Register Address: 440H, 1088	IA32_MC16_CTL	
MC16_CTL		If IA32_MCG_CAP.CNT > 16
Register Address: 441H, 1089	IA32_MC16_STATUS	
MC16_STATUS		If IA32_MCG_CAP.CNT > 16
Register Address: 442H, 1090	IA32_MC16_ADDR ¹	
MC16_ADDR		If IA32_MCG_CAP.CNT > 16
Register Address: 443H, 1091	IA32_MC16_MISC	
MC16_MISC		If IA32_MCG_CAP.CNT > 16
Register Address: 444H, 1092	IA32_MC17_CTL	
MC17_CTL		If IA32_MCG_CAP.CNT > 17
Register Address: 445H, 1093	IA32_MC17_STATUS	
MC17_STATUS		If IA32_MCG_CAP.CNT > 17
Register Address: 446H, 1094	IA32_MC17_ADDR ¹	
MC17_ADDR		If IA32_MCG_CAP.CNT > 17
Register Address: 447H, 1095	IA32_MC17_MISC	
MC17_MISC		If IA32_MCG_CAP.CNT > 17
Register Address: 448H, 1096	IA32_MC18_CTL	
MC18_CTL		If IA32_MCG_CAP.CNT > 18
Register Address: 449H, 1097	IA32_MC18_STATUS	
MC18_STATUS		If IA32_MCG_CAP.CNT > 18
Register Address: 44AH, 1098	IA32_MC18_ADDR ¹	
MC18_ADDR		If IA32_MCG_CAP.CNT > 18
Register Address: 44BH, 1099	IA32_MC18_MISC	
MC18_MISC		If IA32_MCG_CAP.CNT > 18
Register Address: 44CH, 1100	IA32_MC19_CTL	
MC19_CTL		If IA32_MCG_CAP.CNT > 19
Register Address: 44DH, 1101	IA32_MC19_STATUS	
MC19_STATUS		If IA32_MCG_CAP.CNT > 19
Register Address: 44EH, 1102	IA32_MC19_ADDR ¹	
MC19_ADDR		If IA32_MCG_CAP.CNT > 19
Register Address: 44FH, 1103	IA32_MC19_MISC	
MC19_MISC		If IA32_MCG_CAP.CNT > 19
Register Address: 450H, 1104	IA32_MC20_CTL	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC20_CTL		If IA32_MCG_CAP.CNT > 20
Register Address: 451H, 1105		IA32_MC20_STATUS
MC20_STATUS		If IA32_MCG_CAP.CNT > 20
Register Address: 452H, 1106		IA32_MC20_ADDR ¹
MC20_ADDR		If IA32_MCG_CAP.CNT > 20
Register Address: 453H, 1107		IA32_MC20_MISC
MC20_MISC		If IA32_MCG_CAP.CNT > 20
Register Address: 454H, 1108		IA32_MC21_CTL
MC21_CTL		If IA32_MCG_CAP.CNT > 21
Register Address: 455H, 1109		IA32_MC21_STATUS
MC21_STATUS		If IA32_MCG_CAP.CNT > 21
Register Address: 456H, 1110		IA32_MC21_ADDR ¹
MC21_ADDR		If IA32_MCG_CAP.CNT > 21
Register Address: 457H, 1111		IA32_MC21_MISC
MC21_MISC		If IA32_MCG_CAP.CNT > 21
Register Address: 458H, 1112		IA32_MC22_CTL
MC22_CTL		If IA32_MCG_CAP.CNT > 22
Register Address: 459H, 1113		IA32_MC22_STATUS
MC22_STATUS		If IA32_MCG_CAP.CNT > 22
Register Address: 45AH, 1114		IA32_MC22_ADDR ¹
MC22_ADDR		If IA32_MCG_CAP.CNT > 22
Register Address: 45BH, 1115		IA32_MC22_MISC
MC22_MISC		If IA32_MCG_CAP.CNT > 22
Register Address: 45CH, 1116		IA32_MC23_CTL
MC23_CTL		If IA32_MCG_CAP.CNT > 23
Register Address: 45DH, 1117		IA32_MC23_STATUS
MC23_STATUS		If IA32_MCG_CAP.CNT > 23
Register Address: 45EH, 1118		IA32_MC23_ADDR ¹
MC23_ADDR		If IA32_MCG_CAP.CNT > 23
Register Address: 45FH, 1119		IA32_MC23_MISC
MC23_MISC		If IA32_MCG_CAP.CNT > 23
Register Address: 460H, 1120		IA32_MC24_CTL
MC24_CTL		If IA32_MCG_CAP.CNT > 24
Register Address: 461H, 1121		IA32_MC24_STATUS
MC24_STATUS		If IA32_MCG_CAP.CNT > 24
Register Address: 462H, 1122		IA32_MC24_ADDR ¹
MC24_ADDR		If IA32_MCG_CAP.CNT > 24

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 463H, 1123	IA32_MC24_MISC	
MC24_MISC		If IA32_MCG_CAP.CNT > 24
Register Address: 464H, 1124	IA32_MC25_CTL	
MC25_CTL		If IA32_MCG_CAP.CNT > 25
Register Address: 465H, 1125	IA32_MC25_STATUS	
MC25_STATUS		If IA32_MCG_CAP.CNT > 25
Register Address: 466H, 1126	IA32_MC25_ADDR ¹	
MC25_ADDR		If IA32_MCG_CAP.CNT > 25
Register Address: 467H, 1127	IA32_MC25_MISC	
MC25_MISC		If IA32_MCG_CAP.CNT > 25
Register Address: 468H, 1128	IA32_MC26_CTL	
MC26_CTL		If IA32_MCG_CAP.CNT > 26
Register Address: 469H, 1129	IA32_MC26_STATUS	
MC26_STATUS		If IA32_MCG_CAP.CNT > 26
Register Address: 46AH, 1130	IA32_MC26_ADDR ¹	
MC26_ADDR		If IA32_MCG_CAP.CNT > 26
Register Address: 46BH, 1131	IA32_MC26_MISC	
MC26_MISC		If IA32_MCG_CAP.CNT > 26
Register Address: 46CH, 1132	IA32_MC27_CTL	
MC27_CTL		If IA32_MCG_CAP.CNT > 27
Register Address: 46DH, 1133	IA32_MC27_STATUS	
MC27_STATUS		If IA32_MCG_CAP.CNT > 27
Register Address: 46EH, 1134	IA32_MC27_ADDR ¹	
MC27_ADDR		If IA32_MCG_CAP.CNT > 27
Register Address: 46FH, 1135	IA32_MC27_MISC	
MC27_MISC		If IA32_MCG_CAP.CNT > 27
Register Address: 470H, 1136	IA32_MC28_CTL	
MC28_CTL		If IA32_MCG_CAP.CNT > 28
Register Address: 471H, 1137	IA32_MC28_STATUS	
MC28_STATUS		If IA32_MCG_CAP.CNT > 28
Register Address: 472H, 1138	IA32_MC28_ADDR ¹	
MC28_ADDR		If IA32_MCG_CAP.CNT > 28
Register Address: 473H, 1139	IA32_MC28_MISC	
MC28_MISC		If IA32_MCG_CAP.CNT > 28
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL		If IA32_MCG_CAP.CNT > 29
Register Address: 475H, 1141	IA32_MC29_STATUS	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
MC29_STATUS		If IA32_MCG_CAP.CNT > 29
Register Address: 476H, 1142		IA32_MC29_ADDR
MC29_ADDR		If IA32_MCG_CAP.CNT > 29
Register Address: 477H, 1143		IA32_MC29_MISC
MC29_MISC		If IA32_MCG_CAP.CNT > 29
Register Address: 478H, 1144		IA32_MC30_CTL
MC30_CTL		If IA32_MCG_CAP.CNT > 30
Register Address: 479H, 1145		IA32_MC30_STATUS
MC30_STATUS		If IA32_MCG_CAP.CNT > 30
Register Address: 47AH, 1146		IA32_MC30_ADDR
MC30_ADDR		If IA32_MCG_CAP.CNT > 30
Register Address: 47BH, 1147		IA32_MC30_MISC
MC30_MISC		If IA32_MCG_CAP.CNT > 30
Register Address: 47CH, 1148		IA32_MC31_CTL
MC31_CTL		If IA32_MCG_CAP.CNT > 31
Register Address: 47DH, 1149		IA32_MC31_STATUS
MC31_STATUS		If IA32_MCG_CAP.CNT > 31
Register Address: 47EH, 1150		IA32_MC31_ADDR
MC31_ADDR		If IA32_MCG_CAP.CNT > 31
Register Address: 47FH, 1151		IA32_MC31_MISC
MC31_MISC		If IA32_MCG_CAP.CNT > 31
Register Address: 480H, 1152		IA32_VMX_BASIC
Reporting Register of Basic VMX Capabilities (R/O) See Appendix A.1, "Basic VMX Information."		If CPUID.01H:ECX[5] = 1
Register Address: 481H, 1153		IA32_VMX_PINBASED_CTL
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 482H, 1154		IA32_VMX_PROCBASED_CTL
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 483H, 1155		IA32_VMX_EXIT_CTL
Capability Reporting Register of Primary VM-Exit Controls (R/O) See Appendix A.4.1, "Primary VM-Exit Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 484H, 1156		IA32_VMX_ENTRY_CTL
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls."		If CPUID.01H:ECX[5] = 1
Register Address: 485H, 1157		IA32_VMX_MISC

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data."		If CPUID.01H:ECX[5] = 1
Register Address: 486H, 1158	IA32_VMX_CRO_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX[5] = 1
Register Address: 487H, 1159	IA32_VMX_CRO_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0."		If CPUID.01H:ECX[5] = 1
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."		If CPUID.01H:ECX[5] = 1
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4."		If CPUID.01H:ECX[5] = 1
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration."		If CPUID.01H:ECX[5] = 1
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTLS[63])
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_CAP	
Capability Reporting Register of EPT and VPID (R/O) See Appendix A.10, "VPID and EPT Capabilities."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTLS[63 && (IA32_VMX_PROCBASED_CTLS2[3 3]) IA32_VMX_PROCBASED_CTLS2[3 7]))
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Appendix A.3.1, "Pin-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."		If(CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Appendix A.4, "VM-Exit Controls."		If(CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Appendix A.5, "VM-Entry Controls."		If(CPUID.01H:ECX[5] && IA32_VMX_BASIC[55])

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O)		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL[63] && IA32_VMX_PROCBASED_CTL[45])
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL[3]	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Appendix A.3.4, "Tertiary Processor-Based VM-Execution Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_PROCBASED_CTL[49])
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL[2]	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Appendix A.4.2, "Secondary VM-Exit Controls."		If (CPUID.01H:ECX[5] && IA32_VMX_EXIT_CTL[63])
Register Address: 4C1H, 1217	IA32_A_PMC0	
Full Width Writable IA32_PMC0 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 0) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[0] = 1
Register Address: 4C2H, 1218	IA32_A_PMC1	
Full Width Writable IA32_PMC1 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 1) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[1] = 1
Register Address: 4C3H, 1219	IA32_A_PMC2	
Full Width Writable IA32_PMC2 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 2) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[2] = 1
Register Address: 4C4H, 1220	IA32_A_PMC3	
Full Width Writable IA32_PMC3 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 3) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[3] = 1
Register Address: 4C5H, 1221	IA32_A_PMC4	
Full Width Writable IA32_PMC4 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 4) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[4] = 1
Register Address: 4C6H, 1222	IA32_A_PMC5	
Full Width Writable IA32_PMC5 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 5) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[5] = 1
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 6) && IA32_PERF_CAPABILITIES[13] = 1 CPUID.23H.01H:EAX[6] = 1
Register Address: 4C8H, 1224	IA32_A_PMC7	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Full Width Writable IA32_PMC7 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 7) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[7] = 1
Register Address: 4C9H, 1225		IA32_A_PMC8
Full Width Writable IA32_PMC8 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 8) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[8] = 1
Register Address: 4CAH, 1226		IA32_A_PMC9
Full Width Writable IA32_PMC9 Alias (R/W)		If (CPUID.0AH:EAX[15:8] > 9) && IA32_PERF_CAPABILITIES[13] = 1) CPUID.23H.01H:EAX[9] = 1
Register Address: 4D0H, 1232		IA32_MCG_EXT_CTL
Allows software to signal some MCEs to only a single logical processor in the system. (R/W) See Section 17.3.1.4, "IA32_MCG_EXT_CTL MSR."		If IA32_MCG_CAP.LMCE_P = 1
0	LMCE_EN Enable / Disable local machine check exception.	
63:1	Reserved.	
Register Address: 500H, 1280		IA32_SGX_SVN_STATUS
Status and SVN Threshold of SGX Support for ACM (R/O)		If CPUID.07H.00H:EBX[2] = 1
0	Lock.	See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)."
15:1	Reserved.	
23:16	SGX_SVN_SINIT	See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)."
63:24	Reserved.	
Register Address: 560H, 1376		IA32_RTIT_OUTPUT_BASE
Trace Output Base Register (R/W)		If ((CPUID.07H.00H:EBX[25] = 1) && ((CPUID.14H.00H:ECX[0] = 1) (CPUID.14H.00H:ECX[2] = 1)))
6:0	Reserved.	
M-1:7	Base physical address.	M is the value enumerated by CPUID.80000008H:EAX[7:0].
63:M	Reserved.	
Register Address: 561H, 1377		IA32_RTIT_OUTPUT_MASK_PTRS
Trace Output Mask Pointers Register (R/W)		If ((CPUID.07H.00H:EBX[25] = 1) && ((CPUID.14H.00H:ECX[0] = 1) (CPUID.14H.00H:ECX[2] = 1)))
6:0	Reserved.	
31:7	MaskOrTableOffset.	
63:32	Output Offset.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 570H, 1392		IA32_RTIT_CTL
Trace Control Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
0	TraceEn	
1	CYCEn	If (CPUID.07H.00H:EBX[1] = 1)
2	OS	
3	User	
4	PwrEvtEn	If (CPUID.07H.01H:EBX[5] = 1)
5	FUPonPTW	If (CPUID.07H.01H:EBX[4] = 1)
6	FabricEn	If (CPUID.07H.00H:ECX[3] = 1)
7	CR3Filter	If (CPUID.14H.00H:EBX[0] = 1)
8	ToPA	
9	MTCEn	If (CPUID.07H.00H:EBX[3] = 1)
10	TSCEn	
11	DisRETC	
12	PTWEn	If (CPUID.07H.01H:EBX[4] = 1)
13	BranchEn	
17:14	MTCFreq.	If (CPUID.07H.00H:EBX[3] = 1)
18	Reserved, must be zero.	
22:19	CycThresh	If (CPUID.07H.00H:EBX[1] = 1)
23	Reserved, must be zero.	
27:24	PSBFreq	If (CPUID.07H.00H:EBX[1] = 1)
30:28	Reserved, must be zero.	
31	EventEn	If (CPUID.14H.00H:EBX[7] = 1)
35:32	ADDR0_CFG	If (CPUID.07H.01H:EAX[2:0] > 0)
39:36	ADDR1_CFG	If (CPUID.07H.01H:EAX[2:0] > 1)
43:40	ADDR2_CFG	If (CPUID.07H.01H:EAX[2:0] > 2)
47:44	ADDR3_CFG	If (CPUID.07H.01H:EAX[2:0] > 3)
54:48	Reserved, must be zero.	
55	DisTNT	If (CPUID.14H.00H:EBX[8] = 1)
56	InjectPsbPmiOnEnable	If (CPUID.07H.01H:EBX[6] = 1)
63:57	Reserved, must be zero.	
Register Address: 571H, 1393		IA32_RTIT_STATUS
Tracing Status Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
0	FilterEn (writes ignored).	If (CPUID.07H.00H:EBX[2] = 1)
1	ContexEn (writes ignored).	
2	TriggerEn (writes ignored).	
3	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
4	Error	
5	Stopped	
6	PendPSB	If (CPUID.07H.00H:EBX[6] = 1)
7	PendToPAPMI	If (CPUID.07H.00H:EBX[6] = 1)
31:8	Reserved, must be zero.	
48:32	PacketByteCnt	If (CPUID.07H.00H:EBX[1] > 3)
63:49	Reserved.	
Register Address: 572H, 1394		IA32_RTIT_CR3_MATCH
Trace Filter CR3 Match Register (R/W)		If (CPUID.07H.00H:EBX[25] = 1)
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408		IA32_RTIT_ADDR0_A
Region 0 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 0)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 581H, 1409		IA32_RTIT_ADDR0_B
Region 0 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 0)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 582H, 1410		IA32_RTIT_ADDR1_A
Region 1 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 1)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 583H, 1411		IA32_RTIT_ADDR1_B
Region 1 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 1)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 584H, 1412		IA32_RTIT_ADDR2_A
Region 2 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 585H, 1413		IA32_RTIT_ADDR2_B
Region 2 End Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 2)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 586H, 1414		IA32_RTIT_ADDR3_A
Region 3 Start Address (R/W)		If (CPUID.07H.01H:EAX[2:0] > 3)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 587H, 1415		IA32_RTIT_ADDR3_B
Region 3 End Address (R/W)		If (CPUID.07H.01H:EDX[2:0] > 3)
47:0	Virtual Address.	
63:48	SignExt_VA	
Register Address: 600H, 1536		IA32_DS_AREA
DS Save Area (R/W) Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 21.6.3.4, "Debug Store (DS) Mechanism."		If (CPUID.01H:EDX.DS[21] = 1)
63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
63:32	Reserved if not in IA-32e mode.	
Register Address: 6A0H, 1696		IA32_U_CET
Configure User Mode CET (R/W)		Bits 1:0 are defined if CPUID.07H.00H:ECX.CET_SS[7] = 1. Bits 5:2 and bits 63:10 are defined if CPUID.07H.00H:EDX.CET_IBT[20] = 1.
0	SH_STK_EN: When set to 1, enable shadow stacks at CPL3.	
1	WR_SHSTK_EN: When set to 1, enables the WRSSD/WRSSQ instructions.	
2	ENDBR_EN: When set to 1, enables indirect branch tracking.	
3	LEG_IW_EN: Enable legacy compatibility treatment for indirect branch tracking.	
4	NO_TRACK_EN: When set to 1, enables use of no-track prefix for indirect branch tracking.	
5	SUPPRESS_DIS: When set to 1, disables suppression of CET indirect branch tracking on legacy compatibility.	
9:6	Reserved; must be zero.	
10	SUPPRESS: When set to 1, indirect branch tracking is suppressed. This bit can be written to 1 only if TRACKER is written as IDLE.	
11	TRACKER: Value of the indirect branch tracking state machine. Values: IDLE (0), WAIT_FOR_ENDBRANCH(1).	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)	
Bit Fields	MSR/Bit Description		Comment
63:12	EB_LEG_BITMAP_BASE: Linear address bits 63:12 of a legacy code page bitmap used for legacy compatibility when indirect branch tracking is enabled. If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are used.		
Register Address: 6A2H, 1698		IA32_S_CET	
Configure Supervisor Mode CET (R/W)		See IA32_U_CET (6A0H) for reference; similar format.	
Register Address: 6A4H, 1700		IA32_PL0_SSP	
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 0 will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1	
Register Address: 6A5H, 1701		IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 1 from a higher privilege level will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1	
Register Address: 6A6H, 1702		IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0. Transitions to privilege level 2 from a higher privilege level will check that bit 2 is also 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1	
Register Address: 6A7H, 1703		IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) If the processor does not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are reserved. On processors that support Intel 64 architecture this value cannot represent a non-canonical address. In protected mode, only 31:0 are loaded. Bits 1:0 of the MSR must be 0.		If CPUID.07H.00H:ECX.CET_SS[7] = 1	
Register Address: 6A8H, 1704		IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) This MSR is not present on processors that do not support Intel 64 architecture. This field cannot represent a non-canonical address.		If CPUID.07H.00H:ECX.CET_SS[7] = 1	
Register Address: 6E0H, 1760		IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W)		If CPUID.01H:ECX[24] = 1	
63:0	REGISTER_VALUE TSC-deadline value.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 6E1H, 1761		IA32_PKRS
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W)		If CPUID.07H:00H:ECX.PKS[31] = 1
31:0	For domain i (i between 0 and 15), bits 2i and 2i+1 contain the AD and WD permissions, respectively.	
63:32	Reserved.	
Register Address: 770H, 1904		IA32_PM_ENABLE
Enable/disable HWP (R/W)		If CPUID.06H:EAX[7] = 1
0	HWP_ENABLE (R/W) Note this bit can only be enabled once from the default value. Once set, writes to the HWP_ENABLE bit are ignored. Only RESET will clear this bit. Default = 0. See Section 16.4.2, "Enabling HWP."	If CPUID.06H:EAX[7] = 1
63:1	Reserved.	
Register Address: 771H, 1905		IA32_HWP_CAPABILITIES
HWP Performance Range Enumeration (R/O)		If CPUID.06H:EAX[7] = 1
7:0	Highest_Performance See Section 16.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
15:8	Guaranteed_Performance See Section 16.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
23:16	Most_Efficient_Performance See Section 16.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
31:24	Lowest_Performance See Section 16.4.3, "HWP Performance Range and Dynamic Capabilities."	If CPUID.06H:EAX[7] = 1
63:32	Reserved.	
Register Address: 772H, 1906		IA32_HWP_REQUEST_PKG
Power Management Control Hints for All Logical Processors in a Package (R/W)		If CPUID.06H:EAX[11] = 1
7:0	Minimum_Performance See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
15:8	Maximum_Performance See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
23:16	Desired_Performance See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1
31:24	Energy_Performance_Preference See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1 && CPUID.06H:EAX[10] = 1
41:32	Activity_Window See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[11] = 1 && CPUID.06H:EAX[9] = 1
63:42	Reserved.	
Register Address: 773H, 1907		IA32_HWP_INTERRUPT
Control HWP Native Interrupts (R/W)		If CPUID.06H:EAX[8] = 1
0	EN_Guaranteed_Performance_Change See Section 16.4.6, "HWP Notifications."	If CPUID.06H:EAX[8] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
1	EN_Excursion_Minimum See Section 16.4.6, "HWP Notifications."	If CPUID.06H:EAX[8] = 1
63:2	Reserved.	
Register Address: 774H, 1908		IA32_HWP_REQUEST
Power Management Control Hints to a Logical Processor (R/W)		If CPUID.06H:EAX[7] = 1
7:0	Minimum_Performance See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
15:8	Maximum_Performance See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
23:16	Desired_Performance See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1
31:24	Energy_Performance_Preference See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[10] = 1
41:32	Activity_Window See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[9] = 1
42	Package_Control See Section 16.4.4, "Managing HWP."	If CPUID.06H:EAX[7] = 1 && CPUID.06H:EAX[11] = 1
63:43	Reserved.	
Register Address: 775H, 1909		IA32_PECI_HWP_REQUEST_INFO
IA32_PECI_HWP_REQUEST_INFO		
7:0	Minimum Performance (MINIMUM_PERFORMANCE): Used by OS to read the latest value of Peci minimum performance input. Default value is 0.	
15:8	Maximum Performance (MAXIMUM_PERFORMANCE): Used by OS to read the latest value of Peci maximum performance input. Default value is 0.	
23:16	Reserved.	
31:24	Energy Performance Preference (ENERGY_PERFORMANCE_PREFERENCE): Used by OS to read the latest value of Peci Energy Performance Preference input. Default value is 0.	
59:32	Reserved.	
60	EPP Peci Override (EPP_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Energy Performance Preference input. If set to '1', Peci is overriding the Energy Performance Preference input. If clear (0), OS has control over Energy Performance Preference input. Default value is 0.	
61	Reserved.	
62	Max Peci Override (MAX_PECI_OVERRIDE): Indicates whether Peci is currently overriding the Maximum Performance input. If set to '1', Peci is overriding the Maximum Performance input. If clear (0), OS has control over Maximum Performance input. Default value is 0.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63	Min PECL Override (MIN_PECI_OVERRIDE): Indicates whether PECL is currently overriding the Minimum Performance input. If set to '1', PECL is overriding the Minimum Performance input. If clear (0), OS has control over Minimum Performance input. Default value is 0.	
Register Address: 776H, 1910		IA32_HWP_CTL
IA32_HWP_CTL		If CPUID.06H:EAX[22] = 1
0	PKG_CTL_POLARITY Defines which HWP Request MSR is used whether Thread level or package level. When package MSR is used, the thread MSR valid bits define which thread MSR fields override the package. Default value is 0.	If CPUID.06H:EAX[22] = 1
63:1	Reserved.	
Register Address: 777H, 1911		IA32_HWP_STATUS
Log bits indicating changes to Guaranteed & excursions to Minimum (R/W)		If CPUID.06H:EAX[7] = 1
0	Guaranteed_Performance_Change (R/WCO) See Section 16.4.5, "HWP Feedback."	If CPUID.06H:EAX[7] = 1
1	Reserved.	
2	Excursion_To_Minimum (R/WCO) See Section 16.4.5, "HWP Feedback."	If CPUID.06H:EAX[7] = 1
63:3	Reserved.	
Register Address: 7A3H, 1955		IA32_MCU_EXT_SERVICE
MCU Extended Service (R/O)		If IA32_ARCH_CAPABILITIES[22] = 1
3:0	ALLOWED_PERIODS Value indicates the allowed periods for extended servicing. Value x means that all extended servicing periods are allowed till period x.	
63:4	Reserved.	
Register Address: 7A4H, 1956		IA32_MCU_ROLLBACK_MIN_ID
Minimal MCU Revision ID (R/O) Minimal MCU Revision ID that software can rollback to per boot.		If IA32_MCU_ENUMERATION[3] = 1
31:0	REVISION_ID Minimal MCU revision ID for rollback.	
63:32	Reserved for future use.	
Register Address: 7A5H, 1957		IA32_MCU_STAGING_MBOX_ADDR
IA32_MCU_STAGING_MBOX_ADDR (R/O) Reports MMIO address of MCU staging DOE mailbox.		
63:0	ADDR MMIO address base of MCU staging DOE mailbox.	
Register Address: 7B0H, 1968		IA32_ROLLBACK_SIGN_ID_0

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Rollback ID 0 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B1H, 1969		IA32_ROLLBACK_SIGN_ID_1
Rollback ID 1 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B2H, 1970		IA32_ROLLBACK_SIGN_ID_2
Rollback ID 2 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B3H, 1971		IA32_ROLLBACK_SIGN_ID_3
Rollback ID 3 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B4H, 1972		IA32_ROLLBACK_SIGN_ID_4
Rollback ID 4 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 7B5H, 1973		IA32_ROLLBACK_SIGN_ID_5
Rollback ID 5 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B6H, 1974		IA32_ROLLBACK_SIGN_ID_6
Rollback ID 6 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B7H, 1975		IA32_ROLLBACK_SIGN_ID_7
Rollback ID 7 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B8H, 1976		IA32_ROLLBACK_SIGN_ID_8
Rollback ID 8 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7B9H, 1977		IA32_ROLLBACK_SIGN_ID_9
Rollback ID 9 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:48	Reserved.	
Register Address: 7BAH, 1978		IA32_ROLLBACK_SIGN_ID_10
Rollback ID 10 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BBH, 1979		IA32_ROLLBACK_SIGN_ID_11
Rollback ID 11 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BCH, 1980		IA32_ROLLBACK_SIGN_ID_12
Rollback ID 12 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BDH, 1981		IA32_ROLLBACK_SIGN_ID_13
Rollback ID 13 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BEH, 1982		IA32_ROLLBACK_SIGN_ID_14
Rollback ID 14 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 7BFH, 1983		IA32_ROLLBACK_SIGN_ID_15
Rollback ID 15 (R/O) Holds the Revision ID and SVN of a supported rollback target or 0 if none.		If IA32_MCU_ENUMERATION[3] = 1
31:0	MCU_ROLLBACK_ID MCU supported Rollback ID.	
47:32	ROLLBACK_MCU_SVN MCU SVN corresponding to the reported MCU Rollback ID.	
63:48	Reserved.	
Register Address: 802H, 2050		IA32_X2APIC_APICID
x2APIC ID Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 803H, 2051		IA32_X2APIC_VERSION
x2APIC Version Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 808H, 2056		IA32_X2APIC_TPR
x2APIC Task Priority Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80AH, 2058		IA32_X2APIC_PPR
x2APIC Processor Priority Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80BH, 2059		IA32_X2APIC_EOI
x2APIC EOI Register (W/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80DH, 2061		IA32_X2APIC_LDR
x2APIC Logical Destination Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 80FH, 2063		IA32_X2APIC_SIVR
x2APIC Spurious Interrupt Vector Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 810H, 2064		IA32_X2APIC_ISR0
x2APIC In-Service Register Bits 31:0 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 811H, 2065		IA32_X2APIC_ISR1
x2APIC In-Service Register Bits 63:32 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 812H, 2066		IA32_X2APIC_ISR2
x2APIC In-Service Register Bits 95:64 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits 127:96 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits 159:128 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits 191:160 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits 223:192 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits 255:224 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits 31:0 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits 63:32 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits 95:64 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits 127:96 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits 159:128 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits 191:160 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits 223:192 (R/O)	If (CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1)	
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits 255:224 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	
Register Address: 820H, 2080	IA32_X2APIC_IRRO	
x2APIC Interrupt Request Register Bits 31:0 (R/O)	If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits 63:32 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits 95:64 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits 127:96 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits 159:128 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits 191:160 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits 223:192 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits 255:224 (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Interrupt Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 836H, 2102		IA32_X2APIC_LVT_LINT1
x2APIC LVT LINT1 Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 837H, 2103		IA32_X2APIC_LVT_ERROR
x2APIC LVT Error Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 838H, 2104		IA32_X2APIC_INIT_COUNT
x2APIC Initial Count Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 839H, 2105		IA32_X2APIC_CUR_COUNT
x2APIC Current Count Register (R/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83EH, 2110		IA32_X2APIC_DIV_CONF
x2APIC Divide Configuration Register (R/W)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 83FH, 2111		IA32_X2APIC_SELF_IPI
x2APIC Self IPI Register (W/O)		If CPUID.01H:ECX[21] = 1 && IA32_APIC_BASE.[10] = 1
Register Address: 981H, 2433		IA32_TME_CAPABILITY
Memory Encryption Capability MSR		If CPUID.07H.00H:ECX[13] = 1
0	Support for AES-XTS 128-bit encryption algorithm. (NIST standard)	
1	Support for AES-XTS 128-bit encryption with integrity algorithm.	
2	Support for AES-XTS 256-bit encryption algorithm.	
3	Support for AES-XTS 256-bit encryption with integrity algorithm.	
27:4	Reserved.	
28	Non-zero KeyIDs must be SEAM-private.	If set, any write to IA32_TME_ACTIVATE must write identical values to MK_TME_KEYID_BITS and TDX_RESERVED_KEYID_BITS.
29	Reserved.	
30	SUPPORT_IA32_TME_CLEAR_SAVED_KEY Support for the IA32_TME_CLEAR_SAVED_KEY MSR.	
31	TME encryption bypass supported.	
35:32	MK_TME_MAX_KEYID_BITS Number of bits which can be allocated for usage as key identifiers for multi-key memory encryption. 4 bits allow for a maximum value of 15, which could address 32K keys. Zero if TME-MK is not supported.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
50:36	MK_TME_MAX_KEYS Indicates the maximum number of keys which are available for usage. This value may not be a power of 2. KeyID 0 is specially reserved and is not accounted for in this field.	
63:51	Reserved.	
Register Address: 982H, 2434		IA32_TME_ACTIVATE
Memory Encryption Activation MSR This MSR is used to lock the MSRs listed below. Any write to the following MSRs will be ignored after they are locked. The lock is reset when CPU is reset. ▪ IA32_TME_ACTIVATE ▪ IA32_TME_EXCLUDE_MASK ▪ IA32_TME_EXCLUDE_BASE Note that IA32_TME_EXCLUDE_MASK and IA32_TME_EXCLUDE_BASE must be configured before IA32_TME_ACTIVATE.		If CPUID.07H.00H:ECX[13] = 1
0	Lock R/O – Will be set upon successful WRMSR (or first SMI); written value ignored.	
1	Hardware Encryption Enable This bit also enables TME-MK; TME-MK cannot be enabled without enabling encryption hardware. TME is enabled depending on the TME Encryption Bypass Enable bit (bit 31).	
2	Key Select 0: Create a new TME key (expected cold/warm boot). 1: Restore the TME key from storage (expected when resume from standby).	
3	Save TME Key for Standby Save key into storage to be used when resume from standby. Note: This may not be supported in all processors.	
7:4	TME Policy/Encryption Algorithm Only algorithms enumerated in IA32_TME_CAPABILITY MSR are allowed. Any other values are invalid and will result in #GP. Additionally, any algorithm that supports integrity checking is not allowed to be used for TME even if it is listed as allowed in the IA32_TME_CAPABILITY MSR and will result in #GP.	
30:8	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31	TME Encryption Bypass Enable When encryption hardware is enabled: - Total Memory Encryption is enabled using a CPU generated ephemeral key based on a hardware random number generator when this bit is set to 0. - Total Memory Encryption is bypassed (no encryption/decryption for KeyID0) when this bit is set to 1. <i>On some processors, bypassing TME encryption can provide performance benefits to accesses made with KeyID 0 by avoiding the latency of decryption or encryption and decryption.</i> Software must inspect Hardware Encryption Enable (bit 1) and TME Encryption Bypass Enable (bit 31) to determine if TME encryption is enabled.	
35:32	MK_TME_KEYID_BITS Reserved if TME-MK is not enumerated, otherwise: The number of key identifier bits to allocate to TME-MK usage. Similar to enumeration, this is an encoded value. Writing a value greater than MK_TME_MAX_KEYID_BITS will result in #GP. Writing a non-zero value to this field will #GP if bit 1 of EAX (Hardware Encryption Enable) is not also set to 1, as encryption hardware must be enabled to use TME-MK. Example: To support 255 keys, this field would be set to a value of 8.	
39:36	TDX_RESERVED_KEYID_BITS The number of key identifier bits to allocate to TDX usage, which are allocated from the most significant bit downward. Writing a value greater than MK_TME_KEYID_BITS will result in a #GP. Note: These bits are a subset of the overall KeyID bits which are declared by MK_TME_MAX_KEYID_BITS.	If IA32_TME_CAPABILITY[28] = 1, this value must be the same as that of MK_TME_KEYID_BITS.
47:40	Reserved.	
63:48	MK_TME_CRYPT0_ALGS Reserved if TME-MK is not enumerated, otherwise: Bit 48: AES-XTS 128. Bit 49: AES-XTS 128 with integrity. Bit 50: AES-XTS 256. Bit 51: AES-XTS-256 with integrity. Bit 63:52: Reserved (#GP) Bitmask for BIOS to set which encryption algorithms are allowed for TME-MK, would be later enforced by the key loading ISA ('1' = allowed).	
Register Address: 983H, 2435		IA32_TME_EXCLUDE_MASK
Memory Encryption Exclude Mask The IA32_TME_EXCLUDE_MASK MSR must define a contiguous region. WRMSR will #GP if the TMEEMASK field does not specify a contiguous region. This MSR is locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.		If CPUID.07H.00H:ECX[13] = 1
10:0	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
11	Enable: When set to '1', then TME_EXCLUDE_BASE and TME_EXCLUDE_MASK are used to define an exclusion region for TME/TME-MK (for KeyID=0).	
MAXPHYADDR-1:12	TMEEMASK: This field indicates the bits that must match TMEEBASE in order to qualify as a TME/TME-MK (for KeyID=0) exclusion memory range access.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 984H, 2436		IA32_TME_EXCLUDE_BASE
Memory Encryption Exclude Base Note: Writing '1' into bits greater than the max supported physical size will result in #GP. This MSRs is locked by the IA32_TME_ACTIVATE MSR. If lock=1, then WRMSR to IA32_TME_EXCLUDE_MASK/IA32_TME_EXCLUDE_BASE MSRs will result in #GP.		If CPUID.07H.00H:ECX[13] = 1
11:0	Reserved.	
MAXPHYADDR-1:12	TMEEBASE: Base physical address to be excluded for TME/TME-MK (for KeyID=0) encryption.	
63:MAXPHYADDR	Reserved; must be zero.	
Register Address: 985H, 2437		IA32_UINTR_RR
User Interrupt Request Register (R/W)		If CPUID.07H.01H:EDX[13] = 1
63:0	UIRR Bitmap of requested user interrupt vectors.	
Register Address: 986H, 2438		IA32_UINTR_HANDLER
User Interrupt Handler Address (R/W)		If CPUID.07H.01H:EDX[13] = 1
63:0	UIHANDLER User interrupt handler linear address.	
Register Address: 987H, 2439		IA32_UINTR_STACKADJUST
User Interrupt Stack Adjustment (R/W)		If CPUID.07H.01H:EDX[13] = 1
0	LOAD_RSP User interrupt stack mode.	
2:1	Reserved.	
63:3	STACK_ADJUST Stack adjust value.	
Register Address: 988H, 2440		IA32_UINTR_MISC
User-Interrupt Target-Table Size and Notification Vector (R/W)		If CPUID.07H.01H:EDX[13] = 1
31:0	UITTSZ The highest index of a valid entry in the user-interrupt target table. Valid entries are indices 0..UITTSZ (inclusive).	
39:32	UINV User-interrupt notification vector.	
63:40	Reserved.	
Register Address: 989H, 2441		IA32_UINTR_PD
User Interrupt PID Address (R/W)		If CPUID.07H.01H:EDX[13] = 1

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
5:0	Reserved.	
63:6	UPIDADDR User-interrupt notification processing accesses a UPID at this linear address.	
Register Address: 98AH, 2442		IA32_UINTR_TT
User-Interrupt Target Table (R/W)		If CPUID.07H.01H:EDX[13] = 1
0	SENDUIPI_ENABLE User-interrupt target table is valid.	
3:1	Reserved.	
63:4	UITTADDR User-interrupt target table base linear address.	
Register Address: 990H, 2448		IA32_COPY_STATUS ⁴
Status of Most Recent Platform to Local or Local to Platform Copies (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	IWKEY_COPY_SUCCESSFUL Status of most recent copy to or from IwKeyBackup.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: 991H, 2449		IA32_IWKEYBACKUP_STATUS ⁵
Information about IwKeyBackup Register (R/O)		If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	Backup/Restore Valid Cleared when a write to IwKeyBackup is initiated, and then set when the latest write of IwKeyBackup has been written to storage that persists across S3/S4 sleep state. If S3/S4 is entered between when an IwKeyBackup write occurs and when this bit is set, then IwKeyBackup may not be recovered after S3/S4 exit. During S3/S4 sleep state exit (system wake up), this bit is cleared. It is set again when IwKeyBackup is restored from persistent storage and thus available to be copied to IwKey using IA32_COPY_PLATFORM_TO_LOCAL MSR. Another write to IwKeyBackup (via IA32_COPY_LOCAL_TO_PLATFORM MSR) may fail if a previous write has not yet set this bit.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
1	Reserved.	
2	Backup Key Storage Read/Write Error Updated prior to backup/restore valid being set. Set when an error is encountered while backing up or restoring a key to persistent storage.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
3	IwKeyBackup Consumed Set after the previous backup operation has been consumed by the platform. This does not indicate that the system is ready for a second IwKeyBackup write as the previous IwKeyBackup write may still need to set Backup/restore valid.	If ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:4	Reserved.	
Register Address: 9FBH, 2555		IA32_TME_CLEAR_SAVED_KEY
IA32_TME_CLEAR_SAVED_KEY (W/O)		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
0	TME_CLEAR_SAVED_KEY Clear saved TME keys.	
63:1	Reserved.	
Register Address: C80H, 3200		IA32_DEBUG_INTERFACE
Silicon Debug Feature Control (R/W)		If CPUID.01H:ECX[11] = 1
0	Enable (R/W) BIOS set 1 to enable Silicon debug features. Default is 0.	If CPUID.01H:ECX[11] = 1
29:1	Reserved.	
30	Lock (R/W): If 1, locks any further change to the MSR. The lock bit is set automatically on the first SMI assertion even if not explicitly set by BIOS. Default is 0.	If CPUID.01H:ECX[11] = 1
31	Debug Occurred (R/O): This “sticky bit” is set by hardware to indicate the status of bit 0. Default is 0.	If CPUID.01H:ECX[11] = 1
63:32	Reserved.	
Register Address: C81H, 3201		IA32_L3_QOS_CFG
L3 QOS Configuration (R/W)		If (CPUID.10H.01H:ECX[2] = 1)
0	Enable (R/W) Set 1 to enable L3 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C82H, 3202		IA32_L2_QOS_CFG
L2 QOS Configuration (R/W)		If (CPUID.10H.02H:ECX[2] = 1)
0	Enable (R/W) Set 1 to enable L2 CAT masks and CLOS to operate in Code and Data Prioritization (CDP) mode.	
63:1	Reserved. Attempts to write to reserved bits result in a #GP(0).	
Register Address: C83H, 3203		IA32_L3_IO_QOS_CFG
L3 I/O QOS Configuration (R/W) This MSR is used to enable the I/O RDT features.		If (CPUID.0FH.01H:EAX[10:9] = 1)
0	L3 I/O RDT Allocation Enable.	
1	L3 I/O RDT Monitoring Enable.	
63:2	Reserved.	
Register Address: C88H, 3208		IA32_RESOURCE_PRIORITY
Thread scope Resource Priority Enable (R/W)		
0	ENABLE When set, enables model specific features that can be used to create a Resource Priority mode.	
63:1	Reserved.	
Register Address: C89H, 3209		IA32_RESOURCE_PRIORITY_PKG
IA32_RESOURCE_PRIORITY_PKG (R/W)		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
0	ENABLE Enable Resource Priority feature.	
63:1	Reserved.	
Register Address: C8DH, 3213		IA32_QM_EVTSEL
Monitoring Event Select Register (R/W)		If (CPUID.07H.00H:EBX[12] = 1)
7:0	Event ID: ID of a supported monitoring event to report via IA32_QM_CTR.	
31:8	Reserved.	
N+31:32	Resource Monitoring ID: ID for monitoring hardware to report monitored data via IA32_QM_CTR.	N = Ceil (Log ₂ (CPUID.0FH.00H:EBX[31:0] + 1))
63:N+32	Reserved.	
Register Address: C8EH, 3214		IA32_QM_CTR
Monitoring Counter Register (R/O)		If (CPUID.07H.00H:EBX[12] = 1)
61:0	Resource Monitored Data.	
62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
Register Address: C8FH, 3215		IA32_PQR_ASSOC
Resource Association Register (R/W)		If ((CPUID.07H.00H:EBX[12] = 1) or (CPUID.07H.00H:EBX[15] = 1))
N-1:0	Resource Monitoring ID (R/W): ID for monitoring hardware to track internal operation, e.g., memory access.	N = Ceil (Log ₂ (CPUID.0FH.00H:EBX[31:0] + 1))
31:N	Reserved.	
63:32	CLOS (R/W): The class of service (CLOS) to enforce (on writes); returns the current CLOS when read.	If (CPUID.07H.00H:EBX[15] = 1)
Register Address: C90H–D8FH, 3216–3471		Reserved MSR Address Space for CAT Mask Registers
See Section 19.19.4.1, “Enumeration and Detection Support of Cache Allocation Technology.”		
Register Address: C90H, 3216		IA32_L3_MASK_0
L3 CAT Mask for COS0 (R/W)		If (CPUID.10H.00H:EBX[1] != 0)
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: C90H+n, 3216+n		IA32_L3_MASK_n
L3 CAT Mask for COSn (R/W)		n = CPUID.10H.01H:EDX[15:0]
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D10H–D4FH, 3344–3407		Reserved MSR Address Space for L2 CAT Mask Registers
See Section 19.19.4.1, “Enumeration and Detection Support of Cache Allocation Technology.”		
Register Address: D10H, 3344		IA32_L2_MASK_0
L2 CAT Mask for COS0 (R/W)		If (CPUID.10H.00H:EBX[2] != 0)

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D10H+n, 3344+n		IA32_L2_MASK_n
L2 CAT Mask for COSn (R/W)		n = CPUID.10H.02H:EDX[15:0]
31:0	Capacity Bit Mask (R/W)	
63:32	Reserved.	
Register Address: D18H, 3352		IA32_L2_MASK_8
L2 CAT Mask for COS8 (R/W)		
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. '1' in bit indicates allocation to the way is allowed. '0' indicates allocation to the way is not allowed.	
63:16	Reserved.	
Register Address: D19H, 3353		IA32_L2_MASK_9
L2 CAT Mask for COS9 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1AH, 3354		IA32_L2_MASK_10
L2 CAT Mask for COS10 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1BH, 3355		IA32_L2_MASK_11
L2 CAT Mask for COS11 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1CH, 3356		IA32_L2_MASK_12
L2 CAT Mask for COS12 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1DH, 3357		IA32_L2_MASK_13
L2 CAT Mask for COS13 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1EH, 3358		IA32_L2_MASK_14
L2 CAT Mask for COS14 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D1FH, 3359		IA32_L2_MASK_15
L2 CAT Mask for COS15 (R/W) See IA32_L2_MASK_8 (D18H) for reference; similar format.		
Register Address: D50H, 3408		IA32_L2_QOS_EXT_BW_THRTL_0
IA32_L2_QOS_EXT_BW_THRTL_0 (R/W) Memory Bandwidth enforcement for COS0.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 0
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:7	Reserved.	
Register Address: D51H, 3409		IA32_L2_QOS_EXT_BW_THRTL_1
IA32_L2_QOS_EXT_BW_THRTL_1 (R/W) Memory Bandwidth enforcement for COS1.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 1
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D52H, 3410		IA32_L2_QOS_EXT_BW_THRTL_2
IA32_L2_QOS_EXT_BW_THRTL_2 (R/W) Memory Bandwidth enforcement for COS2.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 2
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D53H, 3411		IA32_L2_QOS_EXT_BW_THRTL_3
IA32_L2_QOS_EXT_BW_THRTL_3 (R/W) Memory Bandwidth enforcement for COS3.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 3
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D54H, 3412		IA32_L2_QOS_EXT_BW_THRTL_4
IA32_L2_QOS_EXT_BW_THRTL_4 (R/W) Memory Bandwidth enforcement for COS4.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 4
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D55H, 3413		IA32_L2_QOS_EXT_BW_THRTL_5
IA32_L2_QOS_EXT_BW_THRTL_5 (R/W) Memory Bandwidth enforcement for COS5.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 5
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D56H, 3414		IA32_L2_QOS_EXT_BW_THRTL_6
IA32_L2_QOS_EXT_BW_THRTL_6 (R/W) Memory Bandwidth enforcement for COS6.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 6
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D57H, 3415		IA32_L2_QOS_EXT_BW_THRTL_7

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_L2_QOS_EXT_BW_THRTL_7 (R/W) Memory Bandwidth enforcement for COS7.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 7
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D58H, 3416		IA32_L2_QOS_EXT_BW_THRTL_8
IA32_L2_QOS_EXT_BW_THRTL_8 (R/W) Memory Bandwidth enforcement for COS8.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 8
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D59H, 3417		IA32_L2_QOS_EXT_BW_THRTL_9
IA32_L2_QOS_EXT_BW_THRTL_9 (R/W) Memory Bandwidth enforcement for COS9.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 9
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5AH, 3418		IA32_L2_QOS_EXT_BW_THRTL_10
IA32_L2_QOS_EXT_BW_THRTL_10 (R/W) Memory Bandwidth enforcement for COS10.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 10
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5BH, 3419		IA32_L2_QOS_EXT_BW_THRTL_11
IA32_L2_QOS_EXT_BW_THRTL_11 (R/W) Memory Bandwidth enforcement for COS11.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 11
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5CH, 3420		IA32_L2_QOS_EXT_BW_THRTL_12
IA32_L2_QOS_EXT_BW_THRTL_12 (R/W) Memory Bandwidth enforcement for COS12.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 12
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5DH, 3421		IA32_L2_QOS_EXT_BW_THRTL_13
IA32_L2_QOS_EXT_BW_THRTL_13 (R/W) Memory Bandwidth enforcement for COS13.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 13

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D5EH, 3422		IA32_L2_QOS_EXT_BW_THRTL_14
IA32_L2_QOS_EXT_BW_THRTL_14 (R/W) Memory Bandwidth enforcement for COS14.		CPUID.10H.00H:EBX[3] and CPUID.10H.03H:EDX ≥ 14
6:0	RBE_ENFORCEMENT_VAL Max Delay value cannot be greater than 90 percent - 0x5a.	
63:7	Reserved.	
Register Address: D90H, 3472		IA32_BNDCFGS
Supervisor State of MPX Configuration (R/W)		If (CPUID.07H.00H:EBX[14] = 1)
0	EN: Enable Intel MPX in supervisor mode.	
1	BNDPRESERVE: Preserve the bounds registers for near branch instructions in the absence of the BND prefix.	
11:2	Reserved, must be zero.	
63:12	Base Address of Bound Directory.	
Register Address: D91H, 3473		IA32_COPY_LOCAL_TO_PLATFORM ⁵
Copy Local State to Platform State (W)		IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	lwKeyBackup Copy lwKey to lwKeyBackup.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: D92H, 3474		IA32_COPY_PLATFORM_TO_LOCAL ⁵
Copy Platform State to Local State (W)		IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
0	lwKeyBackup Copy lwKeyBackup to lwKey.	IF ((CPUID.19H:EBX[4] = 1) && (CPUID.07H.00H:ECX[23] = 1))
63:1	Reserved.	
Register Address: D93H, 3475		IA32_PASID
Process Address Space Identifier. (R/W)		
19:0	Process address space identifier (PASID). Specifies the PASID of the currently running software thread.	
30:20	Reserved.	
31	Valid. Execution of ENQCMD causes a #GP if this bit is clear.	
63:32	Reserved.	
Register Address: DA0H, 3488		IA32_XSS
Extended Supervisor State Mask (R/W)		If (CPUID.0DH.01H:EAX[3] = 1)
7:0	Reserved.	
8	PT State (R/W)	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
9	Reserved.	
10	PASID State (R/W)	
11	CET_U State (R/W)	
12	CET_S State (R/W)	
13	HDC State (R/W)	
14	UINTR State (R/W)	
15	LBR State (R/W)	
16	HWP State (R/W)	
63:17	Reserved.	
Register Address: DB0H, 3504		IA32_PKG_HDC_CTL
Package Level Enable/Disable HDC (R/W)		If CPUID.06H:EAX[13] = 1
0	HDC_Pkg_Enable (R/W) Force HDC idling or wake up HDC-idled logical processors in the package. See Section 16.5.2, "Package level Enabling HDC."	If CPUID.06H:EAX[13] = 1
63:1	Reserved.	
Register Address: DB1H, 3505		IA32_PM_CTL1
Enable/Disable the HDC Thread Level Activity (R/W)		If CPUID.06H:EAX[13] = 1
0	SDC_ALLOWED (R/W) Set this bit to allow this thread to be forced into HDC idle state. Clearing this bit blocks HDC-enter (HW) request. Default value: 1. See Section 16.5.3.	If CPUID.06H:EAX[13] = 1
63:1	Reserved.	
Register Address: DB2H, 3506		IA32_THREAD_STALL
Per-Logical_Processor_ID HDC Idle Residency (R/O)		If CPUID.06H:EAX[13] = 1
63:0	Stall_Cycle_Cnt (R/W) Stalled cycles due to HDC forced idle on this logical processor. See Section 16.5.4.1.	If CPUID.06H:EAX[13] = 1
Register Address: E00H, 3584		IA32_QOS_CORE_BW_THRTL_0
CBA Levels Based on COS for Bandwidth Throttling (R/W)		CPUID.10H.00H:EBX[5] = 1
3:0	COS0_LEVEL CBA Level for COS[0]. Levels are programmed from 0 to 15.	
7:4	Reserved.	
11:8	COS1_LEVEL CBA Level for COS[1]. Levels are programmed from 0 to 15.	
15:12	Reserved.	
19:16	COS2_LEVEL CBA Level for COS[2]. Levels are programmed from 0 to 15.	
25:20	Reserved.	
27:24	COS3_LEVEL CBA Level for COS[3]. Levels are programmed from 0 to 15.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
31:28	Reserved.	
35:32	COS4_LEVEL CBA Level for COS[4]. Levels are programmed from 0 to 15.	
39:36	Reserved.	
43:40	COS5_LEVEL CBA Level for COS[5]. Levels are programmed from 0 to 15.	
47:44	Reserved.	
51:48	COS6_LEVEL CBA Level for COS[6]. Levels are programmed from 0 to 15.	
Register Address: E01H, 3585		IA32_QOS_CORE_BW_THRTL_1
CBA Levels Based on COS for Bandwidth Throttling (R/W)		CPUID.10H.00H:EBX[5] = 1
3:0	COS8_LEVEL CBA Level for COS[8]. Levels are programmed from 0 to 15.	
7:4	Reserved.	
11:8	COS9_LEVEL CBA Level for COS[9]. Levels are programmed from 0 to 15.	
15:12	Reserved.	
19:16	COS10_LEVEL CBA Level for COS[10]. Levels are programmed from 0 to 15.	
25:20	Reserved.	
27:24	COS11_LEVEL CBA Level for COS[11]. Levels are programmed from 0 to 15.	
31:28	Reserved.	
35:32	COS12_LEVEL CBA Level for COS[12]. Levels are programmed from 0 to 15.	
39:36	Reserved.	
43:40	COS13_LEVEL CBA Level for COS[13]. Levels are programmed from 0 to 15.	
47:44	Reserved.	
51:48	COS14_LEVEL CBA Level for COS[14]. Levels are programmed from 0 to 15.	
55:50	Reserved.	
59:56	COS15_LEVEL CBA Level for COS[15]. Levels are programmed from 0 to 15.	
63:60	Reserved	
Register Address: 1200H–121FH, 4608–4639		IA32_LBR_x_INFO
Last Branch Record Entry X Info Register (R/W) An attempt to read or write IA32_LBR_x_INFO such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:0	CYC_CNT The elapsed CPU cycles (saturating) since the last LBR was recorded. See Section 18.1.3.3.	Reset Value: 0
55:16	Undefined, may be zero or non-zero. Writes of non-zero values do not fault, but reads may return a different value.	Reset Value: 0
59:56	BR_TYPE The branch type recorded by this LBR. Encodings: 0000B: COND 0001B: JMP Indirect 0010B: JMP Direct 0011B: CALL Indirect 0100B: CALL Direct 0101B: RET 011xB: Reserved 1xxxB: Other Branch	Reset Value: 0
60	CYC_CNT_VALID CYC_CNT value is valid. See Section 20.1.3.3.	Reset Value: 0
61	TSX_ABORT This LBR record is a TSX abort. On processors that do not support Intel TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
62	IN_TSX This LBR record records a branch that retired during a TSX transaction. On processors that do not support Intel TSX (CPUID.07H.00H:EBX.HLE[4] = 0 and CPUID.07H.00H:EBX.RTM[11] = 0), this bit is undefined.	Reset Value: 0
63	MISPRED The recorded branch direction (conditional branch) or target (indirect branch) was mispredicted.	Reset Value: 0
Register Address: 1400H, 5120		IA32_SEAMRR_BASE
SEAM Memory Range Register for TDX - Base Address (R/W)		IA32_MTRRCAP.SEAMRR[15] = 1
2:0	Reserved.	
3	CONFIGURED When set to 1, the SEAM range is configured.	
24:4	Reserved.	
M-1:25	BASE SEAM Range Register BASE address.	M is the value enumerated by CPUID.80000008H:EAX[7:0] Bits M-1:M-k must be 0, where k = IA32_TME_ACTIVATE.MK_TME_KEYID_BITS
63:M	Reserved.	
Register Address: 1401H, 5121		IA32_SEAMRR_MASK

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
SEAM Memory Range Register - Address Mask (R/W)		IA32_MTRRCAP.SEAMRR[15] = 1
9:0	Reserved.	
10	LOCK	When the LOCK bit is set, the IA32_SEAMRR_BASE/MASK registers are non-writable.
11	VALID Returns 1 when SEAM range protections are active	Read-only. Must be written with 0.
24:12	Reserved.	
M-1:25	MASK Mask value for SEAMRR matching. Lowest granularity is 32MB.	M is the value enumerated by CPUID.80000008H:EAX[7:0]
63:M	Reserved.	
Register Address: 1406H, 5126		IA32_MCU_CONTROL
MCU Control (R/W) Controls the behavior of the Microcode Update Trigger MSR, IA32_BIOS_UPDT_TRIG.		If CPUID.07H.00H:EDX[29] = 1 && IA32_ARCH_CAPABILITIES.MCU_CONTROL = 1
0	LOCK Once set, further writes to this MSR will cause a #GP(0) fault. Bypassed during SMM if EN_SMM_BYPASS (bit 2) is set.	
1	DIS_MCU_LOAD If this bit is set on a given logical processor, then any subsequent attempts to load a microcode update by that logical processor will be silently dropped (WRMSR 0x79 has no effect).	
2	EN_SMM_BYPASS If set, then writes to IA32_MCU_CONTROL are allowed during SMM regardless of the LOCK bit. This enables BIOS to Opt-In to the SMM Bypass functionality.	
63:3	Reserved.	
Register Address: 14CEH, 5326		IA32_LBR_CTL
Last Branch Record Enabling and Configuration Register (R/W)		
0	LBREn When set, enables LBR recording.	Reset Value: 0
1	OS When set, allows LBR recording when CPL == 0.	Reset Value: 0
2	USR When set, allows LBR recording when CPL != 0.	Reset Value: 0
3	CALL_STACK When set, records branches in call-stack mode. See Section 20.1.2.4.	Reset Value: 0
15:4	Reserved.	Reset Value: 0
16	COND When set, records taken conditional branches. See Section 20.1.2.3.	
17	NEAR_REL_JMP When set, records near relative JMPs. See Section 20.1.2.3.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
18	NEAR_IND_JMP When set, records near indirect JMPs. See Section 20.1.2.3.	
19	NEAR_REL_CALL When set, records near relative CALLs. See Section 20.1.2.3.	
20	NEAR_IND_CALL When set, records near indirect CALLs. See Section 20.1.2.3.	
21	NEAR_RET When set, records near RETs. See Section 20.1.2.3.	
22	OTHER_BRANCH When set, records other branches. See Section 20.1.2.3.	
63:23	Reserved.	
Register Address: 14CFH, 5327		IA32_LBR_DEPTH
Last Branch Record Maximum Stack Depth Register (R/W)		
N:0	DEPTH The number of LBRs to be used for recording. Supported values are indicated by the bitmap in CPUID.1CH.00H:EAX[7:0]. The reset value will match the maximum supported by the CPU. Writes of unsupported values will #GP fault.	Reset Value: Varies
63:N+1	Reserved.	Reset Value: 0
Register Address: 1500H–151FH, 5376–5407		IA32_LBR_x_FROM_IP
Last Branch Record entry X source IP register (R/W). An attempt to read or write IA32_LBR_x_FROM_IP such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
63:0	FROM_IP The source IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
Register Address: 1600H–161FH, 5632–5663		IA32_LBR_x_TO_IP
Last Branch Record Entry X Destination IP Register (R/W). An attempt to read or write IA32_LBR_x_TO_IP such that $x \geq \text{IA32\_LBR\_DEPTH.DEPTH}$ will #GP.		
63:0	TO_IP The destination IP of the recorded branch or event, in canonical form. Writes to bits above MAXLINADDR-1 are ignored.	Reset Value: 0
Register Address: 17D0H, 6096		IA32_HW_FEEDBACK_PTR
Hardware Feedback Interface Pointer		If CPUID.06H:EAX[19] = 1
0	Valid (R/W) When set to 1, indicates a valid pointer is programmed into the ADDR field of the MSR.	
11:1	Reserved.	
MAXPHYADDR-1:12	ADDR (R/W) Physical address of the page frame of the first page of the hardware feedback interface structure.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
63:MAXPHYADDR	Reserved.	
Register Address: 17D1H, 6097		IA32_HW_FEEDBACK_CONFIG
Hardware Feedback Interface Configuration		If CPUID.06H:EAX[19] = 1
0	Enable (R/W) When set to 1, enables the hardware feedback interface.	
63:1	Reserved.	
Register Address: 17D2H, 6098		IA32_THREAD_FEEDBACK_CHAR
Thread Feedback Characteristics (R/O)		If CPUID.06H:EAX[23] = 1
7:0	Application Class ID, pointing into the Intel Thread Director structure.	
62:8	Reserved.	
63	Valid bit. When set to 1 the OS Scheduler can use the Class ID (in bits 7:0) for its scheduling decisions. If this bit is 0, the Class ID field should be ignored. It is recommended that the OS uses the last known Class ID of the software thread for its scheduling decisions.	
Register Address: 17D4H, 6100		IA32_HW_FEEDBACK_THREAD_CONFIG
Hardware Feedback Thread Configuration (R/W)		
0	Enables Intel Thread Director. When set to 1, logical processor scope Intel Thread Director is enabled. Default is 0 (disabled).	
63:1	Reserved.	
Register Address: 17DAH, 6106		IA32_HRESET_ENABLE
History Reset Enable (R/W)		
0	Enable reset of the Intel Thread Director history.	
31:1	Reserved for other capabilities that can be reset by the HRESET instruction.	
63:32	Reserved.	
Register Address: 1900H, 6400		IA32_PMC_GPO_CTR
Full Width Writable General Performance Counter 0 (R/W)		If CPUID.0AH:EAX[15:8] > 0 and IA32_PERF_CAPABILITIES[13] = 1
47:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:48	Reserved.	
Register Address: 1901H, 6401		IA32_PMC_GPO_CFG_A
IA32_PMC_GPO_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 0.		If CPUID.0AH:EAX[15:8] > 0
7:0	EVENT_SELECT Selects a performance event logic unit.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
15:8	UMASK Qualifies the microarchitectural condition to detect on the selected event logic.	
16	USR When set, events are counted only when the processor is operating at privilege levels 1, 2 or 3. This flag can be used in conjunction with the OS flag.	
17	OS When set, events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USER flag.	
18	EDGE When set, enables edge detection of events.	
19	Reserved.	
20	INT When set, the processor generates an exception through its local APIC on counter overflow for this counter's thread.	
21	ANYTHREAD If CPUID.0AH:EDX[15] is 1, then this bit is deprecated. When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
22	ENABLE When set, performance counting is enabled in the performance-monitoring counter; when clear, the counter is disabled.	
23	INVERT Inverts the result of the counter-mask (CMASK) comparison when set, so that both greater than equal to and less than comparisons can be made. 0: The comparison is: threshold is greater than or equal to the event 1: The comparison is inverted: threshold is less than event.	
31:24	CMASK When CMASK is not zero, the corresponding performance counter increments by 1 each cycle if the event count is $\geq$ CMASK. This mask enables counting cycles in which multiple occurrences happen (for example, two or more instructions retired per clock).	
34:32	Reserved.	
35	EN_LBR_LOG When set enables updating LBRs with that counters event occurrences, if selected event is precise.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
36	EQUAL When EQ flag is set and the INV flag is clear, the comparison evaluates to true if the selected performance monitoring event (the event) is equal to the specified Counter Mask value (CMask). When EQ flag is set and INV flag is set, the comparison evaluates to true if the event is less-than the CMask value and the event is not zero. Note if CMask is zero, the EQ flag is ignored.	
39:37	Reserved.	
47:40	UMASK2 Unit mask 2 (UMASK2) field (bits 40 through 47) - These bits qualify the condition that the selected event logic unit detects. Valid UMASK2 values for each event logic unit are specific to the unit. The new UMASK2 field may also be used in conjunction with UMASK.	
63:48	Reserved.	
Register Address: 1903H, 6403		IA32_PMC_GPO_CFG_C
IA32_PMC_GPO_CFG_C (R/W) Extended Perf event selector for GP counter 0.		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1904H, 6404		IA32_PMC_GP1_CTR
Full Width Writable General Performance Counter 1 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 1 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1905H, 6405		IA32_PMC_GP1_CFG_A
IA32_PMC_GP1_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 1. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 1
Register Address: 1907H, 6407		IA32_PMC_GP1_CFG_C
IA32_PMC_GP1_CFG_C (R/W) Extended Perf event selector for GP counter 1. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 1908H, 6408		IA32_PMC_GP2_CTR
Full Width Writable General Performance Counter 2 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 2 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1909H, 6409		IA32_PMC_GP2_CFG_A
IA32_PMC_GP2_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 2. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 2
Register Address: 190AH, 6410		IA32_PMC_GP2_CFG_B

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GP2_CFG_B (R/W)		
GP counter reload configuration register.		
1:0	Reserved.	
2	RELOAD_PMC2 Reload GP2 when GP2 overflows.	
3	RELOAD_PMC3 Reload GP2 when GP3 overflows.	
4	RELOAD_PMC4 Reload GP2 when GP4 overflows.	
5	RELOAD_PMC5 Reload GP2 when GP5 overflows.	
6	RELOAD_PMC6 Reload GP2 when GP6 overflows.	
7	RELOAD_PMC7 Reload GP2 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload GP2 when FC0 overflows.	
33	RELOAD_FC1 Reload GP2 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of GP2.	
63:49	Reserved.	
Register Address: 190BH, 6411		IA32_PMC_GP2_CFG_C
IA32_PMC_GP2_CFG_C (R/W)		
Extended Perf event selector for GP counter 2.		
See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 190CH, 6412		IA32_PMC_GP3_CTR
Full Width Writable General Performance Counter 3 (R/W)		If CPUID.0AH:EAX[15:8] > 3 and IA32_PERF_CAPABILITIES[13] = 1
See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		
Register Address: 190DH, 6413		IA32_PMC_GP3_CFG_A
IA32_PMC_GP3_CFG_A (R/W)		If CPUID.0AH:EAX[15:8] > 3
Performance Event Select Register used to control the operation of the General Performance Counter 3. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		
Register Address: 190EH, 6414		IA32_PMC_GP3_CFG_B
IA32_PMC_GP3_CFG_B (R/W)		
GP counter reload configuration register.		
See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 190FH, 6415		IA32_PMC_GP3_CFG_C
IA32_PMC_GP3_CFG_C (R/W) Extended Perf event selector for GP counter 3. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 1910H, 6416		IA32_PMC_GP4_CTR
Full Width Writable General Performance Counter 4 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 4 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1911H, 6417		IA32_PMC_GP4_CFG_A
IA32_PMC_GP4_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 4. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 4
Register Address: 1912H, 6418		IA32_PMC_GP4_CFG_B
IA32_PMC_GP4_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1913H, 6419		IA32_PMC_GP4_CFG_C
IA32_PMC_GP4_CFG_C (R/W) Extended Perf event selector for GP counter 4. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 1914H, 6420		IA32_PMC_GP5_CTR
Full Width Writable General Performance Counter 5 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 5 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1915H, 6421		IA32_PMC_GP5_CFG_A
IA32_PMC_GP5_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 5. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 5
Register Address: 1916H, 6422		IA32_PMC_GP5_CFG_B
IA32_PMC_GP5_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 1917H, 6423		IA32_PMC_GP5_CFG_C
IA32_PMC_GP5_CFG_C (R/W) Extended Perf event selector for GP counter 5. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 1918H, 6424		IA32_PMC_GP6_CTR
Full Width Writable General Performance Counter 6 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 6 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1919H, 6425		IA32_PMC_GP6_CFG_A

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GP6_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 6. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 6
Register Address: 191AH, 6426		IA32_PMC_GP6_CFG_B
IA32_PMC_GP6_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 191BH, 6427		IA32_PMC_GP6_CFG_C
IA32_PMC_GP6_CFG_C (R/W) Extended Perf event selector for GP counter 6. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 191CH, 6428		IA32_PMC_GP7_CTR
Full Width Writable General Performance Counter 7 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 7 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 191DH, 6429		IA32_PMC_GP7_CFG_A
IA32_PMC_GP7_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 7. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 7
Register Address: 191EH, 6430		IA32_PMC_GP7_CFG_B
IA32_PMC_GP7_CFG_B (R/W) GP counter reload configuration register. See IA32_PMC_GP2_CFG_B (190AH) for reference; similar format.		
Register Address: 191FH, 6431		IA32_PMC_GP7_CFG_C
IA32_PMC_GP7_CFG_C (R/W) Extended Perf event selector for GP counter 7. See IA32_PMC_GPO_CFG_C (1903H) for reference; similar format.		
Register Address: 1920H, 6432		IA32_PMC_GP8_CTR
Full Width Writable General Performance Counter 8 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 8 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1921H, 6433		IA32_PMC_GP8_CFG_A
IA32_PMC_GP8_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 8. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 8
Register Address: 1924H, 6436		IA32_PMC_GP9_CTR
Full Width Writable General Performance Counter 9 (R/W) See IA32_PMC_GPO_CTR (1900H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 9 and IA32_PERF_CAPABILITIES[13] = 1
Register Address: 1925H, 6437		IA32_PMC_GP9_CFG_A

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA32_PMC_GP9_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 9. See IA32_PMC_GPO_CFG_A (1901H) for reference; similar format.		If CPUID.0AH:EAX[15:8] > 9
Register Address: 1980H, 6528		IA32_PMC_FX0_CTR
Fixed-Function Performance Counter 0 (R/W) Instructions retired.		If CPUID.0AH:EDX[4:0] > 0
47:0	FIXED_COUNTER Instructions Retired Counter.	
63:46	Reserved.	
Register Address: 1982H, 6530		IA32_PMC_FX0_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter0 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter0 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter0 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter0 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter0 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter0 when GP7 overflows.	
33:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter0 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter0 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 0.	
63:49	Reserved.	
Register Address: 1983H, 6531		IA32_PMC_FX0_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 0 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 1984H, 6532		IA32_PMC_FX1_CTR
Fixed-Function Performance Counter 1 (R/W) Unhalted core clock cycles.		If CPUID.0AH:EDX[4:0] > 1
47:0	FIXED_COUNTER Unhalted core clock cycles counter.	
63:46	Reserved.	
Register Address: 1986H, 6534		IA32_PMC_FX1_CFG_B
Fixed-Function Counter Reload Configuration Register (R/W)		
1:0	Reserved.	
2	RELOAD_PMC2 Reload Fixed-Function Counter1 when GP2 overflows.	
3	RELOAD_PMC3 Reload Fixed-Function Counter1 when GP3 overflows.	
4	RELOAD_PMC4 Reload Fixed-Function Counter1 when GP4 overflows.	
5	RELOAD_PMC5 Reload Fixed-Function Counter1 when GP5 overflows.	
6	RELOAD_PMC6 Reload Fixed-Function Counter1 when GP6 overflows.	
7	RELOAD_PMC7 Reload Fixed-Function Counter1 when GP7 overflows.	
31:8	Reserved.	
32	RELOAD_FC0 Reload Fixed-Function Counter1 when FC0 overflows.	
33	RELOAD_FC1 Reload Fixed-Function Counter1 when FC1 overflows.	
47:34	Reserved.	
48	METRICS_CLEAR Clear PERF_METRICS on overflow of Fixed-Function Counter 1.	
63:49	Reserved.	
Register Address: 1987H, 6532		IA32_PMC_FX1_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 1 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1988H, 6536		IA32_PMC_FX2_CTR
Fixed-Function Performance Counter 2 (R/W) Unhalted core reference cycles.		If CPUID.0AH:EDX[4:0] > 2

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
47:0	FIXED_COUNTER Unhalted core reference cycles counter.	
63:48	Reserved.	
Register Address: 198BH, 6539		IA32_PMC_FX2_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 2 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 198CH, 6540		IA32_PMC_FX3_CTR
Fixed-Function Performance Counter 3 (R/W) Top-down Microarchitecture Analysis unhalted number of available slots.		If CPUID.0AH:EDX[4:0] > 3
47:0	FIXED_COUNTER Top-down microarchitecture analysis unhalted number of available slots counter.	
63:48	Reserved.	
Register Address: 1990H, 6544		IA32_PMC_FX4_CTR
Fixed-Function Performance Counter 4 (R/W) Top-down bad speculation.		If CPUID.0AH:EDX[4:0] > 4
47:0	FIXED_COUNTER Top-down bad speculation counter.	
63:48	Reserved.	
Register Address: 1993H, 6547		IA32_PMC_FX4_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 4 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1994H, 6548		IA32_PMC_FX5_CTR
Fixed-Function Performance Counter 5 (R/W) Top-down frontend bound.		If CPUID.0AH:EDX[4:0] > 5
47:0	FIXED_COUNTER Top-down frontend-bound counter.	
63:48	Reserved.	
Register Address: 1997H, 6551		IA32_PMC_FX5_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 5 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
Register Address: 1998H, 6552		IA32_PMC_FX6_CTR
Fixed-Function Performance Counter 6 (R/W) Top-down retiring.		If CPUID.0AH:EDX[4:0] > 6
47:0	FIXED_COUNTER Top-down retiring counter.	
63:48	Reserved.	
Register Address: 199BH, 6555		IA32_PMC_FX6_CFG_C
Extended Perf Event Selector for Fixed-Function Counter 6 (R/W)		
31:0	RELOAD_VALUE Contains the reload value to be loaded into the associated counter by Auto Counter Reload. Will be 1-extended to 48 bits.	
63:32	Reserved.	
Register Address: 1B01H, 6913		IA32_UARCH_MISC_CTL
IA32_UARCH_MISC_CTL (R/W)		If IA32_ARCH_CAPABILITIES[12] = 1
0	Data Operand Independent Timing Mode (DOITM).	If IA32_ARCH_CAPABILITIES[12] = 1
63:1	Reserved.	
Register Address: 4000_0000H–4000_00FFH		Reserved MSR Address Space
All existing and future processors will not implement MSRs in this range.		
Register Address: C000_0080H		IA32_EFER
Extended Feature Enables		If (CPUID.80000001H:EDX[20] CPUID.80000001H:EDX[29])
0	SYSCALL Enable: IA32_EFER.SCE (R/W) Enables SYSCALL/SYSRET instructions in 64-bit mode.	
7:1	Reserved.	
8	IA-32e Mode Enable: IA32_EFER.LME (R/W) Enables IA-32e mode operation.	
9	Reserved.	
10	IA-32e Mode Active: IA32_EFER.LMA (R) Indicates IA-32e mode is active when set.	
11	Execute Disable Bit Enable: IA32_EFER.NXE (R/W)	
63:12	Reserved.	
Register Address: C000_0081H		IA32_STAR
System Call Target Address (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0082H		IA32_LSTAR
IA-32e Mode System Call Target Address (R/W) Target RIP for the called procedure when SYSCALL is executed in 64-bit mode.		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0083H		IA32_CSTAR

Table 2-2. IA-32 Architectural MSRs (Contd.)

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	MSR/Bit Description	Comment
IA-32e Mode System Call Target Address (R/W) Not used, as the SYSCALL instruction is not recognized in compatibility mode.		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0084H		IA32_FMASK
System Call Flag Mask (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0100H		IA32_FS_BASE
Map of BASE Address of FS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0101H		IA32_GS_BASE
Map of BASE Address of GS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0102H		IA32_KERNEL_GS_BASE
Swap Target of BASE Address of GS (R/W)		If CPUID.80000001H:EDX[29] = 1
Register Address: C000_0103H		IA32_TSC_AUX
Auxiliary TSC (R/W)		If CPUID.80000001H:EDX[27] = 1 or CPUID.07H.00H:ECX[22] = 1
31:0	AUX: Auxiliary signature of TSC.	
63:32	Reserved.	

NOTES:

1. Some older processors may have supported this MSR as model-specific and do not enumerate it with CPUID.
2. In processors based on Intel NetBurst® microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.
3. The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MCI_STATUS. See Section 17.3.2.3 and Section 17.3.2.4 for more information.
4. Further details on Key Locker and usage of this MSR can be found here:
<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 2-3 lists model-specific registers (MSRs) for the Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 2-3. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_0FH, see Table 2-1.

MSRs listed in Table 2-2 and Table 2-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_17H.

The column “Shared/Unique” applies to multi-core processors based on Intel Core microarchitecture. “Unique” means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. “Shared” means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	IA32_P5_MC_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 2.23, "MSRs in Pentium Processors."		Unique
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, "Time-Stamp Counter," and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
49:13	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 12.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
3	MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
5	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
6	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processors implement R/W.	
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
11	Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
13	Reserved.	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes.	
15	Reserved.	
17:16	APIC Cluster ID (R/O)	
18	N/2 Non-Integer Bus Ratio (R/O) 0 = Integer ratio; 1 = Non-integer ratio.	
19	Reserved.	
21: 20	Symmetric Arbitration ID (R/O)	
26:22	Integer Bus Frequency Ratio (R/O)	
Register Address: 3AH, 58	MSR_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Unique
3	SMRR Enable (R/WL) When this bit is set and the lock bit is set, this makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.	Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of four pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 19.5.		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of four pairs of last branch record registers on the last branch record stack. This To_IP part of the stack contains pointers to the destination instruction.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: A0H, 160	MSR_SMRR_PHYSBASE	
System Management Mode Base Address register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
11:0	Reserved.	
31:12	PhysBase: SMRR physical Base Address.	
63:32	Reserved.	
Register Address: A1H, 161	MSR_SMRR_PHYSMASK	
System Management Mode Physical Address Mask register (WO in SMM) Model-specific implementation of SMRR-like interface, read visible and write only in SMM.		Unique
10:0	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
11	Valid: Physical address base and range mask are valid.	
31:12	PhysMask: SMRR physical address range mask.	
63:32	Reserved.	
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Core microarchitecture.		Shared
2:0	▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333)	
	133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.	
63:3	Reserved.	
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Enhanced Intel Core microarchitecture.		Shared

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
2:0	▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Unique
11	SMRR Capability Using MSR 0A0H and 0A1H (R)	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Current performance status. See Section 16.1.1, "Software Interface For Initiating Performance State Transitions."		Shared
15:0	Current Performance State Value	
30:16	Reserved.	
31	XE Operation (R/O). If set, XE operation is enabled. Default is cleared.	
39:32	Reserved.	
44:40	Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.	
45	Reserved.	
46	Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.	
63:47	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
15:0	Reserved.	
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.	
63:16	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	Hardware Prefetcher Disable (R/W) When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.01H:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.	Shared
20	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: - Enhanced Intel SpeedStep Technology Select Lock (this bit). - Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Shared
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
34	XD Bit Disable (R/W) When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H:EDX[20] = 0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location if XD bit is not supported. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	Unique
36:35	Reserved.	
37	DCU Prefetcher Disable (R/W) When set to 1, the DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.	Unique
38	IDA Disable (R/W) When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of IDA is enabled. Note: The power-on default value is used by BIOS to detect hardware support of IDA. If the power-on default value is 1, IDA is available in the processor. If the power-on default value is 0, IDA is not available.	Shared
39	IP Prefetcher Disable (R/W) When set to 1, the IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.	Unique
63:40	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Unique
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Unique
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Unique
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Unique
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Unique
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Unique
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Unique
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Unique
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Unique
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Unique
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Unique
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Unique
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Unique
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Unique
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Unique
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Unique
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Unique
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Unique
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Unique
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Unique
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Unique
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Unique
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Unique
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Unique
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Unique
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Unique
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Unique
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 19.4.1, "IA32_DEBUGCTL MSR."		Unique
Register Address: 345H, 837	MSR_PERF_CAPABILITIES	
R/O. This applies to processors that do not support architectural PerfMon version 2.		Unique
5:0	LBR Format. See Table 2-2.	
6	PEBS Record Format.	
7	PEBSSaveArchRegs. See Table 2-2.	
63:8	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	MSR_PERF_GLOBAL_CTRL	
See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 21.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0. (R/W)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 402H, 1026	IA32_MCO_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 40CH, 1036	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Unique
Register Address: 40DH, 1037	IA32_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."		Unique
Register Address: 40EH, 1038	IA32_MC4_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		
Register Address: 411H, 1041	IA32_MC3_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."		
Register Address: 412H, 1042	IA32_MC3_ADDR	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 413H, 1043	IA32_MC3_MISC	
Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 414H, 1044	IA32_MC5_CTL	
Machine Check Error Reporting Register: Controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	IA32_MC5_STATUS	
Machine Check Error Reporting Register: Contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.		Unique
Register Address: 416H, 1046	IA32_MC5_ADDR	
Machine Check Error Reporting Register: Contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 417H, 1047	IA32_MC5_MISC	
Machine Check Error Reporting Register: Contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 419H, 1045	IA32_MC6_STATUS	
Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 25.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 21.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: 107CCH, 67532	MSR_EMON_L3_CTR_CTL0	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107CDH, 67533	MSR_EMON_L3_CTR_CTL1	
GBUSQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107CEH, 67534	MSR_EMON_L3_CTR_CTL2	
GSPNQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107CFH, 67535	MSR_EMON_L3_CTR_CTL3	
GSPNQ Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107D0H, 67536	MSR_EMON_L3_CTR_CTL4	

Table 2-3. MSRs in Processors Based on Intel® Core™ Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107D1H, 67537	MSR_EMON_L3_CTR_CTL5	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107D2H, 67538	MSR_EMON_L3_CTR_CTL6	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107D3H, 67539	MSR_EMON_L3_CTR_CTL7	
FSB Event Control/Counter Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: 107D8H, 67544	MSR_EMON_L3_GL_CTL	
L3/FSB Common Control Register (R/W) Applies to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 19.2.2.		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

2.3 MSRS IN THE 45 NM AND 32 NM INTEL ATOM® PROCESSOR FAMILY

Table 2-4 lists model-specific registers (MSRs) for 45 nm and 32 nm Intel Atom processors, architectural MSR addresses are also included in Table 2-4. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1CH, 06_26H, 06_27H, 06_35H, or 06_36H; see Table 2-1.

The column “Shared/Unique” applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. “Unique” means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. “Shared” means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, “Time-Stamp Counter,” and see Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Shared
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Shared
7:0	Reserved.	
12:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
63:13	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 12.4.4, “Local APIC Status and Location,” and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
3	AERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
4	BERR# Enable for initiator bus requests (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
5	Reserved.	
6	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Always 0.	
8	Reserved.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	AERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0.	
11	Reserved.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0.	
13	Reserved.	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes.	
15	Reserved.	
17:16	APIC Cluster ID (R/O) Always 00B.	
19: 18	Reserved.	
21: 20	Symmetric Arbitration ID (R/O) Always 00B.	
26:22	Integer Bus Frequency Ratio (R/O)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 19.5.		Unique

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Unique
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.		Unique
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Unique
Register Address: C1H, 193	IA32_PMC0	
Performance counter register See Table 2-2.		Unique
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Unique
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Intel Atom microarchitecture.		Shared
2:0	111B: 083 MHz (FSB 333) 101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.	
63:3	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Unique
Register Address: FEH, 254	IA32_MTRRCAP	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Memory Type Range Register (R) See Table 2-2.		Shared
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Shared
0	L2 Hardware Enabled (R/O) 1 = Indicates the L2 is hardware-enabled. 0 = Indicates the L2 is hardware-disabled.	
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set, the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Shared
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Shared
15:0	Current Performance State Value.	
39:16	Reserved.	
44:40	Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.	
63:45	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Shared
15:0	Reserved.	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle). 1 = Thermal Monitor 2 (thermally-initiated frequency transitions). If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.	
63:17	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Unique
0	Fast-Strings Enable See Table 2-2.	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
8	Reserved.	
9	Reserved.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Shared
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is cleared (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.01H:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.	Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
20	Enhanced Intel SpeedStep Technology Select Lock (R/WO) When set, this bit causes the following bits to become read-only: - Enhanced Intel SpeedStep Technology Select Lock (this bit). - Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.	Shared
21	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Unique
23	xTPR Message Disable (R/W) See Table 2-2.	Shared
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Unique
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Shared

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Shared
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Shared
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Unique
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Unique
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Unique
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Unique
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 19.4.1, "IA32_DEBUGCTL MSR."		Shared
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Unique
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Unique
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2. See Section 21.6.2.4, "Processor Event Based Sampling (PEBS)."		Unique
0	Enable PEBS on IA32_PMC0 (R/W)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Shared
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Shared
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Shared
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Shared
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Shared
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Shared
Register Address: 412H, 1042	IA32_MC4_ADDR	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Shared
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	

Table 2-4. MSRs in the 45 nm and 32 nm Intel Atom® Processor Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Unique
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2. See Section 21.6.3.4, "Debug Store (DS) Mechanism."		Unique
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Unique
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Unique
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Unique
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Unique
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Unique

Table 2-5 lists model-specific registers (MSRs) that are specific to Intel Atom® processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_27H.

Table 2-5. MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3F8H, 1016	MSR_PKG_C2_RESIDENCY	
Package C2 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-5. MSRs Supported by Intel Atom® Processors (Contd.)with a CPUID Signature DisplayFamily_DisplayModel Value of 06_27H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:0	Package C2 Residency Counter (R/O) Time that this package is in processor-specific C2 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3F9H, 1017	MSR_PKG_C4_RESIDENCY	
Package C4 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C4 Residency Counter. (R/O) Time that this package is in processor-specific C4 states since last reset. Counts at 1 Mhz frequency.	Package
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Package C6 Residency Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter. (R/O) Time that this package is in processor-specific C6 states since last reset. Counts at 1 Mhz frequency.	Package

2.4 MSRS IN INTEL PROCESSORS BASED ON SILVERMONT MICROARCHITECTURE

Table 2-6 lists model-specific registers (MSRs) common to Intel processors based on the Silvermont microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_37H, 06_4AH, 06_4DH, 06_5AH, or 06_5DH; see Table 2-1. The MSRs listed in Table 2-6 are also common to processors based on the Airmont microarchitecture and newer microarchitectures for next generation Intel Atom processors.

Table 2-7 lists MSRs common to processors based on the Silvermont and Airmont microarchitectures, but not newer microarchitectures.

Table 2-8, Table 2-9, and Table 2-10 lists MSRs that are model-specific across processors based on the Silvermont microarchitecture.

In the Silvermont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Silvermont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Core
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, "Time-Stamp Counter," and Table 2-2.		Core
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 12.4.4, "Local APIC Status and Location," and Table 2-2.		Core
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Writes ignored.		Module
63:0	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Core
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Core
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Core
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Core
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Module
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWait Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWait instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 100b - C4 is the max C-State to include 110b - C6 is the max C-State to include 111b - C7 is the max C-State to include	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Core
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction sets availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note: AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Core
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Core
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Core
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Core
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Core
7:0	Event Select	
15:8	UMask	
16	USR	
17	OS	
18	Edge	
19	PC	
20	INT	
21	Reserved.	
22	EN	
23	INV	
31:24	CMASK	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Module
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Core
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Core
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R) The default thermal throttling or PROCHOT# activation temperature in degrees C. The effective temperature for thermal throttling or PROCHOT# activation is "Temperature Target" + "Target Offset".	
29:24	Target Offset (R/W) Specifies an offset in degrees C to adjust the throttling and PROCHOT# activation temperature from the default target specified in TEMPERATURE_TARGET (bits 23:16).	
63:30	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Module
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Core
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Core
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Core
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Core
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Core
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Core
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Core
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 19.4.1, "IA32_DEBUGCTL MSR."		Core
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Core
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Counts at the TSC Frequency.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Module
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Module
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2. See Appendix A.1, "Basic VMX Information."		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2. See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2. See Appendix A.4, "VM-Exit Controls."		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2. See Appendix A.5, "VM-Entry Controls."		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2. See Appendix A.6, "Miscellaneous Data."		Core
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.7, "VMX-Fixed Bits in CR0."		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2. See Appendix A.8, "VMX-Fixed Bits in CR4."		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2. See Appendix A.9, "VMCS Enumeration."		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTL2	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTL2	
Capability Reporting Register of Primary Processor-based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTL2	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTL2	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Core
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 21.6.3.4, "Debug Store (DS) Mechanism."		Core
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C1 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C1 states. Counts at the TSC frequency.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Core
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Core
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Core
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Core
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Core
Register Address: C000_0103H	IA32_TSC_AUX	

Table 2-6. MSRs Common to Intel Atom® Processors (Silvermont and Newer Microarchitectures) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
AUXILIARY TSC Signature (R/W) See Table 2-2.		Core

Table 2-7 lists model-specific registers (MSRs) that are common to Intel Atom® processors based on the Silvermont and Airmont microarchitectures but not newer microarchitectures.

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
7:0	Reserved.	
13:8	Maximum Qualified Ratio (R) The maximum allowed bus ratio.	
49:13	Reserved.	
52:50	See Table 2-2.	
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation (R/WL)	
Register Address: 40H, 64	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 19.5 and record format in Section 19.4.8.1.		Core
Register Address: 41H, 65	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 42H, 66	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 43H, 67	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 44H, 68	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 45H, 69	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 46H, 70	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 47H, 71	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 60H, 96	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of eight pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the destination instruction.		Core
Register Address: 61H, 97	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 62H, 98	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 63H, 99	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 64H, 100	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 65H, 101	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 66H, 102	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 67H, 103	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information: Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * Scalable Bus Frequency.	Package
63:16	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Module
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 100b: C4 110b: C6 111b: C7 (Silvermont only)	
9:3	Reserved.	
10	I/O MWait Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWait instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 11EH, 281	MSR_BBL_CR_CTL3	
Control Register 3 Used to configure the L2 Cache.		Module
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.	
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 0.	Module
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Module
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Module
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be cleared (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Module
63:39	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 19.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2. See Section 21.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS for precise event on IA32_PMC0 (R/W)	
Register Address: 3FAH, 1018	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Counts at the TSC Frequency.	

Table 2-7. MSRs Common to the Silvermont and Airmont Microarchitectures (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	

2.4.1 MSRs with Model-Specific Behavior in the Silvermont Microarchitecture

Table 2-8 lists MSRs that are specific to the Intel Atom[®] processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H) and Intel Atom processors (CPUID Signature DisplayFamily_DisplayModel value of 06_4AH, 06_5AH, or 06_5DH).

Table 2-8. Specific MSRs Supported by Intel Atom[®] Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Silvermont microarchitecture.		Module
2:0	100B: 080.0 MHz 000B: 083.3 MHz 001B: 100.0 MHz 010B: 133.3 MHz 011B: 116.7 MHz	
63:3	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 16.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in milliWatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.	
7:4	Reserved.	
12:8	Energy Status Units Energy related information (in microJoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microJoules increment.	
15:13	Reserved.	
19:16	Time Unit The value is 0000b, indicating time unit is in one second.	
63:20	Reserved.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W)		Package

Table 2-8. Specific MSRs Supported by Intel Atom® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H, 06_4AH, 06_5AH, or 06_5DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	Package Power Limit #1 (R/W) See Section 16.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 16.10.3, "Package RAPL Domain."	
16	Package Clamping Limitation #1 (R/W) See Section 16.10.3, "Package RAPL Domain."	
23:17	Time Window for Power Limit #1 (R/W) In unit of second. If 0 is specified in bits [23:17], defaults to 1 second window.	
63:24	Reserved.	
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 16.10.3, "Package RAPL Domain," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 16.10.4, "PP0/PP1 RAPL Domains," and MSR_RAPL_POWER_UNIT in Table 2-8.		Package

Table 2-9 lists model-specific registers (MSRs) that are specific to the Intel Atom® processor E3000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_37H).

Table 2-9. Specific MSRs Supported by the Intel Atom® Processor E3000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_37H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 668H, 1640	MSR_CC6_DEMOTION_POLICY_CONFIG	
Core C6 Demotion Policy Config MSR		Package
63:0	Controls per-core C6 demotion policy. Writing a value of 0 disables core level HW demotion policy.	
Register Address: 669H, 1641	MSR_MC6_DEMOTION_POLICY_CONFIG	
Module C6 Demotion Policy Config MSR		Package
63:0	Controls module (i.e., two cores sharing the second-level cache) C6 demotion policy. Writing a value of 0 disables module level HW demotion policy.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	

Table 2-10 lists model-specific registers (MSRs) that are specific to Intel Atom® processor C2000 Series (CPUID Signature DisplayFamily_DisplayModel value of 06_4DH).

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode (R/W)		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 16.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in milliWatts) is based on the multiplier, 2^{PU} ; where PU is an unsigned integer represented by bits 3:0. Default value is 0101b, indicating power unit is in 32 milliWatts increment.	
7:4	Reserved.	

Table 2-10. Specific MSRs Supported by Intel Atom® Processor C2000 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4DH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
12:8	Energy Status Units. Energy related information (in microjoules) is based on the multiplier, 2^{ESU} ; where ESU is an unsigned integer represented by bits 12:8. Default value is 00101b, indicating energy unit is in 32 microjoules increment.	
15:13	Reserved.	
19:16	Time Unit The value is 0000b, indicating time unit is in one second.	
63:20	Reserved.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 66EH, 1646	MSR_PKG_POWER_INFO	
PKG RAPL Parameter (R/O)		Package
14:0	Thermal Spec Power (R/O) The unsigned integer value is the equivalent of the thermal specification power of the package domain. The unit of this field is specified by the "Power Units" field of MSR_RAPL_POWER_UNIT.	
63:15	Reserved.	

2.4.2 MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Intel Atom processor X7-Z8000 and X5-Z8000 series are based on the Airmont microarchitecture. These processors support MSRs listed in Table 2-6, Table 2-7, Table 2-8, and Table 2-11. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_4CH; see Table 2-1.

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CDH, 205	MSR_FSB_FREQ	
Scaleable Bus Speed (R/O) This field indicates the intended scalable bus clock speed for processors based on Airmont microarchitecture.		Module
3:0	▪ 0000B: 083.3 MHz ▪ 0001B: 100.0 MHz ▪ 0010B: 133.3 MHz ▪ 0011B: 116.7 MHz ▪ 0100B: 080.0 MHz ▪ 0101B: 093.3 MHz ▪ 0110B: 090.0 MHz ▪ 0111B: 088.9 MHz ▪ 1000B: 087.5 MHz	
63:5	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Module
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: No limit 001b: C1 010b: C2 110b: C6 111b: C7	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/W0) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Module
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - Deep Power Down Technology is the max C-State. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W)		Package

Table 2-11. MSRs in Intel Atom® Processors Based on Airmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	PP0 Power Limit #1 (R/W) See Section 16.10.4, “PP0/PP1 RAPL Domains,” and MSR_RAPL_POWER_UNIT in Table 2-8.	
15	Enable Power Limit #1 (R/W) See Section 16.10.4, “PP0/PP1 RAPL Domains.”	
16	Reserved.	
23:17	Time Window for Power Limit #1 (R/W) Specifies the time duration over which the average power must remain below PP0_POWER_LIMIT #1(14:0). Supported Encodings: 0x0: 1 second time duration. 0x1: 5 second time duration (Default). 0x2: 10 second time duration. 0x3: 15 second time duration. 0x4: 20 second time duration. 0x5: 25 second time duration. 0x6: 30 second time duration. 0x7: 35 second time duration. 0x8: 40 second time duration. 0x9: 45 second time duration. 0xA: 50 second time duration. 0xB-0x7F - reserved.	
63:24	Reserved.	

2.5 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT MICROARCHITECTURE

Intel Atom processors based on the Goldmont microarchitecture support MSRs listed in Table 2-6 and Table 2-12. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH; see Table 2-1.

In the Goldmont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Module
49:0	Reserved.	
52:50	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:33	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Enable VMX inside SMX operation (R/WL)	
2	Enable VMX outside SMX operation (R/WL)	
14:8	SENTER local functions enables (R/WL)	
15	SENTER global functions enable (R/WL)	
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Core TSC ADJUST (R/W) See Table 2-2.		Core
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Core
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the maximum frequency that does not require turbo. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
30	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.	Package
39:31	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: No limit 0001b: C1 0010b: C3 0011b: C6 0100b: C7 0101b: C7S 0110b: C8 0111b: C9 1000b: C10	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability enhancement. Accessible only while in SMM.		Core
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.	
59	Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.	
63:60	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Core
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Core
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Core
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Package
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Core
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Core
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Core
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Core
21:19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2.	Core
23	xTPR Message Disable (R/W) See Table 2-2.	Package
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Core
37:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	Reserved.	
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
63:3	Reserved.	
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org.		Package
0	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.	
21:1	Reserved.	
22	Thermal Interrupt Coordination Enable (R/W) If set, then thermal interrupt on one core is routed to all cores.	
63:23	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode by Core Groups (R/W) Specifies Maximum Ratio Limit for each Core Group. Max ratio for groups with more cores must decrease monotonically. For groups with less than 4 cores, the max ratio must be 32 or less. For groups with 4-5 cores, the max ratio must be 22 or less. For groups with more than 5 cores, the max ratio must be 16 or less.		Package
7:0	Maximum Ratio Limit for Active Cores in Group 0 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 0 threshold.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
15:8	Maximum Ratio Limit for Active Cores in Group 1 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 1 threshold, and greater than the Group 0 threshold.	Package
23:16	Maximum Ratio Limit for Active Cores in Group 2 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 2 threshold, and greater than the Group 1 threshold.	Package
31:24	Maximum Ratio Limit for Active Cores in Group 3 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 3 threshold, and greater than the Group 2 threshold.	Package
39:32	Maximum Ratio Limit for Active Cores in Group 4 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 4 threshold, and greater than the Group 3 threshold.	Package
47:40	Maximum Ratio Limit for Active Cores in Group 5 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 5 threshold, and greater than the Group 4 threshold.	Package
55:48	Maximum Ratio Limit for Active Cores in Group 6 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 6 threshold, and greater than the Group 5 threshold.	Package
63:56	Maximum Ratio Limit for Active Cores in Group 7 Maximum turbo ratio limit when the number of active cores is less than or equal to the Group 7 threshold, and greater than the Group 6 threshold.	Package
Register Address: 1AEH, 430	MSR_TURBO_GROUP_CORECNT	
Group Size of Active Cores for Turbo Mode Operation (R/W) Writes of 0 threshold is ignored.		Package
7:0	Group 0 Core Count Threshold Maximum number of active cores to operate under the Group 0 Max Turbo Ratio limit.	Package
15:8	Group 1 Core Count Threshold Maximum number of active cores to operate under the Group 1 Max Turbo Ratio limit. Must be greater than the Group 0 Core Count.	Package
23:16	Group 2 Core Count Threshold Maximum number of active cores to operate under the Group 2 Max Turbo Ratio limit. Must be greater than the Group 1 Core Count.	Package
31:24	Group 3 Core Count Threshold Maximum number of active cores to operate under the Group 3 Max Turbo Ratio limit. Must be greater than the Group 2 Core Count.	Package
39:32	Group 4 Core Count Threshold Maximum number of active cores to operate under the Group 4 Max Turbo Ratio limit. Must be greater than the Group 3 Core Count.	Package
47:40	Group 5 Core Count Threshold Maximum number of active cores to operate under the Group 5 Max Turbo Ratio limit. Must be greater than the Group 4 Core Count.	Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55:48	Group 6 Core Count Threshold Maximum number of active cores to operate under the Group 6 Max Turbo Ratio limit. Must be greater than the Group 5 Core Count.	Package
63:56	Group 7 Core Count Threshold Maximum number of active cores to operate under the Group 7 Max Turbo Ratio limit. Must be greater than the Group 6 Core Count, and not less than the total number of processor cores in the package. E.g., specify 255.	Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 19.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:10	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
63:2	Reserved.	
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Core
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Core
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address:	IA32_MTRR_PHYSMASK9	
213H, 531	See Table 2-2.	Core
Register Address:	IA32_MC0_CTL2	
280H, 640	See Table 2-2.	Module
Register Address:	IA32_MC1_CTL2	
281H, 641	See Table 2-2.	Module
Register Address:	IA32_MC2_CTL2	
282H, 642	See Table 2-2.	Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Module
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 300H, 768	MSR_SGXOWNEREPOCH0	
Lower 64 Bit CR_SGXOWNEREPOCH (w) Writes do not update CR_SGXOWNEREPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNEREPOCH1	
Upper 64 Bit CR_SGXOWNEREPOCH (w) Writes do not update CR_SGXOWNEREPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 21.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Ovf_PMC0	
1	Ovf_PMC1	
2	Ovf_PMC2	
3	Ovf_PMC3	
31:4	Reserved.	
32	Ovf_FixedCtr0	
33	Ovf_FixedCtr1	
34	Ovf_FixedCtr2	
54:35	Reserved.	
55	Trace_ToPA_PMI	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
57:56	Reserved.	
58	LBR_Frz	
59	CTR_Frz	
60	ASCI	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 21.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Set 1 to clear Ovf_PMC0.	
1	Set 1 to clear Ovf_PMC1.	
2	Set 1 to clear Ovf_PMC2.	
3	Set 1 to clear Ovf_PMC3.	
31:4	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	
33	Set 1 to clear Ovf_FixedCtr1.	
34	Set 1 to clear Ovf_FixedCtr2.	
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	
59	Set 1 to clear CTR_Frz.	
60	Set 1 to clear ASCI.	
61	Set 1 to clear Ovf_Uncore.	
62	Set 1 to clear Ovf_BufDSSAVE.	
63	Set 1 to clear CondChgd.	
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 21.2.4, "Architectural Performance Monitoring Version 4."		Core
0	Set 1 to cause Ovf_PMC0 = 1.	
1	Set 1 to cause Ovf_PMC1 = 1.	
2	Set 1 to cause Ovf_PMC2 = 1.	
3	Set 1 to cause Ovf_PMC3 = 1.	
31:4	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	
33	Set 1 to cause Ovf_FixedCtr1 = 1.	
34	Set 1 to cause Ovf_FixedCtr2 = 1.	
54:35	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55	Set 1 to cause Trace_ToPA_PMI = 1.	
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	
59	Set 1 to cause CTR_Frz = 1.	
60	Set 1 to cause ASCI = 1.	
61	Set 1 to cause Ovf_Uncore.	
62	Set 1 to cause Ovf_BufDSSAVE.	
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2 and Section 21.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0. (R/W)	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Module
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 18.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Core
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Core
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
0	Lock (SMM-RW0) When set to '1' locks this register from further changes.	
1	Reserved.	
2	SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.		Package
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a processor core of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is OFFFH. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Core
0	Lock See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
15:1	Reserved.	
23:16	SGX_SVN_SINIT See Section 41.11.3, "Interactions with Authenticated Code Modules (ACMs)."	
63:24	Reserved.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Core
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Core
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	BranchEn	
17:14	MTCFreq	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Core
0	FilterEn Writes ignored.	
1	ContextEn Writes ignored.	
2	TriggerEn Writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
31:6	Reserved, must be zero.	
48:32	PacketByteCnt	
63:49	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Core
4:0	Reserved	
63:5	CR3[63:5] value to match.	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Core
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Core
63:0	See Table 2-2.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 16.10.1, "RAPL Interfaces."		Package
3:0	Power Units Power related information (in Watts) is in unit of $1W/2^{PU}$; where PU is an unsigned integer represented by bits 3:0. Default value is 1000b, indicating power unit is in 3.9 milliWatts increment.	
7:4	Reserved.	
12:8	Energy Status Units Energy related information (in Joules) is in unit of $1Joule/2^{ESU}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 01110b, indicating energy unit is in 61 microJoules.	
15:13	Reserved.	
19:16	Time Unit Time related information (in seconds) is in unit of $1S/2^{TU}$; where TU is an unsigned integer represented by bits 19:16. Default value is 1010b, indicating power unit is in 0.977 millisecond.	
63:20	Reserved.	
Register Address: 60AH, 1546	MSR_PKG_C3_IRT_L	
Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKG_C6_C7S_IRT_L1	
Package C6/C7S Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7S state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7S state.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKGC_IRTL2	
Package C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W)		Package

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:0	Thermal Spec Power (R/W) See Section 16.10.3, "Package RAPL Domain."	
15	Reserved.	
30:16	Minimum Power (R/W) See Section 16.10.3, "Package RAPL Domain."	
31	Reserved.	
46:32	Maximum Power (R/W) See Section 16.10.3, "Package RAPL Domain."	
47	Reserved.	
54:48	Maximum Time Window (R/W) Specified by $2^Y * (1.0 + Z/4.0) * \text{Time\_Unit}$, where "Y" is the unsigned integer value represented by bits 52:48, "Z" is an unsigned integer represented by bits 54:53. "Time_Unit" is specified by the "Time Units" field of MSR_RAPL_POWER_UNIT.	
63:55	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C10 Residency Counter (R/O) Value since last reset that the entire SOC is in an S0i3 state. Count at the same frequency as the TSC.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
3	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
8:4	Reserved.	
9	Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	
10	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
11	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
12	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
14	Maximum Efficiency Frequency Status (R0) When set, frequency is reduced below the maximum efficiency frequency.	
15	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24:20	Reserved.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Maximum Efficiency Frequency Log When set, indicates that the Maximum Efficiency Frequency Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:31	Reserved.	
Register Address: 680H, 1664	MSR_LASTBRANCH_O_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 0 From IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 19.6 and record format in Section 19.4.8.1.		Core
0:47	From Linear Address (R/W)	
62:48	Signed extension of bits 47:0.	
63	Mispred	
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	
Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of 32 pairs of last branch record registers on the last branch record stack. The To_IP part of the stack contains pointers to the Destination instruction and elapsed cycles from last LBR update. See Section 19.6.		Core
0:47	Target Linear Address (R/W)	
63:48	Elapsed cycles from last update to the LBR.	
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D0H, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID register (R/O)		Core
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version register (R/O)		Core
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority register (R/W)		Core
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority register (R/O)		Core
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI register (W/O)		Core
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination register (R/O)		Core
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector register (R/W)		Core
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service register bits [31:0] (R/O)		Core
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service register bits [63:32] (R/O)		Core
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service register bits [95:64] (R/O)		Core
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service register bits [127:96] (R/O)		Core
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service register bits [159:128] (R/O)		Core
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service register bits [191:160] (R/O)		Core
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service register bits [223:192] (R/O)		Core
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service register bits [255:224] (R/O)		Core
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode register bits [31:0] (R/O)		Core
Register Address: 819H, 2073	IA32_X2APIC_TMR1	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Trigger Mode register bits [63:32] (R/O)		Core
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode register bits [95:64] (R/O)		Core
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode register bits [127:96] (R/O)		Core
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode register bits [159:128] (R/O)		Core
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode register bits [191:160] (R/O)		Core
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode register bits [223:192] (R/O)		Core
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode register bits [255:224] (R/O)		Core
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request register bits [31:0] (R/O)		Core
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request register bits [63:32] (R/O)		Core
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request register bits [95:64] (R/O)		Core
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request register bits [127:96] (R/O)		Core
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request register bits [159:128] (R/O)		Core
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request register bits [191:160] (R/O)		Core
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request register bits [223:192] (R/O)		Core
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request register bits [255:224] (R/O)		Core
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status register (R/W)		Core
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt register (R/W)		Core
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command register (R/W)		Core
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt register (R/W)		Core

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt register (R/W)		Core
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor register (R/W)		Core
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 register (R/W)		Core
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 register (R/W)		Core
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error register (R/W)		Core
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count register (R/W)		Core
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count register (R/O)		Core
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration register (R/W)		Core
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI register (W/O)		Core
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Core
31:0	Reserved.	
33:32	CLOS (R/W)	
63: 34	Reserved.	
Register Address: D10H, 3344	IA32_L2_QOS_MASK_0	
L2 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D11H, 3345	IA32_L2_QOS_MASK_1	
L2 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	
Register Address: D12H, 3346	IA32_L2_QOS_MASK_2	
L2 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2.		Module
0:7	CBM: Bit vector of available L2 ways for CLOS 0 enforcement.	
63:8	Reserved.	

Table 2-12. MSRs in Intel Atom® Processors Based on Goldmont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D13H, 3347	IA32_L2_QOS_MASK_3	
L2 Class Of Service Mask - CLOS 3 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L2 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Core
Register Address: DA0H, 3488	IA32_XSS	
See Table 2-2.		Core
See Table 2-6, and Table 2-12 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_5CH.		

2.6 MSRS IN INTEL ATOM® PROCESSORS BASED ON GOLDMONT PLUS MICROARCHITECTURE

Intel Atom processors based on the Goldmont Plus microarchitecture support MSRs listed in Table 2-6, Table 2-12, and Table 2-13. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH; see Table 2-1. For an MSR listed in Table 2-13 that also appears in the model-specific tables of prior generations, Table 2-13 supersedes prior generation tables.

In the Goldmont Plus microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Goldmont Plus microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Core
0	Lock (R/WL)	
1	Enable VMX inside SMX operation (R/WL)	
2	Enable VMX outside SMX operation (R/WL)	
14:8	SENTER local functions enables (R/WL)	
15	SENTER global functions enable (R/WL)	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Valid if CPUID.07H.00H:ECX[30] = 1.	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
18	SGX global functions enable (R/WL)	
63:19	Reserved.	
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Core
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Core
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Core
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	
See Table 2-2.		Core
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 21.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
0	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC0.	
1	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC1.	
2	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC2.	
3	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMC3.	
31:4	Reserved.	
32	Enable PEBS trigger and recording for IA32_FIXED_CTR0.	
33	Enable PEBS trigger and recording for IA32_FIXED_CTR1.	
34	Enable PEBS trigger and recording for IA32_FIXED_CTR2.	
63:35	Reserved.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Core
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
4	PwrEvtEn	
5	FUPonPTW	
6	FabricEn	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
11	DisRETC	
12	PTWEn	
13	BranchEn	
17:14	MTCFreq	
18	Reserved, must be zero.	
22:19	CycThresh	
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 19.7, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture.”		Core
Register Address: 681H–69FH, 1665–1695	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP; <i>i</i> = 1-31.		Core
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. The To_IP part of the stack contains pointers to the Destination instruction. See also: ▪ Section 19.7, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Goldmont Plus Microarchitecture.”		Core
Register Address: 6C1H–6DFH, 1729–1759	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP; <i>i</i> = 1-31.		Core
Register Address: DC0H, 3520	MSR_LASTBRANCH_INFO_0	
Last Branch Record 0 Additional Information (R/W) One of the three MSRs that make up the first entry of the 32-entry LBR stack. This part of the stack contains flag and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 19.9.1, “LBR Stack.”		Core
Register Address: DC1H, 3521	MSR_LASTBRANCH_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DC2H, 3522	MSR_LASTBRANCH_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC3H, 3523	MSR_LASTBRANCH_INFO_3	
Last Branch Record 3 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC4H, 3524	MSR_LASTBRANCH_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC5H, 3525	MSR_LASTBRANCH_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC6H, 3526	MSR_LASTBRANCH_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC7H, 3527	MSR_LASTBRANCH_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC8H, 3528	MSR_LASTBRANCH_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DC9H, 3529	MSR_LASTBRANCH_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCAH, 3530	MSR_LASTBRANCH_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCBH, 3531	MSR_LASTBRANCH_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCCH, 3532	MSR_LASTBRANCH_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCDH, 3533	MSR_LASTBRANCH_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DCEH, 3534	MSR_LASTBRANCH_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DCFH, 3535	MSR_LASTBRANCH_INFO_15	
Last Branch Record 15 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD0H, 3536	MSR_LASTBRANCH_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD1H, 3537	MSR_LASTBRANCH_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD2H, 3538	MSR_LASTBRANCH_INFO_18	
Last Branch Record 18 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD3H, 3539	MSR_LASTBRANCH_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD4H, 3520	MSR_LASTBRANCH_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD5H, 3521	MSR_LASTBRANCH_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD6H, 3522	MSR_LASTBRANCH_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD7H, 3523	MSR_LASTBRANCH_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD8H, 3524	MSR_LASTBRANCH_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DD9H, 3525	MSR_LASTBRANCH_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDAH, 3526	MSR_LASTBRANCH_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDBH, 3527	MSR_LASTBRANCH_INFO_27	
Last Branch Record 27 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core

Table 2-13. MSRs in Intel Atom® Processors Based on Goldmont Plus Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DDCH, 3528	MSR_LASTBRANCH_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDDH, 3529	MSR_LASTBRANCH_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDEH, 3530	MSR_LASTBRANCH_INFO_30	
Last Branch Record 30 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
Register Address: DDFH, 3531	MSR_LASTBRANCH_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LASTBRANCH_INFO_0.		Core
See Table 2-6, Table 2-12, and Table 2-13 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7AH.		

2.7 MSRS IN INTEL ATOM® PROCESSORS BASED ON TREMONT MICROARCHITECTURE

Processors based on the Tremont microarchitecture support MSRs listed in Table 2-6, Table 2-12, Table 2-13, and Table 2-14. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_86H, 06_96H, or 06_9CH; see Table 2-1. For an MSR listed in Table 2-14 that also appears in the model-specific tables of prior generations, Table 2-14 supersedes prior generation tables.

In the Tremont microarchitecture, the scope column indicates the following: “Core” means each processor core has a separate MSR, or a bit field not shared with another processor core. “Module” means the MSR or the bit field is shared by a subset of the processor cores in the physical package. The number of processor cores in this subset is model specific and may differ between different processors. For all processors based on Tremont microarchitecture, the L2 cache is also shared between cores in a module and thus CPUID.04H enumeration can be used to figure out which processors are in the same module. “Package” means all processor cores in the physical package share the same MSR or bit interface.

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
28:0	Reserved.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 10.1.2.3, “Features to Disable Bus Locks.”	
30	Reserved.	
31	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
IA32 Core Capabilities Register If CPUID.07H.00H:EDX[30] = 1.		Core
4:0	Reserved.	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).	
63:6	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
(R/W) See Table 2-2. See Section 21.6.2.4, "Processor Event Based Sampling (PEBS)."		Core
$n:0$	Enable PEBS trigger and recording for the programmed event (precise or otherwise) on IA32_PMCx. The maximum value n can be determined from CPUID.0AH:EDX[15:8].	
31: $n+1$	Reserved.	
32: $m:32$	Enable PEBS trigger and recording for IA32_FIXED_CTRx. The maximum value m can be determined from CPUID.0AH:EDX[4:0].	
59:33: m	Reserved.	
60	Pend a PerfMon Interrupt (PMI) after each PEBS event.	
62:61	Specifies PEBS output destination. Encodings: 00B: DS Save Area. 01B: Intel PT trace output. Supported if IA32_PERF_CAPABILITIES.PEBS_OUTPUT_PT_AVAIL[16] and CPUID.07H.00H:EBX[25] are set. 10B: Reserved. 11B: Reserved.	
63	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C4H, 5313–5316	MSR_RELOAD_PMCx	

Table 2-14. MSRs in Intel Atom® Processors Based on Tremont Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Reload value for IA32_PMCx (R/W)		Core
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	
See Table 2-6, Table 2-12, Table 2-13, and Table 2-14 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_86H.		

2.8 MSRS IN PROCESSORS BASED ON NEHALEM MICROARCHITECTURE

Table 2-15 lists model-specific registers (MSRs) that are common for Nehalem microarchitecture. These include the Intel Core i7 and i5 processor family. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, 06_1FH, or 06_2EH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH are listed in Table 2-16. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at <http://biosbits.org>.

The column “Scope” represents the package/core/thread scope of individual bit field of an MSR. “Thread” means this bit field must be programmed on each logical processor independently. “Core” means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. “Package” means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, “MSRs in Pentium Processors.”		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, “MSRs in Pentium Processors.”		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, “Monitor/Mwait Address Range Determination,” and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, “Time-Stamp Counter,” and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 17H, 23	MSR_PLATFORM_ID	
Model Specific Platform ID (R)		Package
49:0	Reserved.	
52:50	See Table 2-2.	
63:53	Reserved.	
Register Address: 1BH, 27	IA32_APIC_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 12.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Running count of SMI events since last RESET.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Programmable TDC-TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDC and TDP Limits for Turbo mode are programmable. When set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 133.33MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-state support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
23:16	Reserved.	
24	Interrupt filtering enable (R/W) When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.	
25	C3 state auto demotion enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
26	C1 state auto demotion enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Core
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-state Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSELO	
See Table 2-2.		Thread
7:0	Event Select	
15:8	UMask	
16	USR	
17	OS	
18	Edge	
19	PC	
20	INT	
21	AnyThread	
22	EN	
23	INV	
31:24	CMASK	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Core
15:0	Current Performance State Value.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
0	Reserved.	
3:1	On demand Clock Modulation Duty Cycle (R/W)	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Thread
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2. Default value is 1.	Thread
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM. (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Thread
15:0	Reserved.	
23:16	Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
63:24	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .		
0	EIST Hardware Coordination Disable (R/W) When 0, enables hardware coordination of Enhanced Intel Speedstep Technology request from processor cores. When 1, disables hardware coordination of Enhanced Intel Speedstep Technology requests.	Package
1	Energy/Performance Bias Enable (R/W) This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.06H:ECX[3].	Thread
63:2	Reserved.	
Register Address: 1ACH, 428	MSR_TURBO_POWER_CURRENT_LIMIT	
See http://biosbits.org .		
14:0	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.	Package
15	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
30:16	TDC Limit (R/W) TDC limit in 1/8 Amp granularity.	Package
31	TDC Limit Override Enable (R/W) A value = 0 indicates override is not active; a value = 1 indicates override is active.	Package
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record Filtering Select Register (R/W) See Section 19.9.2, "Filtering of Last Branch Records."		Core
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
63:2	Reserved.	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MCO_CTL2	
See Table 2-2.		Package
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Package
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Core
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 19.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format	
7	PEBSSaveArchRegs See Table 2-2.	
11:8	PEBS_REC_FORMAT See Table 2-2.	
12	SMM_FREEZE See Table 2-2.	
63:13	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 38EH, 910	MSR_PERF_GLOBAL_STATUS	
Provides single-bit status used by software to query the overflow condition of each performance counter. (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
61	UNC_Ovf Uncore overflowed if 1.	
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities."		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2. See Section 21.6.2.2, "Global Counter Control Facilities." Allows software to clear counter overflow conditions on any combination of fixed-function PMCs (IA32_FIXED_CTRx) or general-purpose PMCs via a single WRMSR.		Thread
Register Address: 390H, 912	MSR_PERF_GLOBAL_OVF_CTRL	
(R/W)		Thread
61	CLR_UNC_Ovf Set 1 to clear UNC_Ovf.	
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 21.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
0	Enable PEBS on IA32_PMC0 (R/W)	
1	Enable PEBS on IA32_PMC1 (R/W)	
2	Enable PEBS on IA32_PMC2 (R/W)	
3	Enable PEBS on IA32_PMC3 (R/W)	
31:4	Reserved.	
32	Enable Load Latency on IA32_PMC0 (R/W)	
33	Enable Load Latency on IA32_PMC1 (R/W)	
34	Enable Load Latency on IA32_PMC2 (R/W)	
35	Enable Load Latency on IA32_PMC3 (R/W)	
63:36	Reserved.	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	
See Section 21.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:0	Package C6 Residency Counter (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package
Register Address: 403H, 1027	IA32_MCO_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Package

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Core

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 21.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. The From_IP part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ See Section 19.9.1 and record format in Section 19.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	

Table 2-15. MSRs in Processors Based on Nehalem Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 19.17.2, "IA32_TSC_AUX Register and RDTSCP Support."		Thread

2.8.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

The Intel Xeon Processor 5500 and 3400 series supports additional model-specific registers listed in Table 2-16. These MSRs also apply to the Intel Core i7 and i5 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_1AH, 06_1EH, or 06_1FH; see Table 2-1.

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Actual maximum turbo frequency is multiplied by 133.33MHz. (Not available in model 06_2EH.)		Package
7:0	Maximum Turbo Ratio Limit 1C (R/O) Maximum Turbo mode ratio limit with 1 core active.	
15:8	Maximum Turbo Ratio Limit 2C (R/O) Maximum Turbo mode ratio limit with 2 cores active.	
23:16	Maximum Turbo Ratio Limit 3C (R/O) Maximum Turbo mode ratio limit with 3 cores active.	
31:24	Maximum Turbo Ratio Limit 4C (R/O) Maximum Turbo mode ratio limit with 4 cores active.	
63:32	Reserved.	
Register Address: 301H, 769	MSR_GQ_SNOOP_MESF	
MSR_GQ_SNOOP_MESF		Package
0	From M to S (R/W)	
1	From E to S (R/W)	
2	From S to S (R/W)	
3	From F to S (R/W)	
4	From M to I (R/W)	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
5	From E to I (R/W)	
6	From S to I (R/W)	
7	From F to I (R/W)	
63:8	Reserved.	
Register Address: 391H, 913	MSR_UNCORE_PERF_GLOBAL_CTRL	
See Section 21.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 392H, 914	MSR_UNCORE_PERF_GLOBAL_STATUS	
See Section 21.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 393H, 915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	
See Section 21.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 394H, 916	MSR_UNCORE_FIXED_CTR0	
See Section 21.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 395H, 917	MSR_UNCORE_FIXED_CTR_CTRL	
See Section 21.3.1.2.1, "Uncore Performance Monitoring Management Facility."		Package
Register Address: 396H, 918	MSR_UNCORE_ADDR_OPCODE_MATCH	
See Section 21.3.1.2.3, "Uncore Address/Opcode Match MSR."		Package
Register Address: 3B0H, 960	MSR_UNCORE_PMC0	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B1H, 961	MSR_UNCORE_PMC1	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B2H, 962	MSR_UNCORE_PMC2	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B3H, 963	MSR_UNCORE_PMC3	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B4H, 964	MSR_UNCORE_PMC4	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B6H, 966	MSR_UNCORE_PMC6	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3B7H, 967	MSR_UNCORE_PMC7	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C0H, 944	MSR_UNCORE_PERFEVTSELO	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C1H, 945	MSR_UNCORE_PERFEVTSEL1	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C2H, 946	MSR_UNCORE_PERFEVTSEL2	

Table 2-16. Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C3H, 947	MSR_UNCORE_PERFEVTSEL3	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C4H, 948	MSR_UNCORE_PERFEVTSEL4	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C5H, 949	MSR_UNCORE_PERFEVTSEL5	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C6H, 950	MSR_UNCORE_PERFEVTSEL6	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package
Register Address: 3C7H, 951	MSR_UNCORE_PERFEVTSEL7	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

2.8.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

The Intel Xeon Processor 7500 series supports MSRs listed in Table 2-15 (except MSR address 1ADH) and additional model-specific registers listed in Table 2-17. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2EH.

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 394H, 816	MSR_W_PMON_FIXED_CTR	
Uncore W-box PerfMon fixed counter.		Package
Register Address: 395H, 817	MSR_W_PMON_FIXED_CTR_CTL	
Uncore U-box PerfMon fixed counter control MSR.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 18.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: C00H, 3072	MSR_U_PMON_GLOBAL_CTRL	
Uncore U-box PerfMon global control MSR.		Package
Register Address: C01H, 3073	MSR_U_PMON_GLOBAL_STATUS	
Uncore U-box PerfMon global status MSR.		Package
Register Address: C02H, 3074	MSR_U_PMON_GLOBAL_OVF_CTRL	
Uncore U-box PerfMon global overflow control MSR.		Package
Register Address: C10H, 3088	MSR_U_PMON_EVNT_SEL	
Uncore U-box PerfMon event select MSR.		Package
Register Address: C11H, 3089	MSR_U_PMON_CTR	
Uncore U-box PerfMon counter MSR.		Package
Register Address: C20H, 3104	MSR_B0_PMON_BOX_CTRL	
Uncore B-box 0 PerfMon local box control MSR.		Package
Register Address: C21H, 3105	MSR_B0_PMON_BOX_STATUS	
Uncore B-box 0 PerfMon local box status MSR.		Package
Register Address: C22H, 3106	MSR_B0_PMON_BOX_OVF_CTRL	
Uncore B-box 0 PerfMon local box overflow control MSR.		Package
Register Address: C30H, 3120	MSR_B0_PMON_EVNT_SELO	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C31H, 3121	MSR_B0_PMON_CTR0	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C32H, 3122	MSR_B0_PMON_EVNT_SEL1	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C33H, 3123	MSR_B0_PMON_CTR1	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C34H, 3124	MSR_B0_PMON_EVNT_SEL2	
Uncore B-box 0 PerfMon event select MSR.		Package
Register Address: C35H, 3125	MSR_B0_PMON_CTR2	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C36H, 3126	MSR_B0_PMON_EVNT_SEL3	
Uncore B-box 0 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C37H, 3127	MSR_BO_PMON_CTR3	
Uncore B-box 0 PerfMon counter MSR.		Package
Register Address: C40H, 3136	MSR_SO_PMON_BOX_CTRL	
Uncore S-box 0 PerfMon local box control MSR.		Package
Register Address: C41H, 3137	MSR_SO_PMON_BOX_STATUS	
Uncore S-box 0 PerfMon local box status MSR.		Package
Register Address: C42H, 3138	MSR_SO_PMON_BOX_OVF_CTRL	
Uncore S-box 0 PerfMon local box overflow control MSR.		Package
Register Address: C50H, 3152	MSR_SO_PMON_EVNT_SELO	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C51H, 3153	MSR_SO_PMON_CTR0	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C52H, 3154	MSR_SO_PMON_EVNT_SEL1	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C53H, 3155	MSR_SO_PMON_CTR1	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C54H, 3156	MSR_SO_PMON_EVNT_SEL2	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C55H, 3157	MSR_SO_PMON_CTR2	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C56H, 3158	MSR_SO_PMON_EVNT_SEL3	
Uncore S-box 0 PerfMon event select MSR.		Package
Register Address: C57H, 3159	MSR_SO_PMON_CTR3	
Uncore S-box 0 PerfMon counter MSR.		Package
Register Address: C60H, 3168	MSR_B1_PMON_BOX_CTRL	
Uncore B-box 1 PerfMon local box control MSR.		Package
Register Address: C61H, 3169	MSR_B1_PMON_BOX_STATUS	
Uncore B-box 1 PerfMon local box status MSR.		Package
Register Address: C62H, 3170	MSR_B1_PMON_BOX_OVF_CTRL	
Uncore B-box 1 PerfMon local box overflow control MSR.		Package
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SELO	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTR2	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vPerfMon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTR3	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C80H, 3120	MSR_W_PMON_BOX_CTRL	
Uncore W-box PerfMon local box control MSR.		Package
Register Address: C81H, 3121	MSR_W_PMON_BOX_STATUS	
Uncore W-box PerfMon local box status MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box PerfMon local box overflow control MSR.		Package
Register Address: C90H, 3136	MSR_W_PMON_EVNT_SELO	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C91H, 3137	MSR_W_PMON_CTR0	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C92H, 3138	MSR_W_PMON_EVNT_SEL1	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C93H, 3139	MSR_W_PMON_CTR1	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C94H, 3140	MSR_W_PMON_EVNT_SEL2	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C95H, 3141	MSR_W_PMON_CTR2	
Uncore W-box PerfMon counter MSR.		Package
Register Address: C96H, 3142	MSR_W_PMON_EVNT_SEL3	
Uncore W-box PerfMon event select MSR.		Package
Register Address: C97H, 3143	MSR_W_PMON_CTR3	
Uncore W-box PerfMon counter MSR.		Package
Register Address: CA0H, 3232	MSR_M0_PMON_BOX_CTRL	
Uncore M-box 0 PerfMon local box control MSR.		Package
Register Address: CA1H, 3233	MSR_M0_PMON_BOX_STATUS	
Uncore M-box 0 PerfMon local box status MSR.		Package
Register Address: CA2H, 3234	MSR_M0_PMON_BOX_OVF_CTRL	
Uncore M-box 0 PerfMon local box overflow control MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CA4H, 3236	MSR_MO_PMON_TIMESTAMP	
Uncore M-box 0 PerfMon time stamp unit select MSR.		Package
Register Address: CA5H, 3237	MSR_MO_PMON_DSP	
Uncore M-box 0 PerfMon DSP unit select MSR.		Package
Register Address: CA6H, 3238	MSR_MO_PMON_ISS	
Uncore M-box 0 PerfMon ISS unit select MSR.		Package
Register Address: CA7H, 3239	MSR_MO_PMON_MAP	
Uncore M-box 0 PerfMon MAP unit select MSR.		Package
Register Address: CA8H, 3240	MSR_MO_PMON_MSC_THR	
Uncore M-box 0 PerfMon MIC THR select MSR.		Package
Register Address: CA9H, 3241	MSR_MO_PMON_PGT	
Uncore M-box 0 PerfMon PGT unit select MSR.		Package
Register Address: CAAH, 3242	MSR_MO_PMON_PLD	
Uncore M-box 0 PerfMon PLD unit select MSR.		Package
Register Address: CABH, 3243	MSR_MO_PMON_ZDP	
Uncore M-box 0 PerfMon ZDP unit select MSR.		Package
Register Address: CB0H, 3248	MSR_MO_PMON_EVNT_SELO	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB1H, 3249	MSR_MO_PMON_CTR0	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB2H, 3250	MSR_MO_PMON_EVNT_SEL1	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB3H, 3251	MSR_MO_PMON_CTR1	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB4H, 3252	MSR_MO_PMON_EVNT_SEL2	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB5H, 3253	MSR_MO_PMON_CTR2	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB6H, 3254	MSR_MO_PMON_EVNT_SEL3	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB7H, 3255	MSR_MO_PMON_CTR3	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CB8H, 3256	MSR_MO_PMON_EVNT_SEL4	
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CB9H, 3257	MSR_MO_PMON_CTR4	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CBAH, 3258	MSR_MO_PMON_EVNT_SEL5	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore M-box 0 PerfMon event select MSR.		Package
Register Address: CBBH, 3259	MSR_M0_PMON_CTR5	
Uncore M-box 0 PerfMon counter MSR.		Package
Register Address: CC0H, 3264	MSR_S1_PMON_BOX_CTRL	
Uncore S-box 1 PerfMon local box control MSR.		Package
Register Address: CC1H, 3265	MSR_S1_PMON_BOX_STATUS	
Uncore S-box 1 PerfMon local box status MSR.		Package
Register Address: CC2H, 3266	MSR_S1_PMON_BOX_OVF_CTRL	
Uncore S-box 1 PerfMon local box overflow control MSR.		Package
Register Address: CD0H, 3280	MSR_S1_PMON_EVNT_SELO	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD1H, 3281	MSR_S1_PMON_CTR0	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD2H, 3282	MSR_S1_PMON_EVNT_SEL1	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD3H, 3283	MSR_S1_PMON_CTR1	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD4H, 3284	MSR_S1_PMON_EVNT_SEL2	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD5H, 3285	MSR_S1_PMON_CTR2	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CD6H, 3286	MSR_S1_PMON_EVNT_SEL3	
Uncore S-box 1 PerfMon event select MSR.		Package
Register Address: CD7H, 3287	MSR_S1_PMON_CTR3	
Uncore S-box 1 PerfMon counter MSR.		Package
Register Address: CE0H, 3296	MSR_M1_PMON_BOX_CTRL	
Uncore M-box 1 PerfMon local box control MSR.		Package
Register Address: CE1H, 3297	MSR_M1_PMON_BOX_STATUS	
Uncore M-box 1 PerfMon local box status MSR.		Package
Register Address: CE2H, 3298	MSR_M1_PMON_BOX_OVF_CTRL	
Uncore M-box 1 PerfMon local box overflow control MSR.		Package
Register Address: CE4H, 3300	MSR_M1_PMON_TIMESTAMP	
Uncore M-box 1 PerfMon time stamp unit select MSR.		Package
Register Address: CE5H, 3301	MSR_M1_PMON_DSP	
Uncore M-box 1 PerfMon DSP unit select MSR.		Package
Register Address: CE6H, 3302	MSR_M1_PMON_ISS	
Uncore M-box 1 PerfMon ISS unit select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CE7H, 3303	MSR_M1_PMON_MAP	
Uncore M-box 1 PerfMon MAP unit select MSR.		Package
Register Address: CE8H, 3304	MSR_M1_PMON_MSC_THR	
Uncore M-box 1 PerfMon MIC THR select MSR.		Package
Register Address: CE9H, 3305	MSR_M1_PMON_PGT	
Uncore M-box 1 PerfMon PGT unit select MSR.		Package
Register Address: CEAH, 3306	MSR_M1_PMON_PLD	
Uncore M-box 1 PerfMon PLD unit select MSR.		Package
Register Address: CEBH, 3307	MSR_M1_PMON_ZDP	
Uncore M-box 1 PerfMon ZDP unit select MSR.		Package
Register Address: CF0H, 3312	MSR_M1_PMON_EVNT_SELO	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF1H, 3313	MSR_M1_PMON_CTR0	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF2H, 3314	MSR_M1_PMON_EVNT_SEL1	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF3H, 3315	MSR_M1_PMON_CTR1	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF4H, 3316	MSR_M1_PMON_EVNT_SEL2	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF5H, 3317	MSR_M1_PMON_CTR2	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF6H, 3318	MSR_M1_PMON_EVNT_SEL3	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF7H, 3319	MSR_M1_PMON_CTR3	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CF8H, 3320	MSR_M1_PMON_EVNT_SEL4	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CF9H, 3321	MSR_M1_PMON_CTR4	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: CFAH, 3322	MSR_M1_PMON_EVNT_SEL5	
Uncore M-box 1 PerfMon event select MSR.		Package
Register Address: CFBH, 3323	MSR_M1_PMON_CTR5	
Uncore M-box 1 PerfMon counter MSR.		Package
Register Address: D00H, 3328	MSR_CO_PMON_BOX_CTRL	
Uncore C-box 0 PerfMon local box control MSR.		Package
Register Address: D01H, 3329	MSR_CO_PMON_BOX_STATUS	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 0 PerfMon local box status MSR.		Package
Register Address: D02H, 3330	MSR_CO_PMON_BOX_OVF_CTRL	
Uncore C-box 0 PerfMon local box overflow control MSR.		Package
Register Address: D10H, 3344	MSR_CO_PMON_EVNT_SELO	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D11H, 3345	MSR_CO_PMON_CTRL0	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D12H, 3346	MSR_CO_PMON_EVNT_SEL1	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D13H, 3347	MSR_CO_PMON_CTRL1	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D14H, 3348	MSR_CO_PMON_EVNT_SEL2	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D15H, 3349	MSR_CO_PMON_CTRL2	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D16H, 3350	MSR_CO_PMON_EVNT_SEL3	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D17H, 3351	MSR_CO_PMON_CTRL3	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D18H, 3352	MSR_CO_PMON_EVNT_SEL4	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D19H, 3353	MSR_CO_PMON_CTRL4	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D1AH, 3354	MSR_CO_PMON_EVNT_SEL5	
Uncore C-box 0 PerfMon event select MSR.		Package
Register Address: D1BH, 3355	MSR_CO_PMON_CTRL5	
Uncore C-box 0 PerfMon counter MSR.		Package
Register Address: D20H, 3360	MSR_C4_PMON_BOX_CTRL	
Uncore C-box 4 PerfMon local box control MSR.		Package
Register Address: D21H, 3361	MSR_C4_PMON_BOX_STATUS	
Uncore C-box 4 PerfMon local box status MSR.		Package
Register Address: D22H, 3362	MSR_C4_PMON_BOX_OVF_CTRL	
Uncore C-box 4 PerfMon local box overflow control MSR.		Package
Register Address: D30H, 3376	MSR_C4_PMON_EVNT_SELO	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D31H, 3377	MSR_C4_PMON_CTRL0	
Uncore C-box 4 PerfMon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D32H, 3378	MSR_C4_PMON_EVNT_SEL1	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D33H, 3379	MSR_C4_PMON_CTR1	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D34H, 3380	MSR_C4_PMON_EVNT_SEL2	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D35H, 3381	MSR_C4_PMON_CTR2	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D36H, 3382	MSR_C4_PMON_EVNT_SEL3	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D37H, 3383	MSR_C4_PMON_CTR3	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D38H, 3384	MSR_C4_PMON_EVNT_SEL4	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D39H, 3385	MSR_C4_PMON_CTR4	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D3AH, 3386	MSR_C4_PMON_EVNT_SEL5	
Uncore C-box 4 PerfMon event select MSR.		Package
Register Address: D3BH, 3387	MSR_C4_PMON_CTR5	
Uncore C-box 4 PerfMon counter MSR.		Package
Register Address: D40H, 3392	MSR_C2_PMON_BOX_CTRL	
Uncore C-box 2 PerfMon local box control MSR.		Package
Register Address: D41H, 3393	MSR_C2_PMON_BOX_STATUS	
Uncore C-box 2 PerfMon local box status MSR.		Package
Register Address: D42H, 3394	MSR_C2_PMON_BOX_OVF_CTRL	
Uncore C-box 2 PerfMon local box overflow control MSR.		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVNT_SELO	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D51H, 3409	MSR_C2_PMON_CTR0	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNT_SEL1	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D53H, 3411	MSR_C2_PMON_CTR1	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D54H, 3412	MSR_C2_PMON_EVNT_SEL2	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D55H, 3413	MSR_C2_PMON_CTR2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D56H, 3414	MSR_C2_PMON_EVNT_SEL3	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR3	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D58H, 3416	MSR_C2_PMON_EVNT_SEL4	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR4	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D5AH, 3418	MSR_C2_PMON_EVNT_SEL5	
Uncore C-box 2 PerfMon event select MSR.		Package
Register Address: D5BH, 3419	MSR_C2_PMON_CTR5	
Uncore C-box 2 PerfMon counter MSR.		Package
Register Address: D60H, 3424	MSR_C6_PMON_BOX_CTRL	
Uncore C-box 6 PerfMon local box control MSR.		Package
Register Address: D61H, 3425	MSR_C6_PMON_BOX_STATUS	
Uncore C-box 6 PerfMon local box status MSR.		Package
Register Address: D62H, 3426	MSR_C6_PMON_BOX_OVF_CTRL	
Uncore C-box 6 PerfMon local box overflow control MSR.		Package
Register Address: D70H, 3440	MSR_C6_PMON_EVNT_SELO	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D71H, 3441	MSR_C6_PMON_CTR0	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D72H, 3442	MSR_C6_PMON_EVNT_SEL1	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D73H, 3443	MSR_C6_PMON_CTR1	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D74H, 3444	MSR_C6_PMON_EVNT_SEL2	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D75H, 3445	MSR_C6_PMON_CTR2	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D76H, 3446	MSR_C6_PMON_EVNT_SEL3	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D77H, 3447	MSR_C6_PMON_CTR3	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D78H, 3448	MSR_C6_PMON_EVNT_SEL4	
Uncore C-box 6 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D79H, 3449	MSR_C6_PMON_CTR4	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D7AH, 3450	MSR_C6_PMON_EVNT_SEL5	
Uncore C-box 6 PerfMon event select MSR.		Package
Register Address: D7BH, 3451	MSR_C6_PMON_CTR5	
Uncore C-box 6 PerfMon counter MSR.		Package
Register Address: D80H, 3456	MSR_C1_PMON_BOX_CTRL	
Uncore C-box 1 PerfMon local box control MSR.		Package
Register Address: D81H, 3457	MSR_C1_PMON_BOX_STATUS	
Uncore C-box 1 PerfMon local box status MSR.		Package
Register Address: D82H, 3458	MSR_C1_PMON_BOX_OVF_CTRL	
Uncore C-box 1 PerfMon local box overflow control MSR.		Package
Register Address: D90H, 3472	MSR_C1_PMON_EVNT_SELO	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D91H, 3473	MSR_C1_PMON_CTR0	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D92H, 3474	MSR_C1_PMON_EVNT_SEL1	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D93H, 3475	MSR_C1_PMON_CTR1	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D94H, 3476	MSR_C1_PMON_EVNT_SEL2	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D95H, 3477	MSR_C1_PMON_CTR2	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D96H, 3478	MSR_C1_PMON_EVNT_SEL3	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D97H, 3479	MSR_C1_PMON_CTR3	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D98H, 3480	MSR_C1_PMON_EVNT_SEL4	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D99H, 3481	MSR_C1_PMON_CTR4	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: D9AH, 3482	MSR_C1_PMON_EVNT_SEL5	
Uncore C-box 1 PerfMon event select MSR.		Package
Register Address: D9BH, 3483	MSR_C1_PMON_CTR5	
Uncore C-box 1 PerfMon counter MSR.		Package
Register Address: DA0H, 3488	MSR_C5_PMON_BOX_CTRL	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 5 PerfMon local box control MSR.		Package
Register Address: DA1H, 3489	MSR_C5_PMON_BOX_STATUS	
Uncore C-box 5 PerfMon local box status MSR.		Package
Register Address: DA2H, 3490	MSR_C5_PMON_BOX_OVF_CTRL	
Uncore C-box 5 PerfMon local box overflow control MSR.		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNT_SELO	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB1H, 3505	MSR_C5_PMON_CTR0	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNT_SEL1	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB3H, 3507	MSR_C5_PMON_CTR1	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB4H, 3508	MSR_C5_PMON_EVNT_SEL2	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB5H, 3509	MSR_C5_PMON_CTR2	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB6H, 3510	MSR_C5_PMON_EVNT_SEL3	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR3	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DB8H, 3512	MSR_C5_PMON_EVNT_SEL4	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR4	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DBAH, 3514	MSR_C5_PMON_EVNT_SEL5	
Uncore C-box 5 PerfMon event select MSR.		Package
Register Address: DBBH, 3515	MSR_C5_PMON_CTR5	
Uncore C-box 5 PerfMon counter MSR.		Package
Register Address: DC0H, 3520	MSR_C3_PMON_BOX_CTRL	
Uncore C-box 3 PerfMon local box control MSR.		Package
Register Address: DC1H, 3521	MSR_C3_PMON_BOX_STATUS	
Uncore C-box 3 PerfMon local box status MSR.		Package
Register Address: DC2H, 3522	MSR_C3_PMON_BOX_OVF_CTRL	
Uncore C-box 3 PerfMon local box overflow control MSR.		Package
Register Address: DD0H, 3536	MSR_C3_PMON_EVNT_SELO	
Uncore C-box 3 PerfMon event select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DD1H, 3537	MSR_C3_PMON_CTRL0	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD2H, 3538	MSR_C3_PMON_EVNT_SEL1	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD3H, 3539	MSR_C3_PMON_CTRL1	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD4H, 3540	MSR_C3_PMON_EVNT_SEL2	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD5H, 3541	MSR_C3_PMON_CTRL2	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD6H, 3542	MSR_C3_PMON_EVNT_SEL3	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD7H, 3543	MSR_C3_PMON_CTRL3	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DD8H, 3544	MSR_C3_PMON_EVNT_SEL4	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DD9H, 3545	MSR_C3_PMON_CTRL4	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DDAH, 3546	MSR_C3_PMON_EVNT_SEL5	
Uncore C-box 3 PerfMon event select MSR.		Package
Register Address: DDBH, 3547	MSR_C3_PMON_CTRL5	
Uncore C-box 3 PerfMon counter MSR.		Package
Register Address: DE0H, 3552	MSR_C7_PMON_BOX_CTRL	
Uncore C-box 7 PerfMon local box control MSR.		Package
Register Address: DE1H, 3553	MSR_C7_PMON_BOX_STATUS	
Uncore C-box 7 PerfMon local box status MSR.		Package
Register Address: DE2H, 3554	MSR_C7_PMON_BOX_OVF_CTRL	
Uncore C-box 7 PerfMon local box overflow control MSR.		Package
Register Address: DF0H, 3568	MSR_C7_PMON_EVNT_SELO	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF1H, 3569	MSR_C7_PMON_CTRL0	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVNT_SEL1	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF3H, 3571	MSR_C7_PMON_CTRL1	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF4H, 3572	MSR_C7_PMON_EVNT_SEL2	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF5H, 3573	MSR_C7_PMON_CTR2	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF6H, 3574	MSR_C7_PMON_EVNT_SEL3	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTR3	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DF8H, 3576	MSR_C7_PMON_EVNT_SEL4	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR4	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: DFAH, 3578	MSR_C7_PMON_EVNT_SEL5	
Uncore C-box 7 PerfMon event select MSR.		Package
Register Address: DFBH, 3579	MSR_C7_PMON_CTR5	
Uncore C-box 7 PerfMon counter MSR.		Package
Register Address: E00H, 3584	MSR_R0_PMON_BOX_CTRL	
Uncore R-box 0 PerfMon local box control MSR.		Package
Register Address: E01H, 3585	MSR_R0_PMON_BOX_STATUS	
Uncore R-box 0 PerfMon local box status MSR.		Package
Register Address: E02H, 3586	MSR_R0_PMON_BOX_OVF_CTRL	
Uncore R-box 0 PerfMon local box overflow control MSR.		Package
Register Address: E04H, 3588	MSR_R0_PMON_IPERF0_P0	
Uncore R-box 0 PerfMon IPERF0 unit Port 0 select MSR.		Package
Register Address: E05H, 3589	MSR_R0_PMON_IPERF0_P1	
Uncore R-box 0 PerfMon IPERF0 unit Port 1 select MSR.		Package
Register Address: E06H, 3590	MSR_R0_PMON_IPERF0_P2	
Uncore R-box 0 PerfMon IPERF0 unit Port 2 select MSR.		Package
Register Address: E07H, 3591	MSR_R0_PMON_IPERF0_P3	
Uncore R-box 0 PerfMon IPERF0 unit Port 3 select MSR.		Package
Register Address: E08H, 3592	MSR_R0_PMON_IPERF0_P4	
Uncore R-box 0 PerfMon IPERF0 unit Port 4 select MSR.		Package
Register Address: E09H, 3593	MSR_R0_PMON_IPERF0_P5	
Uncore R-box 0 PerfMon IPERF0 unit Port 5 select MSR.		Package
Register Address: E0AH, 3594	MSR_R0_PMON_IPERF0_P6	
Uncore R-box 0 PerfMon IPERF0 unit Port 6 select MSR.		Package
Register Address: E0BH, 3595	MSR_R0_PMON_IPERF0_P7	
Uncore R-box 0 PerfMon IPERF0 unit Port 7 select MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E0CH, 3596	MSR_R0_PMON_QLX_P0	
Uncore R-box 0 PerfMon QLX unit Port 0 select MSR.		Package
Register Address: E0DH, 3597	MSR_R0_PMON_QLX_P1	
Uncore R-box 0 PerfMon QLX unit Port 1 select MSR.		Package
Register Address: E0EH, 3598	MSR_R0_PMON_QLX_P2	
Uncore R-box 0 PerfMon QLX unit Port 2 select MSR.		Package
Register Address: E0FH, 3599	MSR_R0_PMON_QLX_P3	
Uncore R-box 0 PerfMon QLX unit Port 3 select MSR.		Package
Register Address: E10H, 3600	MSR_R0_PMON_EVNT_SELO	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E11H, 3601	MSR_R0_PMON_CTR0	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E12H, 3602	MSR_R0_PMON_EVNT_SEL1	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E13H, 3603	MSR_R0_PMON_CTR1	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E14H, 3604	MSR_R0_PMON_EVNT_SEL2	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E15H, 3605	MSR_R0_PMON_CTR2	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E16H, 3606	MSR_R0_PMON_EVNT_SEL3	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E17H, 3607	MSR_R0_PMON_CTR3	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E18H, 3608	MSR_R0_PMON_EVNT_SEL4	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E19H, 3609	MSR_R0_PMON_CTR4	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1AH, 3610	MSR_R0_PMON_EVNT_SEL5	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1BH, 3611	MSR_R0_PMON_CTR5	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1CH, 3612	MSR_R0_PMON_EVNT_SEL6	
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1DH, 3613	MSR_R0_PMON_CTR6	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E1EH, 3614	MSR_R0_PMON_EVNT_SEL7	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore R-box 0 PerfMon event select MSR.		Package
Register Address: E1FH, 3615	MSR_R0_PMON_CTR7	
Uncore R-box 0 PerfMon counter MSR.		Package
Register Address: E20H, 3616	MSR_R1_PMON_BOX_CTRL	
Uncore R-box 1 PerfMon local box control MSR.		Package
Register Address: E21H, 3617	MSR_R1_PMON_BOX_STATUS	
Uncore R-box 1 PerfMon local box status MSR.		Package
Register Address: E22H, 3618	MSR_R1_PMON_BOX_OVF_CTRL	
Uncore R-box 1 PerfMon local box overflow control MSR.		Package
Register Address: E24H, 3620	MSR_R1_PMON_IPERF1_P8	
Uncore R-box 1 PerfMon IPERF1 unit Port 8 select MSR.		Package
Register Address: E25H, 3621	MSR_R1_PMON_IPERF1_P9	
Uncore R-box 1 PerfMon IPERF1 unit Port 9 select MSR.		Package
Register Address: E26H, 3622	MSR_R1_PMON_IPERF1_P10	
Uncore R-box 1 PerfMon IPERF1 unit Port 10 select MSR.		Package
Register Address: E27H, 3623	MSR_R1_PMON_IPERF1_P11	
Uncore R-box 1 PerfMon IPERF1 unit Port 11 select MSR.		Package
Register Address: E28H, 3624	MSR_R1_PMON_IPERF1_P12	
Uncore R-box 1 PerfMon IPERF1 unit Port 12 select MSR.		Package
Register Address: E29H, 3625	MSR_R1_PMON_IPERF1_P13	
Uncore R-box 1 PerfMon IPERF1 unit Port 13 select MSR.		Package
Register Address: E2AH, 3626	MSR_R1_PMON_IPERF1_P14	
Uncore R-box 1 PerfMon IPERF1 unit Port 14 select MSR.		Package
Register Address: E2BH, 3627	MSR_R1_PMON_IPERF1_P15	
Uncore R-box 1 PerfMon IPERF1 unit Port 15 select MSR.		Package
Register Address: E2CH, 3628	MSR_R1_PMON_QLX_P4	
Uncore R-box 1 PerfMon QLX unit Port 4 select MSR.		Package
Register Address: E2DH, 3629	MSR_R1_PMON_QLX_P5	
Uncore R-box 1 PerfMon QLX unit Port 5 select MSR.		Package
Register Address: E2EH, 3630	MSR_R1_PMON_QLX_P6	
Uncore R-box 1 PerfMon QLX unit Port 6 select MSR.		Package
Register Address: E2FH, 3631	MSR_R1_PMON_QLX_P7	
Uncore R-box 1 PerfMon QLX unit Port 7 select MSR.		Package
Register Address: E30H, 3632	MSR_R1_PMON_EVNT_SEL8	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E31H, 3633	MSR_R1_PMON_CTR8	
Uncore R-box 1 PerfMon counter MSR.		Package

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E32H, 3634	MSR_R1_PMON_EVNT_SEL9	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E33H, 3635	MSR_R1_PMON_CTR9	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E34H, 3636	MSR_R1_PMON_EVNT_SEL10	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E35H, 3637	MSR_R1_PMON_CTR10	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E36H, 3638	MSR_R1_PMON_EVNT_SEL11	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E37H, 3639	MSR_R1_PMON_CTR11	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E38H, 3640	MSR_R1_PMON_EVNT_SEL12	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E39H, 3641	MSR_R1_PMON_CTR12	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3AH, 3642	MSR_R1_PMON_EVNT_SEL13	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3BH, 3643	MSR_R1_PMON_CTR13	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3CH, 3644	MSR_R1_PMON_EVNT_SEL14	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3DH, 3645	MSR_R1_PMON_CTR14	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E3EH, 3646	MSR_R1_PMON_EVNT_SEL15	
Uncore R-box 1 PerfMon event select MSR.		Package
Register Address: E3FH, 3647	MSR_R1_PMON_CTR15	
Uncore R-box 1 PerfMon counter MSR.		Package
Register Address: E45H, 3653	MSR_B0_PMON_MATCH	
Uncore B-box 0 PerfMon local box match MSR.		Package
Register Address: E46H, 3654	MSR_B0_PMON_MASK	
Uncore B-box 0 PerfMon local box mask MSR.		Package
Register Address: E49H, 3657	MSR_S0_PMON_MATCH	
Uncore S-box 0 PerfMon local box match MSR.		Package
Register Address: E4AH, 3658	MSR_S0_PMON_MASK	
Uncore S-box 0 PerfMon local box mask MSR.		Package
Register Address: E4DH, 3661	MSR_B1_PMON_MATCH	

Table 2-17. Additional MSRs in the Intel® Xeon® Processor 7500 Series (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore B-box 1 PerfMon local box match MSR.		Package
Register Address: E4EH, 3662	MSR_B1_PMON_MASK	
Uncore B-box 1 PerfMon local box mask MSR.		Package
Register Address: E54H, 3668	MSR_M0_PMON_MM_CONFIG	
Uncore M-box 0 PerfMon local box address match/mask config MSR.		Package
Register Address: E55H, 3669	MSR_M0_PMON_ADDR_MATCH	
Uncore M-box 0 PerfMon local box address match MSR.		Package
Register Address: E56H, 3670	MSR_M0_PMON_ADDR_MASK	
Uncore M-box 0 PerfMon local box address mask MSR.		Package
Register Address: E59H, 3673	MSR_S1_PMON_MATCH	
Uncore S-box 1 PerfMon local box match MSR.		Package
Register Address: E5AH, 3674	MSR_S1_PMON_MASK	
Uncore S-box 1 PerfMon local box mask MSR.		Package
Register Address: E5CH, 3676	MSR_M1_PMON_MM_CONFIG	
Uncore M-box 1 PerfMon local box address match/mask config MSR.		Package
Register Address: E5DH, 3677	MSR_M1_PMON_ADDR_MATCH	
Uncore M-box 1 PerfMon local box address match MSR.		Package
Register Address: E5EH, 3678	MSR_M1_PMON_ADDR_MASK	
Uncore M-box 1 PerfMon local box address mask MSR.		Package
Register Address: 3B5H, 965	MSR_UNCORE_PMC5	
See Section 21.3.1.2.2, "Uncore Performance Event Configuration Facility."		Package

2.9 MSRS IN THE INTEL® XEON® PROCESSOR 5600 SERIES BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor 5600 Series is based on Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15, Table 2-16, plus additional MSRs listed in Table 2-18. These MSRs apply to the Intel Core i7, i5, and i3 processor family with a CPUID Signature DisplayFamily_DisplayModel value of 06_25H or 06_2CH; see Table 2-1.

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core

Table 2-18. Additional MSRs Supported by Intel® Processors Based on Westmere Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
63:48	Reserved.	
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package

2.10 MSRS IN THE INTEL® XEON® PROCESSOR E7 FAMILY BASED ON WESTMERE MICROARCHITECTURE

The Intel® Xeon® Processor E7 Family is based on the Westmere microarchitecture and supports the MSR interfaces listed in Table 2-15 (except MSR address 1ADH), Table 2-16, plus additional MSRs listed in Table 2-19. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2FH.

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Reserved. Attempt to read/write will cause #UD.		Package
Register Address: 1BOH, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: F40H, 3904	MSR_C8_PMON_BOX_CTRL	
Uncore C-box 8 PerfMon local box control MSR.		Package
Register Address: F41H, 3905	MSR_C8_PMON_BOX_STATUS	
Uncore C-box 8 PerfMon local box status MSR.		Package
Register Address: F42H, 3906	MSR_C8_PMON_BOX_OVF_CTRL	
Uncore C-box 8 PerfMon local box overflow control MSR.		Package
Register Address: F50H, 3920	MSR_C8_PMON_EVNT_SELO	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F51H, 3921	MSR_C8_PMON_CTR0	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F52H, 3922	MSR_C8_PMON_EVNT_SEL1	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F53H, 3923	MSR_C8_PMON_CTR1	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F54H, 3924	MSR_C8_PMON_EVNT_SEL2	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F55H, 3925	MSR_C8_PMON_CTR2	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F56H, 3926	MSR_C8_PMON_EVNT_SEL3	
Uncore C-box 8 PerfMon event select MSR.		Package

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: F57H, 3927	MSR_C8_PMON_CTR3	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F58H, 3928	MSR_C8_PMON_EVNT_SEL4	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F59H, 3929	MSR_C8_PMON_CTR4	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: F5AH, 3930	MSR_C8_PMON_EVNT_SEL5	
Uncore C-box 8 PerfMon event select MSR.		Package
Register Address: F5BH, 3931	MSR_C8_PMON_CTR5	
Uncore C-box 8 PerfMon counter MSR.		Package
Register Address: FC0H, 4032	MSR_C9_PMON_BOX_CTRL	
Uncore C-box 9 PerfMon local box control MSR.		Package
Register Address: FC1H, 4033	MSR_C9_PMON_BOX_STATUS	
Uncore C-box 9 PerfMon local box status MSR.		Package
Register Address: FC2H, 4034	MSR_C9_PMON_BOX_OVF_CTRL	
Uncore C-box 9 PerfMon local box overflow control MSR.		Package
Register Address: FD0H, 4048	MSR_C9_PMON_EVNT_SEL0	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD1H, 4049	MSR_C9_PMON_CTR0	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD2H, 4050	MSR_C9_PMON_EVNT_SEL1	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD3H, 4051	MSR_C9_PMON_CTR1	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD4H, 4052	MSR_C9_PMON_EVNT_SEL2	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD5H, 4053	MSR_C9_PMON_CTR2	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD6H, 4054	MSR_C9_PMON_EVNT_SEL3	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD7H, 4055	MSR_C9_PMON_CTR3	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FD8H, 4056	MSR_C9_PMON_EVNT_SEL4	
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FD9H, 4057	MSR_C9_PMON_CTR4	
Uncore C-box 9 PerfMon counter MSR.		Package
Register Address: FDAH, 4058	MSR_C9_PMON_EVNT_SEL5	

Table 2-19. Additional MSRs Supported by the Intel® Xeon® Processor E7 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 9 PerfMon event select MSR.		Package
Register Address: FDBH, 4059	MSR_C9_PMON_CTR5	
Uncore C-box 9 PerfMon counter MSR.		Package

2.11 MSRS IN THE INTEL® PROCESSOR FAMILY BASED ON SANDY BRIDGE MICROARCHITECTURE

Table 2-20 lists model-specific registers (MSRs) that are common to the Intel® processor family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH or 06_2DH; see Table 2-1. Additional MSRs specific to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH are listed in Table 2-21.

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Thread
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, "Time-Stamp Counter," and see Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 12.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O) Count SMIs.	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: C3H, 195	IA32_PMC2	
Performance Counter Register See Table 2-2.		Thread
Register Address: C4H, 196	IA32_PMC3	
Performance Counter Register See Table 2-2.		Thread
Register Address: C5H, 197	IA32_PMC4	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C6H, 198	IA32_PMC5	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C7H, 199	IA32_PMC6	
Performance Counter Register (if core not shared by threads)		Core
Register Address: C8H, 200	IA32_PMC7	
Performance Counter Register (if core not shared by threads)		Core
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Redirection in C-state (R/W) See http://biosbits.org .		Core
15:0	LVL_2 Base Address (R/W) Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	C-State Range (R/W) Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL[bit 10]: 000b - C3 is the max C-State to include. 001b - C6 is the max C-State to include. 010b - C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
See Table 2-2.		Thread
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Thread
0	RIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.	
1	EIPV When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved.	
Register Address: 186H, 390	IA32_PERFEVTSEL0	
See Table 2-2.		Thread
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 188H, 392	IA32_PERFEVTSEL2	
See Table 2-2.		Thread
Register Address: 189H, 393	IA32_PERFEVTSEL3	
See Table 2-2.		Thread
Register Address: 18AH, 394	IA32_PERFEVTSEL4	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 4.		Core
Register Address: 18BH, 395	IA32_PERFEVTSEL5	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 5.		Core
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 6.		Core
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
See Table 2-2. If CPUID.0AH:EAX[15:8] > 7.		Core
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Package
15:0	Current Performance State Value	
63:16	Reserved.	
Register Address: 198H, 408	MSR_PERF_STATUS	
Performance Status		Package
47:32	Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2 ¹³).	
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.		Thread
3:0	On demand Clock Modulation Duty Cycle (R/W) In 6.25% increment.	
4	On demand Clock Modulation Enable (R/W)	
63:5	Reserved.	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Core
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WCO) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WCO) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WCO) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
15:12	Reserved.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
0	Fast-Strings Enable See Table 2-2.	Thread
6:1	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Thread
10:8	Reserved	
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Thread
12	Processor Event Based Sampling Unavailable (R/O) See Table 2-2.	Thread
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) See Table 2-2.	Package
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Thread
21:19	Reserved.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Limit CPUID Maxval (R/W) See Table 2-2.	Thread
23	xTPR Message Disable (R/W) See Table 2-2.	Thread
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Thread
37:35	Reserved.	
38	Turbo Mode Disable (R/W) When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H:EAX[1] = 0). When set to a 0 on processors that support IDA, CPUID.06H:EAX[1] reports the processor's support of turbo mode is enabled. Note: The power-on default value is used by BIOS to detect hardware support of turbo mode. If the power-on default value is 1, turbo mode is available in the processor. If the power-on default value is 0, turbo mode is not available.	Package
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Unique
15:0	Reserved.	
23:16	Temperature Target (R) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
63:24	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	Core
1	L2 Adjacent Cache Line Prefetcher Disable (R/W) If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	Core
2	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	Core
3	DCU IP Prefetcher Disable (R/W) If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	Core
63:4	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control Various model specific features enumeration. See http://biosbits.org .		
Register Address: 1BOH, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Package
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 19.9.2, "Filtering of Last Branch Records."		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
0	LBR: Last Branch Record	
1	BTF	
5:2	Reserved.	
6	TR: Branch Trace	
7	BTS: Log Branch Trace Message to BTS buffer	
8	BTINT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
9	BTS_OFF_OS	
10	BTS_OFF_USER	
11	FREEZE_LBR_ON_PMI	
12	FREEZE_PERFMON_ON_PMI	
13	ENABLE_UNCORE_PMI	
14	FREEZE_WHILE_SMM	
63:15	Reserved.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R/W) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R/W) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 1FCH, 508	MSR_POWER_CTL	
See http://biosbits.org .		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Thread
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Thread
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	
See Table 2-2.		Thread
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Thread
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Thread
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Thread
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Thread
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Thread
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Thread
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Thread
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Thread
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Thread
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Thread
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Thread
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Thread
Register Address: 210H, 528	IA32_MTRR_PHYSBASE8	
See Table 2-2.		Thread
Register Address: 211H, 529	IA32_MTRR_PHYSMASK8	
See Table 2-2.		Thread
Register Address: 212H, 530	IA32_MTRR_PHYSBASE9	
See Table 2-2.		Thread
Register Address: 213H, 531	IA32_MTRR_PHYSMASK9	
See Table 2-2.		Thread
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Thread
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Thread
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Thread
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Thread
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Thread
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Thread
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Thread
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Thread
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Thread
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Thread
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Thread
Register Address: 280H, 640	IA32_MC0_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
Always 0 (CMCI not supported).		Package
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Thread
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2 and Section 19.4.1, "IA32_DEBUGCTL MSR."		Thread
5:0	LBR Format See Table 2-2.	
6	PEBS Record Format.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7	PEBSSaveArchRegs See Table 2-2.	
11:8	PEBS_REC_FORMAT See Table 2-2.	
12	SMM_FREEZE See Table 2-2.	
63:13	Reserved.	
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 21.6.2.2, "Global Counter Control Facilities."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Core
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Core
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Core
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Core
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
60:35	Reserved.	
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2 and Section 21.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to enable PMC0 to count.	Thread
1	Set 1 to enable PMC1 to count.	Thread
2	Set 1 to enable PMC2 to count.	Thread
3	Set 1 to enable PMC3 to count.	Thread
4	Set 1 to enable PMC4 to count (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to enable PMC5 to count (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to enable PMC6 to count (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to enable PMC7 to count (if CPUID.0AH:EAX[15:8] > 7).	Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
31:8	Reserved.	
32	Set 1 to enable FixedCtr0 to count.	Thread
33	Set 1 to enable FixedCtr1 to count.	Thread
34	Set 1 to enable FixedCtr2 to count.	Thread
63:35	Reserved.	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 21.6.2.2, "Global Counter Control Facilities."		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Core
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Core
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Core
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Core
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
60:35	Reserved.	
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 21.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
0	Enable PEBS on IA32_PMC0. (R/W)	
1	Enable PEBS on IA32_PMC1. (R/W)	
2	Enable PEBS on IA32_PMC2. (R/W)	
3	Enable PEBS on IA32_PMC3. (R/W)	
31:4	Reserved.	
32	Enable Load Latency on IA32_PMC0. (R/W)	
33	Enable Load Latency on IA32_PMC1. (R/W)	
34	Enable Load Latency on IA32_PMC2. (R/W)	
35	Enable Load Latency on IA32_PMC3. (R/W)	
62:36	Reserved.	
63	Enable Precise Store (R/W)	
Register Address: 3F6H, 1014	MSR_PEBS_LD_LAT	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 21.3.1.1.2, "Load Latency Performance Monitoring Facility."		Thread
15:0	Minimum threshold latency value of tagged load operation that will be counted. (R/W)	
63:36	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C3 Residency Counter (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C7 Residency Counter (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C7 Residency Counter (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Core

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 401H, 1025	IA32_MC0_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Core
Register Address: 402H, 1026	IA32_MC0_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 403H, 1027	IA32_MC0_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 18.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
0	PCU Hardware Error (R/W) When set, enables signaling of PCU hardware detected errors.	
1	PCU Controller Error (R/W) When set, enables signaling of PCU controller detected errors.	
2	PCU Firmware Error (R/W) When set, enables signaling of PCU firmware detected errors.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:2	Reserved.	
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		Thread
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLs	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLs	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLs	
Capability Reporting Register of VM-Exit Controls (R/O) See Table 2-2 and Appendix A.4, "VM-Exit Controls."		Thread
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLs	
Capability Reporting Register of VM-Entry Controls (R/O) See Table 2-2 and Appendix A.5, "VM-Entry Controls."		Thread
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2 and Appendix A.6, "Miscellaneous Data."		Thread
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.7, "VMX-Fixed Bits in CR0."		Thread
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2 and Appendix A.8, "VMX-Fixed Bits in CR4."		Thread
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2 and Appendix A.9, "VMCS Enumeration."		Thread
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLs2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls."		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2		Thread
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLs	
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLs	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2		Thread
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLs	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2		Thread
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLs	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2		Thread
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 4C3H, 1219	IA32_A_PMC2	
See Table 2-2.		Thread
Register Address: 4C4H, 1220	IA32_A_PMC3	
See Table 2-2.		Thread
Register Address: 4C5H, 1221	IA32_A_PMC4	
See Table 2-2.		Core
Register Address: 4C6H, 1222	IA32_A_PMC5	
See Table 2-2.		Core
Register Address: 4C7H, 1223	IA32_A_PMC6	
See Table 2-2.		Core
Register Address: 4C8H, 1224	IA32_A_PMC7	
See Table 2-2.		Core
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2 and Section 21.6.3.4, "Debug Store (DS) Mechanism."		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers used in RAPL Interfaces (R/O) See Section 16.10.1, "RAPL Interfaces."		Package
Register Address: 60AH, 1546	MSR_PKG_C3_IRT_L	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C3 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 60BH, 1547	MSR_PKGC6_IRTL	
Package C6 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C6 to a C0 state, where an interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
63:0	Package C2 Residency Counter (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP	
Last Branch Record 0 From IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 19.9.1 and record format in Section 19.4.8.1.		Thread
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP	
Last Branch Record 1 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP	
Last Branch Record 2 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP	
Last Branch Record 3 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP	
Last Branch Record 4 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP	
Last Branch Record 5 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP	
Last Branch Record 6 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP	
Last Branch Record 7 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP	
Last Branch Record 8 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP	
Last Branch Record 9 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP	
Last Branch Record 10 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP	
Last Branch Record 11 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP	
Last Branch Record 12 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP	
Last Branch Record 13 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP	
Last Branch Record 14 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP	
Last Branch Record 15 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP	
Last Branch Record 0 To IP (R/W) One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction.		Thread
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP	
Last Branch Record 1 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 2 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP	
Last Branch Record 3 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP	
Last Branch Record 4 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP	
Last Branch Record 5 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP	
Last Branch Record 6 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP	
Last Branch Record 7 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP	
Last Branch Record 8 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP	
Last Branch Record 9 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP	
Last Branch Record 10 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP	
Last Branch Record 11 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP	
Last Branch Record 12 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP	
Last Branch Record 13 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP	
Last Branch Record 14 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP	

Table 2-20. MSRs Supported by Intel® Processors Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 15 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
See Table 2-2.		Thread
Register Address: 802H–83FH, 2050–2111	X2APIC MSRs	
See Table 2-2.		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2 and Section 19.17.2, “IA32_TSC_AUX Register and RDTSCP Support.”		Thread

2.11.1 MSRs in the 2nd Generation Intel® Core™ Processor Family Based on Sandy Bridge Microarchitecture

Table 2-21 and Table 2-22 list model-specific registers (MSRs) that are specific to the 2nd generation Intel® Core™ processor family based on the Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH; see Table 2-1.

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 60CH, 1548	MSR_PKG_C7_IRTL	
Package C7 Interrupt Response Limit (R/W) This MSR defines the budget allocated for the package to exit from a C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency may be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 639H, 1593	MSR_PP0_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 63AH, 1594	MSR_PP0_POLICY	

Table 2-21. MSRs Supported by the 2nd Generation Intel® Core™ Processors (Sandy Bridge Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
PP0 Balance Policy (R/W) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	
PP1 Energy Status (R/O) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-21, and Table 2-22 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.		

Table 2-22 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2AH.

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4 select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTRO	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFEVTSEL2	
Uncore C-Box 0, Counter 2 Event Select MSR		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFEVTSEL3	
Uncore C-Box 0, Counter 3 Event Select MSR		Package
Register Address: 705H, 1797	MSR_UNC_CBO_0_UNIT_STATUS	
Uncore C-Box 0, Unit Status for Counter 0-3		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 708H, 1800	MSR_UNC_CBO_0_PERFCTR2	
Uncore C-Box 0, Performance Counter 2		Package
Register Address: 709H, 1801	MSR_UNC_CBO_0_PERFCTR3	
Uncore C-Box 0, Performance Counter 3		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSEL0	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_1_PERFEVTSEL2	
Uncore C-Box 1, Counter 2 Event Select MSR		Package
Register Address: 713H, 1811	MSR_UNC_CBO_1_PERFEVTSEL3	
Uncore C-Box 1, Counter 3 Event Select MSR		Package
Register Address: 715H, 1813	MSR_UNC_CBO_1_UNIT_STATUS	
Uncore C-Box 1, Unit Status for Counter 0-3		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_1_PERFCTR2	
Uncore C-Box 1, Performance Counter 2		Package
Register Address: 719H, 1817	MSR_UNC_CBO_1_PERFCTR3	
Uncore C-Box 1, Performance Counter 3		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSEL0	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 722H, 1826	MSR_UNC_CBO_2_PERFEVTSEL2	
Uncore C-Box 2, Counter 2 Event Select MSR		Package
Register Address: 723H, 1827	MSR_UNC_CBO_2_PERFEVTSEL3	
Uncore C-Box 2, Counter 3 Event Select MSR		Package
Register Address: 725H, 1829	MSR_UNC_CBO_2_UNIT_STATUS	
Uncore C-Box 2, Unit Status for Counter 0-3		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_3_PERFCTR2	

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 729H, 1833	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_3_PERFEVTSEL2	
Uncore C-Box 3, Counter 2 Event Select MSR		Package
Register Address: 733H, 1843	MSR_UNC_CBO_3_PERFEVTSEL3	
Uncore C-Box 3, counter 3 Event Select MSR		Package
Register Address: 735H, 1845	MSR_UNC_CBO_3_UNIT_STATUS	
Uncore C-Box 3, Unit Status for Counter 0-3		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_3_PERFCTR2	
Uncore C-Box 3, Performance Counter 2		Package
Register Address: 739H, 1849	MSR_UNC_CBO_3_PERFCTR3	
Uncore C-Box 3, Performance Counter 3		Package
Register Address: 740H, 1856	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 741H, 1857	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 742H, 1858	MSR_UNC_CBO_4_PERFEVTSEL2	
Uncore C-Box 4, Counter 2 Event Select MSR		Package
Register Address: 743H, 1859	MSR_UNC_CBO_4_PERFEVTSEL3	
Uncore C-Box 4, Counter 3 Event Select MSR		Package
Register Address: 745H, 1861	MSR_UNC_CBO_4_UNIT_STATUS	
Uncore C-Box 4, Unit status for Counter 0-3		Package
Register Address: 746H, 1862	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 747H, 1863	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 748H, 1864	MSR_UNC_CBO_4_PERFCTR2	
Uncore C-Box 4, Performance Counter 2		Package

Table 2-22. Uncore PMU MSRs Supported by 2nd Generation Intel® Core™ Processors (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 749H, 1865	MSR_UNC_CBO_4_PERFCTR3	
Uncore C-Box 4, Performance Counter 3		Package

2.11.2 MSRs in the Intel® Xeon® Processor E5 Family Based on Sandy Bridge Microarchitecture

Table 2-23 lists additional model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 Family based on Sandy Bridge microarchitecture. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH, and also support MSRs listed in Table 2-20 and Table 2-24.

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
	MC Bank Error Configuration (R/W)	Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 cores active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 cores active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 cores active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 cores active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 cores active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 cores active.	Package
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 cores active.	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 39CH, 924	MSR_PEBS_NUM_ALT	
ENABLE_PEBS_NUM_ALT (R/W)		Package
0	ENABLE_PEBS_NUM_ALT (R/W) Write 1 to enable alternate PEBS counting logic for specific events requiring additional configuration, see https://perfmon-events.intel.com/ .	
63:1	Reserved, must be zero.	
Register Address: 414H, 1044	IA32_MCI_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MCI_STATUS	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 17.3.2.4, "IA32_MCi_MISC MSRs."		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs," and Chapter 18.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 18.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 18.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 18.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs," and Chapter 18.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."		Package

Table 2-23. Additional MSRs Supported by the Intel® Xeon® Processors E5 Family Based on Sandy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.2, "IA32_MCI_STATUS MSRS," and Chapter 18.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs."		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.4, "IA32_MCI_MISC MSRs."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-23, and Table 2-24 for MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.		

2.11.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 Family

Intel Xeon Processor E5 family is based on the Sandy Bridge microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_2DH.

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C08H, 3080	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-box UCLK Fixed Counter Control		Package
Register Address: C09H, 3081	MSR_U_PMON_UCLK_FIXED_CTR	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore U-box UCLK Fixed Counter		Package
Register Address: C10H, 3088	MSR_U_PMON_EVTNSELO	
Uncore U-box PerfMon Event Select for U-box Counter 0		Package
Register Address: C11H, 3089	MSR_U_PMON_EVTNSEL1	
Uncore U-box PerfMon Event Select for U-box Counter 1		Package
Register Address: C16H, 3094	MSR_U_PMON_CTR0	
Uncore U-box PerfMon Counter 0		Package
Register Address: C17H, 3095	MSR_U_PMON_CTR1	
Uncore U-box PerfMon Counter 1		Package
Register Address: C24H, 3108	MSR_PCU_PMON_BOX_CTL	
Uncore PCU PerfMon for PCU-box-wide Control		Package
Register Address: C30H, 3120	MSR_PCU_PMON_EVTNSELO	
Uncore PCU PerfMon Event Select for PCU Counter 0		Package
Register Address: C31H, 3121	MSR_PCU_PMON_EVTNSEL1	
Uncore PCU PerfMon Event Select for PCU Counter 1		Package
Register Address: C32H, 3122	MSR_PCU_PMON_EVTNSEL2	
Uncore PCU PerfMon Event Select for PCU Counter 2		Package
Register Address: C33H, 3123	MSR_PCU_PMON_EVTNSEL3	
Uncore PCU PerfMon Event Select for PCU Counter 3		Package
Register Address: C34H, 3124	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU PerfMon box-wide Filter		Package
Register Address: C36H, 3126	MSR_PCU_PMON_CTR0	
Uncore PCU PerfMon Counter 0		Package
Register Address: C37H, 3127	MSR_PCU_PMON_CTR1	
Uncore PCU PerfMon Counter 1		Package
Register Address: C38H, 3128	MSR_PCU_PMON_CTR2	
Uncore PCU PerfMon Counter 2		Package
Register Address: C39H, 3129	MSR_PCU_PMON_CTR3	
Uncore PCU PerfMon Counter 3		Package
Register Address: D04H, 3332	MSR_CO_PMON_BOX_CTL	
Uncore C-box 0 PerfMon Local Box Wide Control		Package
Register Address: D10H, 3344	MSR_CO_PMON_EVTNSELO	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 0		Package
Register Address: D11H, 3345	MSR_CO_PMON_EVTNSEL1	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 1		Package
Register Address: D12H, 3346	MSR_CO_PMON_EVTNSEL2	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 2		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D13H, 3347	MSR_CO_PMON_EVTNSEL3	
Uncore C-box 0 PerfMon Event Select for C-box 0 Counter 3		Package
Register Address: D14H, 3348	MSR_CO_PMON_BOX_FILTER	
Uncore C-box 0 PerfMon Box Wide Filter		Package
Register Address: D16H, 3350	MSR_CO_PMON_CTR0	
Uncore C-box 0 PerfMon Counter 0		Package
Register Address: D17H, 3351	MSR_CO_PMON_CTR1	
Uncore C-box 0 PerfMon Counter 1		Package
Register Address: D18H, 3352	MSR_CO_PMON_CTR2	
Uncore C-box 0 PerfMon Counter 2		Package
Register Address: D19H, 3353	MSR_CO_PMON_CTR3	
Uncore C-box 0 PerfMon Counter 3		Package
Register Address: D24H, 3364	MSR_C1_PMON_BOX_CTL	
Uncore C-box 1 PerfMon Local Box Wide Control		Package
Register Address: D30H, 3376	MSR_C1_PMON_EVTNSEL0	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 0		Package
Register Address: D31H, 3377	MSR_C1_PMON_EVTNSEL1	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 1		Package
Register Address: D32H, 3378	MSR_C1_PMON_EVTNSEL2	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 2		Package
Register Address: D33H, 3379	MSR_C1_PMON_EVTNSEL3	
Uncore C-box 1 PerfMon Event Select for C-box 1 Counter 3		Package
Register Address: D34H, 3380	MSR_C1_PMON_BOX_FILTER	
Uncore C-box 1 PerfMon Box Wide Filter		Package
Register Address: D36H, 3382	MSR_C1_PMON_CTR0	
Uncore C-box 1 PerfMon Counter 0		Package
Register Address: D37H, 3383	MSR_C1_PMON_CTR1	
Uncore C-box 1 PerfMon Counter 1		Package
Register Address: D38H, 3384	MSR_C1_PMON_CTR2	
Uncore C-box 1 PerfMon Counter 2		Package
Register Address: D39H, 3385	MSR_C1_PMON_CTR3	
Uncore C-box 1 PerfMon Counter 3		Package
Register Address: D44H, 3396	MSR_C2_PMON_BOX_CTL	
Uncore C-box 2 PerfMon Local Box Wide Control		Package
Register Address: D50H, 3408	MSR_C2_PMON_EVTNSEL0	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 0		Package
Register Address: D51H, 3409	MSR_C2_PMON_EVTNSEL1	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 1		Package
Register Address: D52H, 3410	MSR_C2_PMON_EVNTSEL2	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 2		Package
Register Address: D53H, 3411	MSR_C2_PMON_EVNTSEL3	
Uncore C-box 2 PerfMon Event Select for C-box 2 Counter 3		Package
Register Address: D54H, 3412	MSR_C2_PMON_BOX_FILTER	
Uncore C-box 2 PerfMon Box Wide Filter		Package
Register Address: D56H, 3414	MSR_C2_PMON_CTR0	
Uncore C-box 2 PerfMon Counter 0		Package
Register Address: D57H, 3415	MSR_C2_PMON_CTR1	
Uncore C-box 2 PerfMon Counter 1		Package
Register Address: D58H, 3416	MSR_C2_PMON_CTR2	
Uncore C-box 2 PerfMon Counter 2		Package
Register Address: D59H, 3417	MSR_C2_PMON_CTR3	
Uncore C-box 2 PerfMon Counter 3		Package
Register Address: D64H, 3428	MSR_C3_PMON_BOX_CTL	
Uncore C-box 3 PerfMon Local Box Wide Control		Package
Register Address: D70H, 3440	MSR_C3_PMON_EVNTSEL0	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 0		Package
Register Address: D71H, 3441	MSR_C3_PMON_EVNTSEL1	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 1		Package
Register Address: D72H, 3442	MSR_C3_PMON_EVNTSEL2	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 2		Package
Register Address: D73H, 3443	MSR_C3_PMON_EVNTSEL3	
Uncore C-box 3 PerfMon Event Select for C-box 3 Counter 3		Package
Register Address: D74H, 3444	MSR_C3_PMON_BOX_FILTER	
Uncore C-box 3 PerfMon Box Wide Filter		Package
Register Address: D76H, 3446	MSR_C3_PMON_CTR0	
Uncore C-box 3 PerfMon Counter 0		Package
Register Address: D77H, 3447	MSR_C3_PMON_CTR1	
Uncore C-box 3 PerfMon Counter 1		Package
Register Address: D78H, 3448	MSR_C3_PMON_CTR2	
Uncore C-box 3 PerfMon Counter 2		Package
Register Address: D79H, 3449	MSR_C3_PMON_CTR3	
Uncore C-box 3 PerfMon Counter 3		Package
Register Address: D84H, 3460	MSR_C4_PMON_BOX_CTL	
Uncore C-box 4 PerfMon Local Box Wide Control		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D90H, 3472	MSR_C4_PMON_EVNTSEL0	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 0		Package
Register Address: D91H, 3473	MSR_C4_PMON_EVNTSEL1	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 1		Package
Register Address: D92H, 3474	MSR_C4_PMON_EVNTSEL2	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 2		Package
Register Address: D93H, 3475	MSR_C4_PMON_EVNTSEL3	
Uncore C-box 4 PerfMon Event Select for C-box 4 Counter 3		Package
Register Address: D94H, 3476	MSR_C4_PMON_BOX_FILTER	
Uncore C-box 4 PerfMon Box Wide Filter		Package
Register Address: D96H, 3478	MSR_C4_PMON_CTR0	
Uncore C-box 4 PerfMon Counter 0		Package
Register Address: D97H, 3479	MSR_C4_PMON_CTR1	
Uncore C-box 4 PerfMon Counter 1		Package
Register Address: D98H, 3480	MSR_C4_PMON_CTR2	
Uncore C-box 4 PerfMon Counter 2		Package
Register Address: D99H, 3481	MSR_C4_PMON_CTR3	
Uncore C-box 4 PerfMon Counter 3		Package
Register Address: DA4H, 3492	MSR_C5_PMON_BOX_CTL	
Uncore C-box 5 PerfMon Local Box Wide Control		Package
Register Address: DB0H, 3504	MSR_C5_PMON_EVNTSEL0	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 0		Package
Register Address: DB1H, 3505	MSR_C5_PMON_EVNTSEL1	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 1		Package
Register Address: DB2H, 3506	MSR_C5_PMON_EVNTSEL2	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 2		Package
Register Address: DB3H, 3507	MSR_C5_PMON_EVNTSEL3	
Uncore C-box 5 PerfMon Event Select for C-box 5 Counter 3		Package
Register Address: DB4H, 3508	MSR_C5_PMON_BOX_FILTER	
Uncore C-box 5 PerfMon Box Wide Filter		Package
Register Address: DB6H, 3510	MSR_C5_PMON_CTR0	
Uncore C-box 5 PerfMon Counter 0		Package
Register Address: DB7H, 3511	MSR_C5_PMON_CTR1	
Uncore C-box 5 PerfMon Counter 1		Package
Register Address: DB8H, 3512	MSR_C5_PMON_CTR2	
Uncore C-box 5 PerfMon Counter 2		Package
Register Address: DB9H, 3513	MSR_C5_PMON_CTR3	

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-box 5 PerfMon Counter 3		Package
Register Address: DC4H, 3524	MSR_C6_PMON_BOX_CTL	
Uncore C-box 6 PerfMon Local Box Wide Control		Package
Register Address: DDOH, 3536	MSR_C6_PMON_EVNTSEL0	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 0		Package
Register Address: DD1H, 3537	MSR_C6_PMON_EVNTSEL1	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 1		Package
Register Address: DD2H, 3538	MSR_C6_PMON_EVNTSEL2	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 2		Package
Register Address: DD3H, 3539	MSR_C6_PMON_EVNTSEL3	
Uncore C-box 6 PerfMon Event Select for C-box 6 Counter 3		Package
Register Address: DD4H, 3540	MSR_C6_PMON_BOX_FILTER	
Uncore C-box 6 PerfMon Box Wide Filter		Package
Register Address: DD6H, 3542	MSR_C6_PMON_CTRL0	
Uncore C-box 6 PerfMon Counter 0		Package
Register Address: DD7H, 3543	MSR_C6_PMON_CTRL1	
Uncore C-box 6 PerfMon Counter 1		Package
Register Address: DD8H, 3544	MSR_C6_PMON_CTRL2	
Uncore C-box 6 PerfMon Counter 2		Package
Register Address: DD9H, 3545	MSR_C6_PMON_CTRL3	
Uncore C-box 6 PerfMon Counter 3		Package
Register Address: DE4H, 3556	MSR_C7_PMON_BOX_CTL	
Uncore C-box 7 PerfMon Local Box Wide Control		Package
Register Address: DFOH, 3568	MSR_C7_PMON_EVNTSEL0	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 0		Package
Register Address: DF1H, 3569	MSR_C7_PMON_EVNTSEL1	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 1		Package
Register Address: DF2H, 3570	MSR_C7_PMON_EVNTSEL2	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 2		Package
Register Address: DF3H, 3571	MSR_C7_PMON_EVNTSEL3	
Uncore C-box 7 PerfMon Event Select for C-box 7 Counter 3		Package
Register Address: DF4H, 3572	MSR_C7_PMON_BOX_FILTER	
Uncore C-box 7 PerfMon Box Wide Filter		Package
Register Address: DF6H, 3574	MSR_C7_PMON_CTRL0	
Uncore C-box 7 PerfMon Counter 0		Package
Register Address: DF7H, 3575	MSR_C7_PMON_CTRL1	
Uncore C-box 7 PerfMon Counter 1		Package

Table 2-24. Uncore PMU MSRs in Intel® Xeon® Processor E5 Family (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DF8H, 3576	MSR_C7_PMON_CTR2	
Uncore C-box 7 PerfMon Counter 2		Package
Register Address: DF9H, 3577	MSR_C7_PMON_CTR3	
Uncore C-box 7 PerfMon Counter 3		Package

2.12 MSRS IN THE 3RD GENERATION INTEL® CORE™ PROCESSOR FAMILY BASED ON IVY BRIDGE MICROARCHITECTURE

The 3rd generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v2 product family based on Ivy Bridge microarchitecture support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-25. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates that TDP Limit for Turbo mode is not programmable.	Package
31:30	Reserved.	
32	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.	Package
34:33	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved	Package
39:35	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org .	Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
24:16	Reserved	
25	C3 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Enable C3 Undemotion (R/W) When set, enables undemotion from demoted C3.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
28	Enable C1 Undemotion (R/W) When set, enables undemotion from demoted C1.	
63:29	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PP0 Energy Status (R/O) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
	ConfigTDP Level 1 ratio and power level (R/O)	Package
14:0	PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.	
15	Reserved.	
23:16	Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.	
47	Reserved.	
62:48	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O)		Package
14:0	PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.	
15	Reserved.	
23:16	Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.	
47	Reserved.	

Table 2-25. Additional MSRs Supported by 3rd Generation Intel® Core™ Processors Based on Ivy Bridge Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
62:48	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
1:0	TDP_LEVEL (RW/L) System BIOS can program this field.	
30:2	Reserved.	
31	Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
See Table 2-20, Table 2-21, and Table 2-22 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.		

2.12.1 MSRs in the Intel® Xeon® Processor E5 v2 Product Family Based on Ivy Bridge-E Microarchitecture

Table 2-26 lists model-specific registers (MSRs) that are specific to the Intel® Xeon® Processor E5 v2 Product Family (based on Ivy Bridge-E microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH; see Table 2-1. These processors supports the MSR interfaces listed in Table 2-20 and Table 2-26.

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number	Enable Control (R/W)	Package
0	LockOut (R/WO) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) When set to 1, indicates that Protected Processor Inventory Number (PPIN) capability can be enabled for a privileged system inventory agent to read PPIN from MSR_PPIN. When set to 0, PPIN capability is not supported. An attempt to access MSR_PPIN_CTL or MSR_PPIN will cause #GP.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
30	Programmable TJ OFFSET (R/O) When set to 1, indicates that MSR_TEMPERATURE_TARGET.[27:24] is valid and writable to specify a temperature offset.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.	
9:3	Reserved.	
10	I/O MWait Redirection Enable (R/W) When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWait instructions.	
14:11	Reserved.	
15	CFG Lock (R/WO) When set, locks bits 15:0 of this register until next reset.	
63:16	Reserved.	
Register Address: 179H, 377		
IA32_MCG_CAP		
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	Reserved.	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17FH, 383		
MSR_ERROR_CONTROL		
MC Bank Error Configuration (R/W)		Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
27:24	TCC Activation Offset (R/W) Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO.[30] is set.	
63:28	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0. R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
63:32	Reserved.	
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 296H, 662	IA32_MC22_CTL2	
See Table 2-2.		Package
Register Address: 297H, 663	IA32_MC23_CTL2IA32_MC23_CTL2	
See Table 2-2.		Package
Register Address: 298H, 664	IA32_MC24_CTL2	
See Table 2-2.		Package
Register Address: 299H, 665	IA32_MC25_CTL2	
See Table 2-2.		Package
Register Address: 29AH, 666	IA32_MC26_CTL2	
See Table 2-2.		Package
Register Address: 29BH, 667	IA32_MC27_CTL2	
See Table 2-2.		Package
Register Address: 29CH, 668	IA32_MC28_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC7 and MC 8 report MC errors from the two home agents.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC11 reports MC errors from a specific channel of the integrated memory controller.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs." Bank MC20 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 458H, 1112	IA32_MC22_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 459H, 1113	IA32_MC22_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 45AH, 1114	IA32_MC22_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 45BH, 1115	IA32_MC22_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC22 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 45CH, 1116	IA32_MC23_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 45DH, 1117	IA32_MC23_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 45EH, 1118	IA32_MC23_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 45FH, 1119	IA32_MC23_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC23 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 460H, 1120	IA32_MC24_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 461H, 1121	IA32_MC24_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 462H, 1122	IA32_MC24_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	
Register Address: 463H, 1123	IA32_MC24_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC24 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.	Package	

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 464H, 1124	IA32_MC25_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 465H, 1125	IA32_MC25_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 466H, 1126	IA32_MC25_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 467H, 1127	IA32_MC2MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC25 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 468H, 1128	IA32_MC26_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 469H, 1129	IA32_MC26_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 46AH, 1130	IA32_MC26_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 46BH, 1131	IA32_MC26_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC26 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 46CH, 1132	IA32_MC27_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 46DH, 1133	IA32_MC27_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 46EH, 1134	IA32_MC27_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 46FH, 1135	IA32_MC27_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC27 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 470H, 1136	IA32_MC28_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-26. MSRs Supported by the Intel® Xeon® Processor E5 v2 Product Family (Ivy Bridge-E Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 471H, 1137	IA32_MC28_STATUS	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 472H, 1138	IA32_MC28_ADDR	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 473H, 1139	IA32_MC28_MISC	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC28 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
Package RAPL Perf Status (R/O)		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, for other MSR definitions applicable to Intel Xeon processor E5 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.		

2.12.2 Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family

The Intel® Xeon® processor E7 v2 family (based on Ivy Bridge-E microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH supports the MSR interfaces listed in Table 2-20, Table 2-26, and Table 2-27.

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
63:16	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
63:25	Reserved.	
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status (R/W)		Thread
0	RIPV	
1	EIPV	
2	MCIP	
3	LMCE Signaled	
63:4	Reserved.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
62:56	Reserved.	
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT and MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 29DH, 669	IA32_MC29_CTL2	
See Table 2-2.		Package
Register Address: 29EH, 670	IA32_MC30_CTL2	
See Table 2-2.		Package
Register Address: 29FH, 671	IA32_MC31_CTL2	
See Table 2-2.		Package
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Section 21.3.1.1.1, "Processor Event Based Sampling (PEBS)."		Thread
$n:0$	Enable PEBS on IA32_PMCx. (R/W)	
$31:n+1$	Reserved.	
$32+m:32$	Enable Load Latency on IA32_PMCx. (R/W)	
$63:33+m$	Reserved.	
Register Address: 41BH, 1051	IA32_MC6_MISC	
Misc MAC Information of Integrated I/O (R/O) See Section 17.3.2.4.		Package
5:0	Recoverable Address LSB	
8:6	Address Mode	
15:9	Reserved.	
31:16	PCI Express Requestor ID	
39:32	PCI Express Segment Number	
63:32	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package

Table 2-27. Additional MSRs Supported by the Intel® Xeon® Processor E7 v2 Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_3EH (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 476H, 1142	IA32_MC29_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC29 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC30 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
Register Address: 47FH, 1147	IA32_MC31_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC31 reports MC errors from a specific CBo (core broadcast) and its corresponding slice of L3.		Package
See Table 2-20, Table 2-26 for other MSR definitions applicable to Intel Xeon processor E7 v2 with a CPUID Signature DisplayFamily_DisplayModel value of 06_3AH.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.12.3 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Intel Xeon Processor E5 v2 and E7 v2 families are based on the Ivy Bridge-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-24 and Table 2-28. For complete detail of the uncore PMU, refer to Intel

Xeon Processor E5 v2 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3EH.

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C00H, 3072	MSR_PMON_GLOBAL_CTL	
Uncore PerfMon Per-Socket Global Control		Package
Register Address: C01H, 3073	MSR_PMON_GLOBAL_STATUS	
Uncore PerfMon Per-Socket Global Status		Package
Register Address: C06H, 3078	MSR_PMON_GLOBAL_CONFIG	
Uncore PerfMon Per-Socket Global Configuration		Package
Register Address: C15H, 3093	MSR_U_PMON_BOX_STATUS	
Uncore U-box PerfMon U-Box Wide Status		Package
Register Address: C35H, 3125	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU PerfMon Box Wide Status		Package
Register Address: D1AH, 3354	MSR_C0_PMON_BOX_FILTER1	
Uncore C-Box 0 PerfMon Box Wide Filter1		Package
Register Address: D3AH, 3386	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 PerfMon Box Wide Filter1		Package
Register Address: D5AH, 3418	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 PerfMon Box Wide Filter1		Package
Register Address: D7AH, 3450	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 PerfMon Box Wide Filter1		Package
Register Address: D9AH, 3482	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 PerfMon Box Wide Filter1		Package
Register Address: DBAH, 3514	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 PerfMon Box Wide Filter1		Package
Register Address: DDAH, 3546	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 PerfMon Box Wide Filter1		Package
Register Address: DFAH, 3578	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 PerfMon Box Wide Filter1		Package
Register Address: E04H, 3588	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 PerfMon Local Box Wide Control		Package
Register Address: E10H, 3600	MSR_C8_PMON_EVNTSELO	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 0		Package
Register Address: E11H, 3601	MSR_C8_PMON_EVNTSEL1	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 1		Package
Register Address: E12H, 3602	MSR_C8_PMON_EVNTSEL2	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 2		Package
Register Address: E13H, 3603	MSR_C8_PMON_EVNTSEL3	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 3		Package
Register Address: E14H, 3604	MSR_C8_PMON_BOX_FILTER	
Uncore C-Box 8 PerfMon Box Wide Filter		Package
Register Address: E16H, 3606	MSR_C8_PMON_CTR0	
Uncore C-Box 8 PerfMon Counter 0		Package
Register Address: E17H, 3607	MSR_C8_PMON_CTR1	
Uncore C-Box 8 PerfMon Counter 1		Package
Register Address: E18H, 3608	MSR_C8_PMON_CTR2	
Uncore C-Box 8 PerfMon Counter 2		Package
Register Address: E19H, 3609	MSR_C8_PMON_CTR3	
Uncore C-Box 8 PerfMon Counter 3		Package
Register Address: E1AH, 3610	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 PerfMon Box Wide Filter1		Package
Register Address: E24H, 3620	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 PerfMon Local Box Wide Control		Package
Register Address: E30H, 3632	MSR_C9_PMON_EVNTSELO	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 0		Package
Register Address: E31H, 3633	MSR_C9_PMON_EVNTSEL1	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 1		Package
Register Address: E32H, 3634	MSR_C9_PMON_EVNTSEL2	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 2		Package
Register Address: E33H, 3635	MSR_C9_PMON_EVNTSEL3	
Uncore C-Box 9 PerfMon Event Select for C-box 9 Counter 3		Package
Register Address: E34H, 3636	MSR_C9_PMON_BOX_FILTER	
Uncore C-Box 9 PerfMon Box Wide Filter		Package
Register Address: E36H, 3638	MSR_C9_PMON_CTR0	
Uncore C-Box 9 PerfMon Counter 0		Package
Register Address: E37H, 3639	MSR_C9_PMON_CTR1	
Uncore C-Box 9 PerfMon Counter 1		Package
Register Address: E38H, 3640	MSR_C9_PMON_CTR2	
Uncore C-Box 9 PerfMon Counter 2		Package
Register Address: E39H, 3641	MSR_C9_PMON_CTR3	
Uncore C-Box 9 PerfMon Counter 3		Package
Register Address: E3AH, 3642	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 PerfMon Box Wide Filter1		Package
Register Address: E44H, 3652	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 PerfMon Local Box Wide Control		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E50H, 3664	MSR_C10_PMON_EVNTSELO	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 0		Package
Register Address: E51H, 3665	MSR_C10_PMON_EVNTSEL1	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 1		Package
Register Address: E52H, 3666	MSR_C10_PMON_EVNTSEL2	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 2		Package
Register Address: E53H, 3667	MSR_C10_PMON_EVNTSEL3	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 3		Package
Register Address: E54H, 3668	MSR_C10_PMON_BOX_FILTER	
Uncore C-Box 10 PerfMon Box Wide Filter		Package
Register Address: E56H, 3670	MSR_C10_PMON_CTR0	
Uncore C-Box 10 PerfMon Counter 0		Package
Register Address: E57H, 3671	MSR_C10_PMON_CTR1	
Uncore C-Box 10 PerfMon Counter 1		Package
Register Address: E58H, 3672	MSR_C10_PMON_CTR2	
Uncore C-Box 10 PerfMon Counter 2		Package
Register Address: E59H, 3673	MSR_C10_PMON_CTR3	
Uncore C-Box 10 PerfMon Counter 3		Package
Register Address: E5AH, 3674	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 PerfMon Box Wide Filter1		Package
Register Address: E64H, 3684	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 PerfMon Local Box Wide Control		Package
Register Address: E70H, 3696	MSR_C11_PMON_EVNTSELO	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 0		Package
Register Address: E71H, 3697	MSR_C11_PMON_EVNTSEL1	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 1		Package
Register Address: E72H, 3698	MSR_C11_PMON_EVNTSEL2	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 2		Package
Register Address: E73H, 3699	MSR_C11_PMON_EVNTSEL3	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 3		Package
Register Address: E74H, 3700	MSR_C11_PMON_BOX_FILTER	
Uncore C-Box 11 PerfMon Box Wide Filter		Package
Register Address: E76H, 3702	MSR_C11_PMON_CTR0	
Uncore C-Box 11 PerfMon Counter 0		Package
Register Address: E77H, 3703	MSR_C11_PMON_CTR1	
Uncore C-Box 11 PerfMon Counter 1		Package
Register Address: E78H, 3704	MSR_C11_PMON_CTR2	

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 11 PerfMon Counter 2		Package
Register Address: E79H, 3705	MSR_C11_PMON_CTR3	
Uncore C-Box 11 PerfMon Counter 3		Package
Register Address: E7AH, 3706	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 PerfMon Box Wide Filter1		Package
Register Address: E84H, 3716	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 PerfMon Local Box Wide Control		Package
Register Address: E90H, 3728	MSR_C12_PMON_EVNTSELO	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 0		Package
Register Address: E91H, 3729	MSR_C12_PMON_EVNTSEL1	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 1		Package
Register Address: E92H, 3730	MSR_C12_PMON_EVNTSEL2	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 2		Package
Register Address: E93H, 3731	MSR_C12_PMON_EVNTSEL3	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 3		Package
Register Address: E94H, 3732	MSR_C12_PMON_BOX_FILTER	
Uncore C-Box 12 PerfMon Box Wide Filter		Package
Register Address: E96H, 3734	MSR_C12_PMON_CTR0	
Uncore C-Box 12 PerfMon Counter 0		Package
Register Address: E97H, 3735	MSR_C12_PMON_CTR1	
Uncore C-Box 12 PerfMon Counter 1		Package
Register Address: E98H, 3736	MSR_C12_PMON_CTR2	
Uncore C-Box 12 PerfMon Counter 2		Package
Register Address: E99H, 3737	MSR_C12_PMON_CTR3	
Uncore C-Box 12 PerfMon Counter 3		Package
Register Address: E9AH, 3738	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 PerfMon Box Wide Filter1		Package
Register Address: EA4H, 3748	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 PerfMon Local Box Wide Control		Package
Register Address: EB0H, 3760	MSR_C13_PMON_EVNTSELO	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 0		Package
Register Address: EB1H, 3761	MSR_C13_PMON_EVNTSEL1	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 1		Package
Register Address: EB2H, 3762	MSR_C13_PMON_EVNTSEL2	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 2		Package
Register Address: EB3H, 3763	MSR_C13_PMON_EVNTSEL3	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 3		Package

Table 2-28. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v2 and E7 v2 Families (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EB4H, 3764	MSR_C13_PMON_BOX_FILTER	
Uncore C-Box 13 PerfMon Box Wide Filter		Package
Register Address: EB6H, 3766	MSR_C13_PMON_CTR0	
Uncore C-Box 13 PerfMon Counter 0		Package
Register Address: EB7H, 3767	MSR_C13_PMON_CTR1	
Uncore C-Box 13 PerfMon Counter 1		Package
Register Address: EB8H, 3768	MSR_C13_PMON_CTR2	
Uncore C-Box 13 PerfMon Counter 2		Package
Register Address: EB9H, 3769	MSR_C13_PMON_CTR3	
Uncore C-Box 13 PerfMon Counter 3		Package
Register Address: EBAH, 3770	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 PerfMon Box Wide Filter1		Package
Register Address: EC4H, 3780	MSR_C14_PMON_BOX_CTL	
Uncore C-Box 14 PerfMon Local Box Wide Control		Package
Register Address: ED0H, 3792	MSR_C14_PMON_EVNTSELO	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 0		Package
Register Address: ED1H, 3793	MSR_C14_PMON_EVNTSEL1	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 1		Package
Register Address: ED2H, 3794	MSR_C14_PMON_EVNTSEL2	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 2		Package
Register Address: ED3H, 3795	MSR_C14_PMON_EVNTSEL3	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 3		Package
Register Address: ED4H, 3796	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 PerfMon Box Wide Filter		Package
Register Address: ED6H, 3798	MSR_C14_PMON_CTR0	
Uncore C-Box 14 PerfMon Counter 0		Package
Register Address: ED7H, 3799	MSR_C14_PMON_CTR1	
Uncore C-Box 14 PerfMon Counter 1		Package
Register Address: ED8H, 3800	MSR_C14_PMON_CTR2	
Uncore C-Box 14 PerfMon Counter 2		Package
Register Address: ED9H, 3801	MSR_C14_PMON_CTR3	
Uncore C-Box 14 PerfMon Counter 3		Package
Register Address: EDAH, 3802	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 PerfMon Box Wide Filter1		Package

2.13 MSRS IN THE 4TH GENERATION INTEL® CORE™ PROCESSORS BASED ON HASWELL MICROARCHITECTURE

The 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200v3 product family (based on Haswell microarchitecture), with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H, support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, and Table 2-29. For an MSR listed in Table 2-20 that also appears in Table 2-29, Table 2-29 supersedes Table 2-20.

The MSRs listed in Table 2-29 also apply to processors based on Haswell-E microarchitecture (see Section 2.14).

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
31:30	Reserved.	
32	Low Power Mode Support (LPM) (R/O) When set to 1, indicates that LPM is supported. When set to 0, indicates LPM is not supported.	Package
34:33	Number of ConfigTDP Levels (R/O) 00: Only Base TDP level available. 01: One additional TDP level available. 02: Two additional TDP level available. 03: Reserved.	Package
39:35	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
55:48	Minimum Operating Ratio (R/O) Contains the minimum supported operating ratio in units of 100 MHz.	Package
63:56	Reserved.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 186H, 390	IA32_PERFEVTSELO	
Performance Event Select for Counter 0 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 21.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
Performance Event Select for Counter 1 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 21.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 188H, 392	IA32_PERFEVTSEL2	
Performance Event Select for Counter 2 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 21.3.6.5.1. When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
33	IN_TXCP: See Section 21.3.6.5.1. When IN_TXCP=1 & IN_TX=1 and in sampling, a spurious PMI may occur and transactions may continuously abort near overflow conditions. Software should favor using IN_TXCP for counting over sampling. If sampling, software should use large “sample-after” value after clearing the counter configured to use IN_TXCP and also always reset the counter even when no overflow condition was reported.	
Register Address: 189H, 393	IA32_PERFEVTSEL3	
Performance Event Select for Counter 3 (R/W) Supports all fields described in Table 2-2 and the fields below.		Thread
32	IN_TX: See Section 21.3.6.5.1 When IN_TX (bit 32) is set, AnyThread (bit 21) should be cleared to prevent incorrect results.	
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W)		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
8	FAR_BRANCH	
9	EN_CALL_STACK	
63:9	Reserved.	
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) See Table 2-2.		Thread
0	LBR: Last Branch Record	
1	BTF	
5:2	Reserved.	
6	TR: Branch Trace	
7	BTS: Log Branch Trace Message to BTS Buffer	
8	BTINT	
9	BTS_OFF_OS	
10	BTS_OFF_USER	
11	FREEZE_LBR_ON_PMI	
12	FREEZE_PERFMON_ON_PMI	
13	ENABLE_UNCORE_PMI	
14	FREEZE_WHILE_SMM	
15	RTM_DEBUG	
63:15	Reserved.	
Register Address: 491H, 1169	IA32_VMX_VMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Thread
Register Address: 60BH, 1548	MSR_PKG_C6_C7_INTERRUPT_RESPONSE_LIMIT_1	
Package C6/C7 Interrupt Response Limit 1 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the shorter-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt Response Time Limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:16	Reserved.	
Register Address: 60CH, 1548	MSR_PKG_C_IRT_L2	
Package C6/C7 Interrupt Response Limit 2 (R/W) This MSR defines the interrupt response time limit used by the processor to manage a transition to a package C6 or C7 state. The latency programmed in this register is for the longer-latency sub C-states used by an MWAIT hint to a C6 or C7 state. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
9:0	Interrupt response time limit (R/W) Specifies the limit that should be used to decide if the package should be put into a package C6 or C7 state.	
12:10	Time Unit (R/W) Specifies the encoding value of time unit of the interrupt response time limit. See Table 2-20 for supported time unit encodings.	
14:13	Reserved.	
15	Valid (R/W) Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-state management.	
63:16	Reserved.	
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O)		Package
7:0	Config_TDP_Base Base TDP level ratio to be used for this specific processor (in units of 100 MHz).	
63:8	Reserved.	
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 Ratio and Power Level (R/O)		Package
14:0	PKG_TDP_LVL1 Power setting for ConfigTDP Level 1.	
15	Reserved.	
23:16	Config_TDP_LVL1_Ratio ConfigTDP level 1 ratio to be used for this specific processor.	

Table 2-29. Additional MSRs Supported by Processors Based on the Haswell and Haswell-E Microarchitectures

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL1 Max Power setting allowed for ConfigTDP Level 1.	
62:47	PKG_MIN_PWR_LVL1 MIN Power setting allowed for ConfigTDP Level 1.	
63	Reserved.	
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 Ratio and Power Level (R/O)		Package
14:0	PKG_TDP_LVL2 Power setting for ConfigTDP Level 2.	
15	Reserved.	
23:16	Config_TDP_LVL2_Ratio ConfigTDP level 2 ratio to be used for this specific processor.	
31:24	Reserved.	
46:32	PKG_MAX_PWR_LVL2 Max Power setting allowed for ConfigTDP Level 2.	
62:47	PKG_MIN_PWR_LVL2 MIN Power setting allowed for ConfigTDP Level 2.	
63	Reserved.	
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W)		Package
1:0	TDP_LEVEL (RW/L) System BIOS can program this field.	
30:2	Reserved.	
31	Config_TDP_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W)		Package
7:0	MAX_NON_TURBO_RATIO (RW/L) System BIOS can program this field.	
30:8	Reserved.	
31	TURBO_ACTIVATION_RATIO_Lock (RW/L) When this bit is set, the content of this register is locked until a reset.	
63:32	Reserved.	
Register Address: C80H, 3200	IA32_DEBUG_INTERFACE	
Silicon Debug Feature Control (R/W) See Table 2-2.		Package

2.13.1 MSRs in the 4th Generation Intel® Core™ Processor Family Based on Haswell Microarchitecture

Table 2-30 lists model-specific registers (MSRs) that are specific to the 4th generation Intel® Core™ processor family and the Intel® Xeon® processor E3-1200 v3 product family (based on Haswell microarchitecture). These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH, 06_45H, or 06_46H; see Table 2-1.

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s Package C states C7 are not available to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3CH.	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and the MSR_SMM_FEATURE_CONTROL is supported.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and the MSR_SMM_DELAYED is supported.	
63:60	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 391H, 913	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Core 0 select.	
1	Core 1 select.	
2	Core 2 select.	
3	Core 3 select.	
18:4	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 392H, 914	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Encoded number of C-Box, derive value by "-1".	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (SMM-RW) Reports SMM capability Enhancement. Accessible only while in SMM.		Package
0	Lock (SMM-RW) When set to '1' locks this register from further changes.	
1	Reserved.	
2	SMM_Code_Chk_En (SMM-RW) This control bit is available only if MSR_SMM_MCA_CAP[58] == 1. When set to '0' (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set to '1' any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 4E2H, 1250	MSR_SMM_DELAYED	
SMM Delayed (SMM-RO) Reports the interruptible state of all logical processors in the package. Available only while in SMM and MSR_SMM_MCA_CAP[LONG_FLOW_INDICATION] == 1.		Package

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its state in a long flow of internal operation which delays servicing an interrupt. The corresponding bit will be set at the start of long events such as: Microcode Update Load, C6, WBINVD, Ratio Change, Throttle. The bit is automatically cleared at the end of each long event. The reset value of this field is 0. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 4E3H, 1251	MSR_SMM_BLOCKED	
SMM Blocked (SMM-RO) Reports the blocked state of all logical processors in the package. Available only while in SMM.		Package
N-1:0	LOG_PROC_STATE (SMM-RO) Each bit represents a logical processor of its blocked state to service an SMI. The corresponding bit will be set if the logical processor is in one of the following states: Wait For SIPI or SENTER Sleep. The reset value of this field is OFFFH. Only bit positions below N = CPUID.0BH.PKG_LVL:EBX[15:0] can be updated.	
63:N	Reserved.	
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 16.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 16.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 640H, 1600	MSR_PP1_POWER_LIMIT	
PP1 RAPL Power Limit Control (R/W) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 641H, 1601	MSR_PP1_ENERGY_STATUS	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
PP1 Energy Status (R/O) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 642H, 1602	MSR_PP1_POLICY	
PP1 Balance Policy (R/W) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Core Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
12	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Graphics Driver Status (R0) When set, frequency is reduced below the operating system request due to Processor Graphics driver override.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Graphics Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to domain-level power limiting.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
15:12	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
5:2	Reserved.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Package-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL1.	
11	Package-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package-level power limiting PL2.	
15:12	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Graphics Driver Log When set, indicates that the Graphics Driver Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the Autonomous Utilization-Based Frequency Control Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Core Power Limiting Log When set, indicates that the Core Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
26	Package-Level PL1 Power Limiting Log When set, indicates that the Package Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package-Level PL2 Power Limiting Log When set, indicates that the Package Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-30. MSRs Supported by 4th Generation Intel® Core™ Processors (Haswell Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:30	Reserved.	
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1824	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTRO	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 063CH or 06_46H.		

2.13.2 Additional Residency MSRs Supported in 4th Generation Intel® Core™ Processors

The 4th generation Intel® Core™ processor family (based on Haswell microarchitecture) with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-30, and Table 2-31.

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
63:29	Reserved.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C8 Residency Counter (R/O) Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	

Table 2-31. Additional Residency MSRs Supported by 4th Generation Intel® Core™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_45H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C9 Residency Counter (R/O) Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C10 Residency Counter (R/O) Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
See Table 2-20, Table 2-21, Table 2-22, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.		

2.14 MSRS IN THE INTEL® XEON® PROCESSOR E5 V3 AND E7 V3 PRODUCT FAMILY

The Intel® Xeon® processor E5 v3 family and the Intel® Xeon® processor E7 v3 family are based on Haswell-E microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_3F). These processors support the MSR interfaces listed in Table 2-20, Table 2-29, and Table 2-32.

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 35H, 53	MSR_CORE_THREAD_COUNT	
Configured State of Enabled Processor Core Count and Logical Processor Count (R/O) - After a Power-On RESET, enumerates factory configuration of the number of processor cores and logical processors in the physical package. - Following the sequence of (i) BIOS modified a Configuration Mask which selects a subset of processor cores to be active post RESET and (ii) a RESET event after the modification, enumerates the current configuration of enabled processor core count and logical processor count in the physical package.		Package
15:0	THREAD_COUNT (R/O) The number of logical processors that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.	
31:16	Core_COUNT (R/O) The number of processor cores that are currently enabled (by either factory configuration or BIOS configuration) in the physical package.	
63:32	Reserved.	
Register Address: 53H, 83	MSR_THREAD_ID_INFO	
A Hardware Assigned ID for the Logical Processor (R/O)		Thread

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
7:0	Logical_Processor_ID (R/O) An implementation-specific numerical value physically assigned to each logical processor. This ID is not related to Initial APIC ID or x2APIC ID, it is unique within a physical package.	
63:8	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 17FH, 383	MSR_ERROR_CONTROL	
MC Bank Error Configuration (R/W)		Package
0	Reserved.	
1	MemError Log Enable (R/W) When set, enables IMC status bank to log additional info in bits 36:32.	
63:2	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5 core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6 core active.	Package
55:48	Maximum Ratio Limit for 7C Maximum turbo ratio limit of 7 core active.	Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:56	Maximum Ratio Limit for 8C Maximum turbo ratio limit of 8 core active.	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C Maximum turbo ratio limit of 9 core active.	Package
15:8	Maximum Ratio Limit for 10C Maximum turbo ratio limit of 10 core active.	Package
23:16	Maximum Ratio Limit for 11C Maximum turbo ratio limit of 11 core active.	Package
31:24	Maximum Ratio Limit for 12C Maximum turbo ratio limit of 12 core active.	Package
39:32	Maximum Ratio Limit for 13C Maximum turbo ratio limit of 13 core active.	Package
47:40	Maximum Ratio Limit for 14C Maximum turbo ratio limit of 14 core active.	Package
55:48	Maximum Ratio Limit for 15C Maximum turbo ratio limit of 15 core active.	Package
63:56	Maximum Ratio Limit for 16C Maximum turbo ratio limit of 16 core active.	Package
Register Address: 1AFH, 431	MSR_TURBO_RATIO_LIMIT2	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 17C Maximum turbo ratio limit of 17 core active.	Package
15:8	Maximum Ratio Limit for 18C Maximum turbo ratio limit of 18 core active.	Package
62:16	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.	Package	
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 16.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 16.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy Consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61EH, 1566	MSR_PCIE_PLL_RATIO	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Configuration of PCIE PLL Relative to BCLK(R/W)		Package
1:0	PCIE Ratio (R/W) 00b: Use 5:5 mapping for 100MHz operation (default). 01b: Use 5:4 mapping for 125MHz operation. 10b: Use 5:3 mapping for 166MHz operation. 11b: Use 5:2 mapping for 250MHz operation.	Package
2	LPLL Select (R/W) If 1, use configured setting of PCIE Ratio.	Package
3	LONG RESET (R/W) If 1, wait an additional time-out before re-locking Gen2/Gen3 PLLs.	Package
63:4	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit	
3	Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit	
4	Reserved.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.	
12:11	Reserved.	
13	Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.	
14	Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.	
15	Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20	Reserved.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28:27	Reserved.	
29	Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
31	Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:32	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x0: No monitoring. 0x1: L3 occupancy monitoring. All other encoding reserved.	
31:8	Reserved.	
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8EH, 3214	IA32_QM_CTR	
Monitoring Counter Register (R/O) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread

Table 2-32. Additional MSRs Supported by the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
61:0	Resource Monitored Data	
62	Unavailable: If 1, indicates data for this RMID is not available or not monitored for this resource or RMID.	
63	Error: If 1, indicates an unsupported RMID or event type was written to IA32_PQR_QM_EVTSEL.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
63: 10	Reserved.	
See Table 2-20 and Table 2-29 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.14.1 Additional Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

The Intel Xeon Processor E5 v3 and E7 v3 families are based on Haswell-E microarchitecture. The MSR-based uncore PMU interfaces are listed in Table 2-33. For complete details of the uncore PMU, refer to the Intel Xeon Processor E5 v3 Product Family Uncore Performance Monitoring Guide. These processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3FH.

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 700H, 1792	MSR_PMON_GLOBAL_CTL	
Uncore PerfMon Per-Socket Global Control		Package
Register Address: 701H, 1793	MSR_PMON_GLOBAL_STATUS	
Uncore PerfMon Per-Socket Global Status		Package
Register Address: 702H, 1794	MSR_PMON_GLOBAL_CONFIG	
Uncore PerfMon Per-Socket Global Configuration		Package
Register Address: 703H, 1795	MSR_U_PMON_UCLK_FIXED_CTL	
Uncore U-Box UCLK Fixed Counter Control		Package
Register Address: 704H, 1796	MSR_U_PMON_UCLK_FIXED_CTR	
Uncore U-Box UCLK Fixed Counter		Package
Register Address: 705H, 1797	MSR_U_PMON_EVNTSELO	
Uncore U-Box PerfMon Event Select for U-Box Counter 0		Package
Register Address: 706H, 1798	MSR_U_PMON_EVNTSEL1	
Uncore U-Box PerfMon Event Select for U-Box Counter 1		Package
Register Address: 708H, 1800	MSR_U_PMON_BOX_STATUS	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore U-Box PerfMon U-Box Wide Status		Package
Register Address: 709H, 1801	MSR_U_PMON_CTRL0	
Uncore U-Box PerfMon Counter 0		Package
Register Address: 70AH, 1802	MSR_U_PMON_CTRL1	
Uncore U-Box PerfMon Counter 1		Package
Register Address: 710H, 1808	MSR_PCU_PMON_BOX_CTL	
Uncore PCU PerfMon for PCU-Box-Wide Control		Package
Register Address: 711H, 1809	MSR_PCU_PMON_EVTSEL0	
Uncore PCU PerfMon Event Select for PCU Counter 0		Package
Register Address: 712H, 1810	MSR_PCU_PMON_EVTSEL1	
Uncore PCU PerfMon Event Select for PCU Counter 1		Package
Register Address: 713H, 1811	MSR_PCU_PMON_EVTSEL2	
Uncore PCU PerfMon Event Select for PCU Counter 2		Package
Register Address: 714H, 1812	MSR_PCU_PMON_EVTSEL3	
Uncore PCU PerfMon Event Select for PCU Counter 3		Package
Register Address: 715H, 1813	MSR_PCU_PMON_BOX_FILTER	
Uncore PCU PerfMon Box-Wide Filter		Package
Register Address: 716H, 1814	MSR_PCU_PMON_BOX_STATUS	
Uncore PCU PerfMon Box Wide Status		Package
Register Address: 717H, 1815	MSR_PCU_PMON_CTRL0	
Uncore PCU PerfMon Counter 0		Package
Register Address: 718H, 1816	MSR_PCU_PMON_CTRL1	
Uncore PCU PerfMon Counter 1		Package
Register Address: 719H, 1817	MSR_PCU_PMON_CTRL2	
Uncore PCU PerfMon Counter 2		Package
Register Address: 71AH, 1818	MSR_PCU_PMON_CTRL3	
Uncore PCU PerfMon Counter 3		Package
Register Address: 720H, 1824	MSR_SO_PMON_BOX_CTL	
Uncore SBo 0 PerfMon for SBo 0 Box-Wide Control		Package
Register Address: 721H, 1825	MSR_SO_PMON_EVTSEL0	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 0		Package
Register Address: 722H, 1826	MSR_SO_PMON_EVTSEL1	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 1		Package
Register Address: 723H, 1827	MSR_SO_PMON_EVTSEL2	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 2		Package
Register Address: 724H, 1828	MSR_SO_PMON_EVTSEL3	
Uncore SBo 0 PerfMon Event Select for SBo 0 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 725H, 1829	MSR_S0_PMON_BOX_FILTER	
Uncore SBo 0 PerfMon Box-Wide Filter		Package
Register Address: 726H, 1830	MSR_S0_PMON_CTR0	
Uncore SBo 0 PerfMon Counter 0		Package
Register Address: 727H, 1831	MSR_S0_PMON_CTR1	
Uncore SBo 0 PerfMon Counter 1		Package
Register Address: 728H, 1832	MSR_S0_PMON_CTR2	
Uncore SBo 0 PerfMon Counter 2		Package
Register Address: 729H, 1833	MSR_S0_PMON_CTR3	
Uncore SBo 0 PerfMon Counter 3		Package
Register Address: 72AH, 1834	MSR_S1_PMON_BOX_CTL	
Uncore SBo 1 PerfMon for SBo 1 Box-Wide Control		Package
Register Address: 72BH, 1835	MSR_S1_PMON_EVTNSELO	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 0		Package
Register Address: 72CH, 1836	MSR_S1_PMON_EVTNSEL1	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 1		Package
Register Address: 72DH, 1837	MSR_S1_PMON_EVTNSEL2	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 2		Package
Register Address: 72EH, 1838	MSR_S1_PMON_EVTNSEL3	
Uncore SBo 1 PerfMon Event Select for SBo 1 Counter 3		Package
Register Address: 72FH, 1839	MSR_S1_PMON_BOX_FILTER	
Uncore SBo 1 PerfMon Box-Wide Filter		Package
Register Address: 730H, 1840	MSR_S1_PMON_CTR0	
Uncore SBo 1 PerfMon Counter 0		Package
Register Address: 731H, 1841	MSR_S1_PMON_CTR1	
Uncore SBo 1 PerfMon Counter 1		Package
Register Address: 732H, 1842	MSR_S1_PMON_CTR2	
Uncore SBo 1 PerfMon Counter 2		Package
Register Address: 733H, 1843	MSR_S1_PMON_CTR3	
Uncore SBo 1 PerfMon Counter 3		Package
Register Address: 734H, 1844	MSR_S2_PMON_BOX_CTL	
Uncore SBo 2 PerfMon for SBo 2 Box-Wide Control		Package
Register Address: 735H, 1845	MSR_S2_PMON_EVTNSELO	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 0		Package
Register Address: 736H, 1846	MSR_S2_PMON_EVTNSEL1	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 1		Package
Register Address: 737H, 1847	MSR_S2_PMON_EVTNSEL2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 2		Package
Register Address: 738H, 1848	MSR_S2_PMON_EVTSEL3	
Uncore SBo 2 PerfMon Event Select for SBo 2 Counter 3		Package
Register Address: 739H, 1849	MSR_S2_PMON_BOX_FILTER	
Uncore SBo 2 PerfMon Box-Wide Filter		Package
Register Address: 73AH, 1850	MSR_S2_PMON_CTR0	
Uncore SBo 2 PerfMon Counter 0		Package
Register Address: 73BH, 1851	MSR_S2_PMON_CTR1	
Uncore SBo 2 PerfMon Counter 1		Package
Register Address: 73CH, 1852	MSR_S2_PMON_CTR2	
Uncore SBo 2 PerfMon Counter 2		Package
Register Address: 73DH, 1853	MSR_S2_PMON_CTR3	
Uncore SBo 2 PerfMon Counter 3		Package
Register Address: 73EH, 1854	MSR_S3_PMON_BOX_CTL	
Uncore SBo 3 PerfMon for SBo 3 Box-Wide Control		Package
Register Address: 73FH, 1855	MSR_S3_PMON_EVTSELO	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 0		Package
Register Address: 740H, 1856	MSR_S3_PMON_EVTSEL1	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 1		Package
Register Address: 741H, 1857	MSR_S3_PMON_EVTSEL2	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 2		Package
Register Address: 742H, 1858	MSR_S3_PMON_EVTSEL3	
Uncore SBo 3 PerfMon Event Select for SBo 3 Counter 3		Package
Register Address: 743H, 1859	MSR_S3_PMON_BOX_FILTER	
Uncore SBo 3 PerfMon Box-Wide Filter		Package
Register Address: 744H, 1860	MSR_S3_PMON_CTR0	
Uncore SBo 3 PerfMon Counter 0		Package
Register Address: 745H, 1861	MSR_S3_PMON_CTR1	
Uncore SBo 3 PerfMon Counter 1		Package
Register Address: 746H, 1862	MSR_S3_PMON_CTR2	
Uncore SBo 3 PerfMon Counter 2		Package
Register Address: 747H, 1863	MSR_S3_PMON_CTR3	
Uncore SBo 3 PerfMon Counter 3		Package
Register Address: E00H, 3584	MSR_CO_PMON_BOX_CTL	
Uncore C-Box 0 PerfMon for Box-Wide Control		Package
Register Address: E01H, 3585	MSR_CO_PMON_EVTSELO	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E02H, 3586	MSR_CO_PMON_EVTSEL1	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 1		Package
Register Address: E03H, 3587	MSR_CO_PMON_EVTSEL2	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 2		Package
Register Address: E04H, 3588	MSR_CO_PMON_EVTSEL3	
Uncore C-Box 0 PerfMon Event Select for C-Box 0 Counter 3		Package
Register Address: E05H, 3589	MSR_CO_PMON_BOX_FILTER0	
Uncore C-Box 0 PerfMon Box Wide Filter 0		Package
Register Address: E06H, 3590	MSR_CO_PMON_BOX_FILTER1	
Uncore C-Box 0 PerfMon Box Wide Filter 1		Package
Register Address: E07H, 3591	MSR_CO_PMON_BOX_STATUS	
Uncore C-Box 0 PerfMon Box Wide Status		Package
Register Address: E08H, 3592	MSR_CO_PMON_CTR0	
Uncore C-Box 0 PerfMon Counter 0		Package
Register Address: E09H, 3593	MSR_CO_PMON_CTR1	
Uncore C-Box 0 PerfMon Counter 1		Package
Register Address: E0AH, 3594	MSR_CO_PMON_CTR2	
Uncore C-Box 0 PerfMon Counter 2		Package
Register Address: E0BH, 3595	MSR_CO_PMON_CTR3	
Uncore C-Box 0 PerfMon Counter 3		Package
Register Address: E10H, 3600	MSR_C1_PMON_BOX_CTL	
Uncore C-Box 1 PerfMon for Box-Wide Control		Package
Register Address: E11H, 3601	MSR_C1_PMON_EVTSEL0	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 0		Package
Register Address: E12H, 3602	MSR_C1_PMON_EVTSEL1	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 1		Package
Register Address: E13H, 3603	MSR_C1_PMON_EVTSEL2	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 2		Package
Register Address: E14H, 3604	MSR_C1_PMON_EVTSEL3	
Uncore C-Box 1 PerfMon Event Select for C-Box 1 Counter 3		Package
Register Address: E15H, 3605	MSR_C1_PMON_BOX_FILTER0	
Uncore C-Box 1 PerfMon Box Wide Filter 0		Package
Register Address: E16H, 3606	MSR_C1_PMON_BOX_FILTER1	
Uncore C-Box 1 PerfMon Box Wide Filter1		Package
Register Address: E17H, 3607	MSR_C1_PMON_BOX_STATUS	
Uncore C-Box 1 PerfMon Box Wide Status		Package
Register Address: E18H, 3608	MSR_C1_PMON_CTR0	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 1 PerfMon Counter 0		Package
Register Address: E19H, 3609	MSR_C1_PMON_CTR1	
Uncore C-Box 1 PerfMon Counter 1		Package
Register Address: E1AH, 3610	MSR_C1_PMON_CTR2	
Uncore C-Box 1 PerfMon Counter 2		Package
Register Address: E1BH, 3611	MSR_C1_PMON_CTR3	
Uncore C-Box 1 PerfMon Counter 3		Package
Register Address: E20H, 3616	MSR_C2_PMON_BOX_CTL	
Uncore C-Box 2 PerfMon for Box-Wide Control		Package
Register Address: E21H, 3617	MSR_C2_PMON_EVTSELO	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 0		Package
Register Address: E22H, 3618	MSR_C2_PMON_EVTSEL1	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 1		Package
Register Address: E23H, 3619	MSR_C2_PMON_EVTSEL2	
Uncore C-Box 2 PerfMon Event Select for C-Box 2 Counter 2		Package
Register Address: E24H, 3620	MSR_C2_PMON_EVTSEL3	
Uncore C-Box 2 PerfMon Event select for C-Box 2 Counter 3		Package
Register Address: E25H, 3621	MSR_C2_PMON_BOX_FILTER0	
Uncore C-Box 2 PerfMon Box Wide Filter 0		Package
Register Address: E26H, 3622	MSR_C2_PMON_BOX_FILTER1	
Uncore C-Box 2 PerfMon Box Wide Filter1		Package
Register Address: E27H, 3623	MSR_C2_PMON_BOX_STATUS	
Uncore C-Box 2 PerfMon Box Wide Status		Package
Register Address: E28H, 3624	MSR_C2_PMON_CTR0	
Uncore C-Box 2 PerfMon Counter 0		Package
Register Address: E29H, 3625	MSR_C2_PMON_CTR1	
Uncore C-Box 2 PerfMon Counter 1		Package
Register Address: E2AH, 3626	MSR_C2_PMON_CTR2	
Uncore C-Box 2 PerfMon Counter 2		Package
Register Address: E2BH, 3627	MSR_C2_PMON_CTR3	
Uncore C-Box 2 PerfMon Counter 3		Package
Register Address: E30H, 3632	MSR_C3_PMON_BOX_CTL	
Uncore C-Box 3 PerfMon for Box-Wide Control		Package
Register Address: E31H, 3633	MSR_C3_PMON_EVTSELO	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 0		Package
Register Address: E32H, 3634	MSR_C3_PMON_EVTSEL1	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E33H, 3635	MSR_C3_PMON_EVTSEL2	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 2		Package
Register Address: E34H, 3636	MSR_C3_PMON_EVTSEL3	
Uncore C-Box 3 PerfMon Event Select for C-Box 3 Counter 3		Package
Register Address: E35H, 3637	MSR_C3_PMON_BOX_FILTER0	
Uncore C-Box 3 PerfMon Box Wide Filter 0		Package
Register Address: E36H, 3638	MSR_C3_PMON_BOX_FILTER1	
Uncore C-Box 3 PerfMon Box Wide Filter1		Package
Register Address: E37H, 3639	MSR_C3_PMON_BOX_STATUS	
Uncore C-Box 3 PerfMon Box Wide Status		Package
Register Address: E38H, 3640	MSR_C3_PMON_CTR0	
Uncore C-Box 3 PerfMon Counter 0		Package
Register Address: E39H, 3641	MSR_C3_PMON_CTR1	
Uncore C-Box 3 PerfMon Counter 1		Package
Register Address: E3AH, 3642	MSR_C3_PMON_CTR2	
Uncore C-Box 3 PerfMon Counter 2		Package
Register Address: E3BH, 3643	MSR_C3_PMON_CTR3	
Uncore C-Box 3 PerfMon Counter 3		Package
Register Address: E40H, 3648	MSR_C4_PMON_BOX_CTL	
Uncore C-Box 4 PerfMon for Box-Wide Control		Package
Register Address: E41H, 3649	MSR_C4_PMON_EVTSELO	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 0		Package
Register Address: E42H, 3650	MSR_C4_PMON_EVTSEL1	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 1		Package
Register Address: E43H, 3651	MSR_C4_PMON_EVTSEL2	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 2		Package
Register Address: E44H, 3652	MSR_C4_PMON_EVTSEL3	
Uncore C-Box 4 PerfMon Event Select for C-Box 4 Counter 3		Package
Register Address: E45H, 3653	MSR_C4_PMON_BOX_FILTER0	
Uncore C-Box 4 PerfMon Box Wide Filter 0		Package
Register Address: E46H, 3654	MSR_C4_PMON_BOX_FILTER1	
Uncore C-Box 4 PerfMon Box Wide Filter1		Package
Register Address: E47H, 3655	MSR_C4_PMON_BOX_STATUS	
Uncore C-Box 4 PerfMon Box Wide Status		Package
Register Address: E48H, 3656	MSR_C4_PMON_CTR0	
Uncore C-Box 4 PerfMon Counter 0		Package
Register Address: E49H, 3657	MSR_C4_PMON_CTR1	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 4 PerfMon Counter 1		Package
Register Address: E4AH, 3658	MSR_C4_PMON_CTR2	
Uncore C-Box 4 PerfMon Counter 2		Package
Register Address: E4BH, 3659	MSR_C4_PMON_CTR3	
Uncore C-Box 4 PerfMon Counter 3		Package
Register Address: E50H, 3664	MSR_C5_PMON_BOX_CTL	
Uncore C-Box 5 PerfMon for Box-Wide Control		Package
Register Address: E51H, 3665	MSR_C5_PMON_EVNTSELO	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 0		Package
Register Address: E52H, 3666	MSR_C5_PMON_EVNTSEL1	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 1		Package
Register Address: E53H, 3667	MSR_C5_PMON_EVNTSEL2	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 2		Package
Register Address: E54H, 3668	MSR_C5_PMON_EVNTSEL3	
Uncore C-Box 5 PerfMon Event Select for C-Box 5 Counter 3		Package
Register Address: E55H, 3669	MSR_C5_PMON_BOX_FILTER0	
Uncore C-Box 5 PerfMon Box Wide Filter 0		Package
Register Address: E56H, 3670	MSR_C5_PMON_BOX_FILTER1	
Uncore C-Box 5 PerfMon Box Wide Filter 1		Package
Register Address: E57H, 3671	MSR_C5_PMON_BOX_STATUS	
Uncore C-Box 5 PerfMon Box Wide Status		Package
Register Address: E58H, 3672	MSR_C5_PMON_CTR0	
Uncore C-Box 5 PerfMon Counter 0		Package
Register Address: E59H, 3673	MSR_C5_PMON_CTR1	
Uncore C-Box 5 PerfMon Counter 1		Package
Register Address: E5AH, 3674	MSR_C5_PMON_CTR2	
Uncore C-Box 5 PerfMon Counter 2		Package
Register Address: E5BH, 3675	MSR_C5_PMON_CTR3	
Uncore C-Box 5 PerfMon Counter 3		Package
Register Address: E60H, 3680	MSR_C6_PMON_BOX_CTL	
Uncore C-Box 6 PerfMon for Box-Wide Control		Package
Register Address: E61H, 3681	MSR_C6_PMON_EVNTSELO	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 0		Package
Register Address: E62H, 3682	MSR_C6_PMON_EVNTSEL1	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 1		Package
Register Address: E63H, 3683	MSR_C6_PMON_EVNTSEL2	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 2		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E64H, 3684	MSR_C6_PMON_EVTSEL3	
Uncore C-Box 6 PerfMon Event Select for C-Box 6 Counter 3		Package
Register Address: E65H, 3685	MSR_C6_PMON_BOX_FILTER0	
Uncore C-Box 6 PerfMon Box Wide Filter 0		Package
Register Address: E66H, 3686	MSR_C6_PMON_BOX_FILTER1	
Uncore C-Box 6 PerfMon Box Wide Filter 1		Package
Register Address: E67H, 3687	MSR_C6_PMON_BOX_STATUS	
Uncore C-Box 6 PerfMon Box Wide Status		Package
Register Address: E68H, 3688	MSR_C6_PMON_CTR0	
Uncore C-Box 6 PerfMon Counter 0		Package
Register Address: E69H, 3689	MSR_C6_PMON_CTR1	
Uncore C-Box 6 PerfMon Counter 1		Package
Register Address: E6AH, 3690	MSR_C6_PMON_CTR2	
Uncore C-Box 6 PerfMon Counter 2		Package
Register Address: E6BH, 3691	MSR_C6_PMON_CTR3	
Uncore C-Box 6 PerfMon Counter 3		Package
Register Address: E70H, 3696	MSR_C7_PMON_BOX_CTL	
Uncore C-Box 7 PerfMon for Box-Wide Control		Package
Register Address: E71H, 3697	MSR_C7_PMON_EVTSELO	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 0		Package
Register Address: E72H, 3698	MSR_C7_PMON_EVTSEL1	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 1		Package
Register Address: E73H, 3699	MSR_C7_PMON_EVTSEL2	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 2		Package
Register Address: E74H, 3700	MSR_C7_PMON_EVTSEL3	
Uncore C-Box 7 PerfMon Event Select for C-Box 7 Counter 3		Package
Register Address: E75H, 3701	MSR_C7_PMON_BOX_FILTER0	
Uncore C-Box 7 PerfMon Box Wide Filter 0		Package
Register Address: E76H, 3702	MSR_C7_PMON_BOX_FILTER1	
Uncore C-Box 7 PerfMon Box Wide Filter 1		Package
Register Address: E77H, 3703	MSR_C7_PMON_BOX_STATUS	
Uncore C-Box 7 PerfMon Box Wide Status		Package
Register Address: E78H, 3704	MSR_C7_PMON_CTR0	
Uncore C-Box 7 PerfMon Counter 0		Package
Register Address: E79H, 3705	MSR_C7_PMON_CTR1	
Uncore C-Box 7 PerfMon Counter 1		Package
Register Address: E7AH, 3706	MSR_C7_PMON_CTR2	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 7 PerfMon Counter 2		Package
Register Address: E7BH, 3707	MSR_C7_PMON_CTR3	
Uncore C-Box 7 PerfMon Counter 3		Package
Register Address: E80H, 3712	MSR_C8_PMON_BOX_CTL	
Uncore C-Box 8 PerfMon Local Box Wide Control		Package
Register Address: E81H, 3713	MSR_C8_PMON_EVTNSELO	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 0		Package
Register Address: E82H, 3714	MSR_C8_PMON_EVTNSEL1	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 1		Package
Register Address: E83H, 3715	MSR_C8_PMON_EVTNSEL2	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 2		Package
Register Address: E84H, 3716	MSR_C8_PMON_EVTNSEL3	
Uncore C-Box 8 PerfMon Event Select for C-Box 8 Counter 3		Package
Register Address: E85H, 3717	MSR_C8_PMON_BOX_FILTER0	
Uncore C-Box 8 PerfMon Box Wide Filter 0		Package
Register Address: E86H, 3718	MSR_C8_PMON_BOX_FILTER1	
Uncore C-Box 8 PerfMon Box Wide Filter 1		Package
Register Address: E87H, 3719	MSR_C8_PMON_BOX_STATUS	
Uncore C-Box 8 PerfMon Box Wide Status		Package
Register Address: E88H, 3720	MSR_C8_PMON_CTR0	
Uncore C-Box 8 PerfMon Counter 0		Package
Register Address: E89H, 3721	MSR_C8_PMON_CTR1	
Uncore C-Box 8 PerfMon Counter 1		Package
Register Address: E8AH, 3722	MSR_C8_PMON_CTR2	
Uncore C-Box 8 PerfMon Counter 2		Package
Register Address: E8BH, 3723	MSR_C8_PMON_CTR3	
Uncore C-Box 8 PerfMon Counter 3		Package
Register Address: E90H, 3728	MSR_C9_PMON_BOX_CTL	
Uncore C-Box 9 PerfMon Local Box Wide Control		Package
Register Address: E91H, 3729	MSR_C9_PMON_EVTNSELO	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 0		Package
Register Address: E92H, 3730	MSR_C9_PMON_EVTNSEL1	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 1		Package
Register Address: E93H, 3731	MSR_C9_PMON_EVTNSEL2	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 2		Package
Register Address: E94H, 3732	MSR_C9_PMON_EVTNSEL3	
Uncore C-Box 9 PerfMon Event Select for C-Box 9 Counter 3		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E95H, 3733	MSR_C9_PMON_BOX_FILTER0	
Uncore C-Box 9 PerfMon Box Wide Filter 0		Package
Register Address: E96H, 3734	MSR_C9_PMON_BOX_FILTER1	
Uncore C-Box 9 PerfMon Box Wide Filter 1		Package
Register Address: E97H, 3735	MSR_C9_PMON_BOX_STATUS	
Uncore C-Box 9 PerfMon Box Wide Status		Package
Register Address: E98H, 3736	MSR_C9_PMON_CTR0	
Uncore C-Box 9 PerfMon Counter 0		Package
Register Address: E99H, 3737	MSR_C9_PMON_CTR1	
Uncore C-Box 9 PerfMon Counter 1		Package
Register Address: E9AH, 3738	MSR_C9_PMON_CTR2	
Uncore C-Box 9 PerfMon Counter 2		Package
Register Address: E9BH, 3739	MSR_C9_PMON_CTR3	
Uncore C-Box 9 PerfMon Counter 3		Package
Register Address: EA0H, 3744	MSR_C10_PMON_BOX_CTL	
Uncore C-Box 10 PerfMon Local Box Wide Control		Package
Register Address: EA1H, 3745	MSR_C10_PMON_EVNTSEL0	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 0		Package
Register Address: EA2H, 3746	MSR_C10_PMON_EVNTSEL1	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 1		Package
Register Address: EA3H, 3747	MSR_C10_PMON_EVNTSEL2	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 2		Package
Register Address: EA4H, 3748	MSR_C10_PMON_EVNTSEL3	
Uncore C-Box 10 PerfMon Event Select for C-Box 10 Counter 3		Package
Register Address: EA5H, 3749	MSR_C10_PMON_BOX_FILTER0	
Uncore C-Box 10 PerfMon Box Wide Filter 0		Package
Register Address: EA6H, 3750	MSR_C10_PMON_BOX_FILTER1	
Uncore C-Box 10 PerfMon Box Wide Filter 1		Package
Register Address: EA7H, 3751	MSR_C10_PMON_BOX_STATUS	
Uncore C-Box 10 PerfMon Box Wide Status		Package
Register Address: EA8H, 3752	MSR_C10_PMON_CTR0	
Uncore C-Box 10 PerfMon Counter 0		Package
Register Address: EA9H, 3753	MSR_C10_PMON_CTR1	
Uncore C-Box 10 PerfMon Counter 1		Package
Register Address: EAAH, 3754	MSR_C10_PMON_CTR2	
Uncore C-Box 10 PerfMon Counter 2		Package
Register Address: EABH, 3755	MSR_C10_PMON_CTR3	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 10 PerfMon Counter 3		Package
Register Address: EB0H, 3760	MSR_C11_PMON_BOX_CTL	
Uncore C-Box 11 PerfMon Local Box Wide Control		Package
Register Address: EB1H, 3761	MSR_C11_PMON_EVTSELO	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 0		Package
Register Address: EB2H, 3762	MSR_C11_PMON_EVTSEL1	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 1		Package
Register Address: EB3H, 3763	MSR_C11_PMON_EVTSEL2	
Uncore C-Box 11 PerfMon Event Select for C-Box 11 Counter 2		Package
Register Address: EB4H, 3764	MSR_C11_PMON_EVTSEL3	
Uncore C-box 11 PerfMon Event Select for C-Box 11 Counter 3		Package
Register Address: EB5H, 3765	MSR_C11_PMON_BOX_FILTER0	
Uncore C-Box 11 PerfMon Box Wide Filter 0		Package
Register Address: EB6H, 3766	MSR_C11_PMON_BOX_FILTER1	
Uncore C-Box 11 PerfMon Box Wide Filter 1		Package
Register Address: EB7H, 3767	MSR_C11_PMON_BOX_STATUS	
Uncore C-Box 11 PerfMon Box Wide Status		Package
Register Address: EB8H, 3768	MSR_C11_PMON_CTR0	
Uncore C-Box 11 PerfMon Counter 0		Package
Register Address: EB9H, 3769	MSR_C11_PMON_CTR1	
Uncore C-Box 11 PerfMon Counter 1		Package
Register Address: EBAH, 3770	MSR_C11_PMON_CTR2	
Uncore C-Box 11 PerfMon Counter 2		Package
Register Address: EBBH, 3771	MSR_C11_PMON_CTR3	
Uncore C-Box 11 PerfMon Counter 3		Package
Register Address: EC0H, 3776	MSR_C12_PMON_BOX_CTL	
Uncore C-Box 12 PerfMon Local Box Wide Control		Package
Register Address: EC1H, 3777	MSR_C12_PMON_EVTSELO	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 0		Package
Register Address: EC2H, 3778	MSR_C12_PMON_EVTSEL1	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 1		Package
Register Address: EC3H, 3779	MSR_C12_PMON_EVTSEL2	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 2		Package
Register Address: EC4H, 3780	MSR_C12_PMON_EVTSEL3	
Uncore C-Box 12 PerfMon Event Select for C-Box 12 Counter 3		Package
Register Address: EC5H, 3781	MSR_C12_PMON_BOX_FILTER0	
Uncore C-Box 12 PerfMon Box Wide Filter 0		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EC6H, 3782	MSR_C12_PMON_BOX_FILTER1	
Uncore C-Box 12 PerfMon Box Wide Filter 1		Package
Register Address: EC7H, 3783	MSR_C12_PMON_BOX_STATUS	
Uncore C-Box 12 PerfMon Box Wide Status		Package
Register Address: EC8H, 3784	MSR_C12_PMON_CTR0	
Uncore C-Box 12 PerfMon Counter 0		Package
Register Address: EC9H, 3785	MSR_C12_PMON_CTR1	
Uncore C-Box 12 PerfMon Counter 1		Package
Register Address: ECAH, 3786	MSR_C12_PMON_CTR2	
Uncore C-Box 12 PerfMon Counter 2		Package
Register Address: ECBH, 3787	MSR_C12_PMON_CTR3	
Uncore C-Box 12 PerfMon Counter 3		Package
Register Address: ED0H, 3792	MSR_C13_PMON_BOX_CTL	
Uncore C-Box 13 PerfMon local box wide control.		Package
Register Address: ED1H, 3793	MSR_C13_PMON_EVTSEL0	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 0		Package
Register Address: ED2H, 3794	MSR_C13_PMON_EVTSEL1	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 1		Package
Register Address: ED3H, 3795	MSR_C13_PMON_EVTSEL2	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 2		Package
Register Address: ED4H, 3796	MSR_C13_PMON_EVTSEL3	
Uncore C-Box 13 PerfMon Event Select for C-Box 13 Counter 3		Package
Register Address: ED5H, 3797	MSR_C13_PMON_BOX_FILTER0	
Uncore C-Box 13 PerfMon Box Wide Filter 0		Package
Register Address: ED6H, 3798	MSR_C13_PMON_BOX_FILTER1	
Uncore C-Box 13 PerfMon Box Wide Filter 1		Package
Register Address: ED7H, 3799	MSR_C13_PMON_BOX_STATUS	
Uncore C-Box 13 PerfMon Box Wide Status		Package
Register Address: ED8H, 3800	MSR_C13_PMON_CTR0	
Uncore C-Box 13 PerfMon Counter 0		Package
Register Address: ED9H, 3801	MSR_C13_PMON_CTR1	
Uncore C-Box 13 PerfMon Counter 1		Package
Register Address: EDAH, 3802	MSR_C13_PMON_CTR2	
Uncore C-Box 13 PerfMon Counter 2		Package
Register Address: EDBH, 3803	MSR_C13_PMON_CTR3	
Uncore C-Box 13 PerfMon Counter 3		Package
Register Address: EE0H, 3808	MSR_C14_PMON_BOX_CTL	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 14 PerfMon Local Box Wide Control		Package
Register Address: EE1H, 3809	MSR_C14_PMON_EVNTSELO	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 0		Package
Register Address: EE2H, 3810	MSR_C14_PMON_EVNTSEL1	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 1		Package
Register Address: EE3H, 3811	MSR_C14_PMON_EVNTSEL2	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 2		Package
Register Address: EE4H, 3812	MSR_C14_PMON_EVNTSEL3	
Uncore C-Box 14 PerfMon Event Select for C-Box 14 Counter 3		Package
Register Address: EE5H, 3813	MSR_C14_PMON_BOX_FILTER	
Uncore C-Box 14 PerfMon Box Wide Filter 0		Package
Register Address: EE6H, 3814	MSR_C14_PMON_BOX_FILTER1	
Uncore C-Box 14 PerfMon Box Wide Filter 1		Package
Register Address: EE7H, 3815	MSR_C14_PMON_BOX_STATUS	
Uncore C-Box 14 PerfMon Box Wide Status		Package
Register Address: EE8H, 3816	MSR_C14_PMON_CTR0	
Uncore C-Box 14 PerfMon Counter 0		Package
Register Address: EE9H, 3817	MSR_C14_PMON_CTR1	
Uncore C-Box 14 PerfMon Counter 1		Package
Register Address: EEAH, 3818	MSR_C14_PMON_CTR2	
Uncore C-Box 14 PerfMon Counter 2		Package
Register Address: EEBH, 3819	MSR_C14_PMON_CTR3	
Uncore C-Box 14 PerfMon Counter 3		Package
Register Address: EF0H, 3824	MSR_C15_PMON_BOX_CTL	
Uncore C-Box 15 PerfMon Local Box Wide Control		Package
Register Address: EF1H, 3825	MSR_C15_PMON_EVNTSELO	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 0		Package
Register Address: EF2H, 3826	MSR_C15_PMON_EVNTSEL1	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 1		Package
Register Address: EF3H, 3827	MSR_C15_PMON_EVNTSEL2	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 2		Package
Register Address: EF4H, 3828	MSR_C15_PMON_EVNTSEL3	
Uncore C-Box 15 PerfMon Event Select for C-Box 15 Counter 3		Package
Register Address: EF5H, 3829	MSR_C15_PMON_BOX_FILTER0	
Uncore C-Box 15 PerfMon Box Wide Filter 0		Package
Register Address: EF6H, 3830	MSR_C15_PMON_BOX_FILTER1	
Uncore C-Box 15 PerfMon Box Wide Filter 1		Package

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: EF7H, 3831	MSR_C15_PMON_BOX_STATUS	
Uncore C-Box 15 PerfMon Box Wide Status		Package
Register Address: EF8H, 3832	MSR_C15_PMON_CTR0	
Uncore C-Box 15 PerfMon Counter 0		Package
Register Address: EF9H, 3833	MSR_C15_PMON_CTR1	
Uncore C-Box 15 PerfMon Counter 1		Package
Register Address: EFAH, 3834	MSR_C15_PMON_CTR2	
Uncore C-Box 15 PerfMon Counter 2		Package
Register Address: EFBH, 3835	MSR_C15_PMON_CTR3	
Uncore C-Box 15 PerfMon Counter 3		Package
Register Address: F00H, 3840	MSR_C16_PMON_BOX_CTL	
Uncore C-Box 16 PerfMon for Box-Wide Control		Package
Register Address: F01H, 3841	MSR_C16_PMON_EVTNSELO	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 0		Package
Register Address: F02H, 3842	MSR_C16_PMON_EVTNSEL1	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 1		Package
Register Address: F03H, 3843	MSR_C16_PMON_EVTNSEL2	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 2		Package
Register Address: F04H, 3844	MSR_C16_PMON_EVTNSEL3	
Uncore C-Box 16 PerfMon Event Select for C-Box 16 Counter 3		Package
Register Address: F05H, 3845	MSR_C16_PMON_BOX_FILTER0	
Uncore C-Box 16 PerfMon Box Wide Filter 0		Package
Register Address: F06H, 3846	MSR_C16_PMON_BOX_FILTER1	
Uncore C-Box 16 PerfMon Box Wide Filter 1		Package
Register Address: F07H, 3847	MSR_C16_PMON_BOX_STATUS	
Uncore C-Box 16 PerfMon Box Wide Status		Package
Register Address: F08H, 3848	MSR_C16_PMON_CTR0	
Uncore C-Box 16 PerfMon Counter 0		Package
Register Address: F09H, 3849	MSR_C16_PMON_CTR1	
Uncore C-Box 16 PerfMon Counter 1		Package
Register Address: FOAH, 3850	MSR_C16_PMON_CTR2	
Uncore C-Box 16 PerfMon Counter 2		Package
Register Address: FOBH, 3851	MSR_C16_PMON_CTR3	
Uncore C-Box 16 PerfMon Counter 3		Package
Register Address: F10H, 3856	MSR_C17_PMON_BOX_CTL	
Uncore C-Box 17 PerfMon for Box-Wide Control		Package
Register Address: F11H, 3857	MSR_C17_PMON_EVTNSELO	

Table 2-33. Uncore PMU MSRs in the Intel® Xeon® Processor E5 v3 Family (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 0		Package
Register Address: F12H, 3858	MSR_C17_PMON_EVTSEL1	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 1		Package
Register Address: F13H, 3859	MSR_C17_PMON_EVTSEL2	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 2		Package
Register Address: F14H, 3860	MSR_C17_PMON_EVTSEL3	
Uncore C-Box 17 PerfMon Event Select for C-Box 17 Counter 3		Package
Register Address: F15H, 3861	MSR_C17_PMON_BOX_FILTER0	
Uncore C-Box 17 PerfMon Box Wide Filter 0		Package
Register Address: F16H, 3862	MSR_C17_PMON_BOX_FILTER1	
Uncore C-Box 17 PerfMon Box Wide Filter 1		Package
Register Address: F17H, 3863	MSR_C17_PMON_BOX_STATUS	
Uncore C-Box 17 PerfMon Box Wide Status		Package
Register Address: F18H, 3864	MSR_C17_PMON_CTR0	
Uncore C-Box 17 PerfMon Counter 0		Package
Register Address: F19H, 3865	MSR_C17_PMON_CTR1	
Uncore C-Box 17 PerfMon Counter 1		Package
Register Address: F1AH, 3866	MSR_C17_PMON_CTR2	
Uncore C-Box 17 PerfMon Counter 2		Package
Register Address: F1BH, 3867	MSR_C17_PMON_CTR3	
Uncore C-Box 17 PerfMon Counter 3		Package

2.15 MSRS IN THE INTEL® CORE™ M PROCESSORS AND THE 5TH GENERATION INTEL® CORE™ PROCESSORS

The Intel® Core™ M-5xxx processors, 5th generation Intel® Core™ Processors, and the Intel® Xeon® Processor E3-1200 v4 family are based on Broadwell microarchitecture. The Intel® Core™ M-5xxx processors and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH. The Intel® Xeon® Processor E3-1200 v4 family and 5th generation Intel® Core™ Processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_47H. Processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH or 06_47H support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, Table 2-34, and Table 2-35. For an MSR listed in Table 2-35 that also appears in the model-specific tables of prior generations, Table 2-35 supersedes prior generation tables.

Table 2-34 lists MSRs that are common to processors based on the Broadwell microarchitectures (including CPUID Signature DisplayFamily_DisplayModel values of 06_3DH, 06_47H, 06_4FH, and 06_56H).

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2 and Section 21.6.2.2, "Global Counter Control Facilities."		Thread
0	Ovf_PMC0	
1	Ovf_PMC1	
2	Ovf_PMC2	
3	Ovf_PMC3	
31:4	Reserved	
32	Ovf_FixedCtr0	
33	Ovf_FixedCtr1	
34	Ovf_FixedCtr2	
54:35	Reserved.	
55	Trace_ToPA_PMI See Section 35.2.7.2, "Table of Physical Addresses (ToPA)."	
60:56	Reserved.	
61	Ovf_Uncore	
62	Ovf_BufDSSAVE	
63	CondChgd	
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2 and Section 21.6.2.2, "Global Counter Control Facilities."		Thread
0	Set 1 to clear Ovf_PMC0.	
1	Set 1 to clear Ovf_PMC1.	
2	Set 1 to clear Ovf_PMC2.	
3	Set 1 to clear Ovf_PMC3.	
31:4	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	
33	Set 1 to clear Ovf_FixedCtr1.	
34	Set 1 to clear Ovf_FixedCtr2	
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI. See Section 35.2.7.2, "Table of Physical Addresses (ToPA)."	
60:56	Reserved.	
61	Set 1 to clear Ovf_Uncore.	
62	Set 1 to clear Ovf_BufDSSAVE.	
63	Set 1 to clear CondChgd.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W)		Thread
6:0	Reserved.	
M-1:7	Base physical address. <i>M</i> is the value enumerated by CPUID.80000008H:EAX[7:0].	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:M	Reserved.	
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W)		Thread
6:0	Reserved.	
31:7	MaskOrTableOffset	
63:32	Output Offset.	
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	Reserved, must be zero.	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	Reserved, must be zero.	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	Reserved; writing 0 will #GP if also setting TraceEn.	
63:14	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	Reserved, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
63:6	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved.	
63:5	CR3[63:5] value to match.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	

Table 2-34. Additional MSRs Common to Processors Based on Broadwell Microarchitectures

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	

Table 2-35 lists MSRs that are specific to Intel Core M processors and 5th Generation Intel Core Processors.

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core
3:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
9:4	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/WO)	
24:16	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	

Table 2-35. Additional MSRs Supported by Intel® Core™ M Processors and 5th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Enable Package C-State Auto-Demotion (R/W)	
30	Enable Package C-State Undemotion (R/W)	
63:31	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
39:32	Maximum Ratio Limit for 5C Maximum turbo ratio limit of 5core active.	Package
47:40	Maximum Ratio Limit for 6C Maximum turbo ratio limit of 6core active.	Package
63:48	Reserved.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
See Table 2-20, Table 2-21, Table 2-22, Table 2-25, Table 2-29, Table 2-30, and Table 2-34 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_3DH.		

2.16 MSRS IN THE INTEL® XEON® PROCESSOR E5 V4 FAMILY

The MSRs listed in Table 2-36 are available and common to the Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H) and to the Intel Xeon processors E5 v4 and E7 v4 families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). These processors are based on Broadwell microarchitecture.

See Section 2.16.1 for lists of tables of MSRs that are supported by the Intel® Xeon® Processor D Family.

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/W/O) See Table 2-2.	
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
16	Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).	
24:17	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1, indicates that the SMM code access restriction is supported and a host-space interface available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 1C	Package
15:8	Maximum Ratio Limit for 2C	Package
23:16	Maximum Ratio Limit for 3C	Package
31:24	Maximum Ratio Limit for 4C	Package
39:32	Maximum Ratio Limit for 5C	Package
47:40	Maximum Ratio Limit for 6C	Package
55:48	Maximum Ratio Limit for 7C	Package
63:56	Maximum Ratio Limit for 8C	Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT1	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
7:0	Maximum Ratio Limit for 9C	Package
15:8	Maximum Ratio Limit for 10C	Package

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
23:16	Maximum Ratio Limit for 11C	Package
31:24	Maximum Ratio Limit for 12C	Package
39:32	Maximum Ratio Limit for 13C	Package
47:40	Maximum Ratio Limit for 14C	Package
55:48	Maximum Ratio Limit for 15C	Package
63:56	Maximum Ratio Limit for 16C	Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 16.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 16.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
63:15	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
7	Reserved.	
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0) When set, processor core frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
2	Power Budget Management Status (R0) When set, frequency is reduced below the operating system request due to PBM limit.	
3	Platform Configuration Services Status (R0) When set, frequency is reduced below the operating system request due to PCS limit.	
4	Reserved.	
5	Autonomous Utilization-Based Frequency Control Status (R0) When set, frequency is reduced below the operating system request because the processor has detected that utilization is low.	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from the Voltage Regulator.	
7	Reserved.	
8	Electrical Design Point Status (R0) When set, frequency is reduced below the operating system request due to electrical design point constraints (e.g., maximum electrical current consumption).	
9	Reserved.	
10	Multi-Core Turbo Status (R0) When set, frequency is reduced below the operating system request due to Multi-Core Turbo limits.	
12:11	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Core Frequency P1 Status (R0) When set, frequency is reduced below max non-turbo P1.	
14	Core Max N-Core Turbo Frequency Limiting Status (R0) When set, frequency is reduced below max n-core turbo frequency.	
15	Core Frequency Limiting Status (R0) When set, frequency is reduced below the operating system request.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
18	Power Budget Management Log When set, indicates that the PBM Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19	Platform Configuration Services Log When set, indicates that the PCS Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20	Reserved.	
21	Autonomous Utilization-Based Frequency Control Log When set, indicates that the AUBFC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	Reserved.	
24	Electrical Design Point Log When set, indicates that the EDP Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Multi-Core Turbo Log When set, indicates that the Multi-Core Turbo Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28:27	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
29	Core Frequency P1 Log When set, indicates that the Core Frequency P1 Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
30	Core Max N-Core Turbo Frequency Limiting Log When set, indicates that the Core Max n-core Turbo Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
31	Core Frequency Limiting Log When set, indicates that the Core Frequency Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:32	Reserved.	
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 16.4.2, "Enabling HWP."		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 16.4.3, "HWP Performance Range and Dynamic Capabilities."		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 16.4.4, "Managing HWP."		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	
23:16	Desired Performance (R/W)	
63:24	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 16.4.5, "HWP Feedback."		Thread
1:0	Reserved.	
2	Excursion to Minimum (R/O)	
63:3	Reserved.	
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.	
31:8	Reserved.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	

Table 2-36. Additional MSRs Common to the Intel® Xeon® Processor D and the Intel® Xeon® Processor E5 v4 Family Based on Broadwell Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.16.1 Additional MSRs Supported in the Intel® Xeon® Processor D Product Family

The MSRs listed in Table 2-37 are available to Intel® Xeon® Processor D Product Family (CPUID Signature DisplayFamily_DisplayModel value of 06_56H). The Intel® Xeon® processor D product family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-29, Table 2-34, Table 2-36, and Table 2-37.

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 10 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	

Table 2-37. Additional MSRs Supported by Intel® Xeon® Processor D with a CPUID Signature DisplayFamily_DisplayModel Value of 06_56H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
See Table 2-20, Table 2-29, Table 2-34, and Table 2-36 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_56H.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.16.2 Additional MSRs Supported in Intel® Xeon® Processors E5 v4 and E7 v4 Families

The MSRs listed in Table 2-37 are available to the Intel® Xeon® Processor E5 v4 and E7 v4 Families (CPUID Signature DisplayFamily_DisplayModel value of 06_4FH). The Intel® Xeon® processor E5 v4 family is based on Broadwell microarchitecture and supports the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-29, Table 2-34, Table 2-36, and Table 2-38.

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1ACH, 428	MSR_TURBO_RATIO_LIMIT3	
Config Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1.		Package
62:0	Reserved.	Package
63	Semaphore for Turbo Ratio Limit Configuration If 1, the processor uses override configuration ¹ specified in MSR_TURBO_RATIO_LIMIT, MSR_TURBO_RATIO_LIMIT1, and MSR_TURBO_RATIO_LIMIT2. If 0, the processor uses factory-set configuration (Default).	Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package

**Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature
DisplayFamily_DisplayModel Value of 06_4FH**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 294H, 660	IA32_MC20_CTL2	
See Table 2-2.		Package
Register Address: 295H, 661	IA32_MC21_CTL2	
See Table 2-2.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from the Intel QPI 0 module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the home agent HA 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the home agent HA 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 435H, 1077	IA32_MC13_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 through MC 16 report MC errors from each channel of the integrated memory controllers.		Package

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC17 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo0, CBo3, CBo6, CBo9, CBo12, CBo15.	Package	
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC18 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo1, CBo4, CBo7, CBo10, CBo13, CBo16.	Package	
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.	Package	
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.	Package	
Register Address: 44EH, 1102	IA32_MC19_ADDR	

Table 2-38. Additional MSRs Supported by Intel® Xeon® Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_4FH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from the following pair of CBo/L3 Slices (if the pair is present): CBo2, CBo5, CBo8, CBo11, CBo14, CBo17.		Package
Register Address: 450H, 1104	IA32_MC20_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 451H, 1105	IA32_MC20_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 452H, 1106	IA32_MC20_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 453H, 1107	IA32_MC20_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC20 reports MC errors from the Intel QPI 1 module.		Package
Register Address: 454H, 1108	IA32_MC21_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 455H, 1109	IA32_MC21_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 456H, 1110	IA32_MC21_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: 457H, 1111	IA32_MC21_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC21 reports MC errors from the Intel QPI 2 module.		Package
Register Address: C81H, 3201	IA32_L3_QOS_CFG	
Cache Allocation Technology Configuration (R/W)		Package
0	CAT Enable. Set 1 to enable Cache Allocation Technology.	
63:1	Reserved.	
See Table 2-20, Table 2-21, Table 2-29, and Table 2-30 for other MSR definitions applicable to processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_45H.		

NOTES:

1. An override configuration lower than the factory-set configuration is always supported. An override configuration higher than the factory-set configuration is dependent on features specific to the processor and the platform.

2.17 MSRS IN THE 6TH—13TH GENERATION INTEL® CORE™ PROCESSORS, 1ST—5TH GENERATION INTEL® XEON® SCALABLE PROCESSOR FAMILIES, INTEL® CORE™ ULTRA 7 PROCESSORS, 8TH GENERATION INTEL® CORE™ i3 PROCESSORS, INTEL® XEON® E PROCESSORS, INTEL® XEON® 6 P-CORE PROCESSORS, INTEL® XEON® 6 E-CORE PROCESSORS, AND INTEL® SERIES 2 CORE™ ULTRA PROCESSORS

6th generation Intel® Core™ processors are based on Skylake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH or 06_5EH.

The Intel® Xeon® Scalable Processor Family based on the Skylake microarchitecture, the 2nd generation Intel® Xeon® Scalable Processor Family based on the Cascade Lake product, and the 3rd generation Intel® Xeon® Scalable Processor Family based on the Cooper Lake product all have a CPUID Signature DisplayFamily_DisplayModel value of 06_55H.

7th generation Intel® Core™ processors are based on the Kaby Lake microarchitecture, 8th generation and 9th generation Intel® Core™ processors, and Intel® Xeon® E processors are based on Coffee Lake microarchitecture; these processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH.

8th generation Intel® Core™ i3 processors are based on Cannon Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

10th generation Intel® Core™ processors are based on Comet Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_A5H or 06_A6H) and Ice Lake microarchitecture (with a CPUID Signature DisplayFamily_DisplayModel value of 06_7EH).

11th generation Intel® Core™ processors are based on Tiger Lake microarchitecture and have a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH.

The 3rd generation Intel® Xeon® Scalable Processor Family is based on Ice Lake microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH.

12th generation Intel® Core™ processors supporting the Alder Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_97H or 06_9AH.

13th generation Intel® Core™ processors supporting the Raptor Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BAH, 06_B7H, or 06_BFH.

The 4th generation Intel® Xeon® Scalable Processor Family is based on Sapphire Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_8FH.

The 5th generation Intel® Xeon® Scalable Processor Family is based on Emerald Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_CFH.

The Intel® Core™ Ultra 7 processors supporting the Meteor Lake hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_AAH.

The Intel® Xeon® 6 P-core processor is based on the Granite Rapids microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_ADH or 06_AEH.

The Intel® Xeon® 6 E-core processor is based on the Sierra Forest microarchitecture and has a CPUID Signature DisplayFamily_DisplayModel value of 06_AFH.

The Intel® Series 2 Core™ Ultra processors supporting the Lunar Lake performance hybrid architecture have a CPUID Signature DisplayFamily_DisplayModel value of 06_BDH.

These processors support the MSR interfaces listed in Table 2-20, Table 2-21, Table 2-25, Table 2-29, Table 2-35, and Table 2-39¹. For an MSR listed in Table 2-39 that also appears in the model-specific tables of prior generations, Table 2-39 supersedes prior generation tables.

Tables 2-40 through 2-60 list additional supported MSR interfaces introduced in specific processors; see each table for additional details.

The notation of “Platform” in the Scope column (with respect to MSR_PLATFORM_ENERGY_COUNTER and MSR_PLATFORM_POWER_LIMIT) is limited to the power-delivery domain and the specifics of the power delivery integration may vary by platform vendor’s implementation.

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
MTRR Capability (R/O, Architectural) See Table 2-2		Thread
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal threshold #1 Status (R/O) See Table 2-2.	
7	Thermal threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	

1. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core: 3F7H. MSRs at the following addresses are not supported in the 12th generation Intel Core processor E-core or P-core: 652H, 653H, 655H, 656H, DB0H, DB1H, DB2H, and D90H.

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
9	Thermal Threshold #2 Log (R/WCO) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WCO) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	
13	Current Limit Log (R/WCO) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WCO) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode R/O if MSR_PLATFORM_INFO.[28] = 0, and R/W if MSR_PLATFORM_INFO.[28] = 1		Package
7:0	Maximum Ratio Limit for 1C Maximum turbo ratio limit of 1 core active.	Package
15:8	Maximum Ratio Limit for 2C Maximum turbo ratio limit of 2 core active.	Package
23:16	Maximum Ratio Limit for 3C Maximum turbo ratio limit of 3 core active.	Package
31:24	Maximum Ratio Limit for 4C Maximum turbo ratio limit of 4 core active.	Package
63:32	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-4) that points to the MSR containing the most recent branch record.		Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register See http://biosbits.org .		Core
0	Reserved.	
1	C1E Enable (R/W) When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	Package
18:2	Reserved.	
19	Disable Energy Efficiency Optimization (R/W) Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.	
20	Disable Race to Halt Optimization (R/W) Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.	
63:21	Reserved.	
Register Address: 300H, 768	MSR_SGXOWNERPOCH0	
Lower 64 Bit CR_SGXOWNERPOCH (W) Writes do not update CR_SGXOWNERPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Lower 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 301H, 769	MSR_SGXOWNERPOCH1	
Upper 64 Bit CR_SGXOWNERPOCH (W) Writes do not update CR_SGXOWNERPOCH if CPUID.12H.00H:EAX.SGX1 is 1 on any thread in the package.		Package
63:0	Upper 64 bits of an 128-bit external entropy value for key derivation of an enclave.	
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2 and Section 21.2.4, "Architectural Performance Monitoring Version 4."		
0	Ovf_PMC0	Thread
1	Ovf_PMC1	Thread
2	Ovf_PMC2	Thread
3	Ovf_PMC3	Thread
4	Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4)	Thread
5	Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5)	Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6)	Thread
7	Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7)	Thread
31:8	Reserved.	
32	Ovf_FixedCtr0	Thread
33	Ovf_FixedCtr1	Thread
34	Ovf_FixedCtr2	Thread
54:35	Reserved	
55	Trace_ToPA_PMI	Thread
57:56	Reserved.	
58	LBR_Frz	Thread
59	CTR_Frz	Thread
60	ASCI	Thread
61	Ovf_Uncore	Thread
62	Ovf_BufDSSAVE	Thread
63	CondChgd	Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_STATUS_RESET	
See Table 2-2 and Section 21.2.4, “Architectural Performance Monitoring Version 4.”		
0	Set 1 to clear Ovf_PMC0.	Thread
1	Set 1 to clear Ovf_PMC1.	Thread
2	Set 1 to clear Ovf_PMC2.	Thread
3	Set 1 to clear Ovf_PMC3.	Thread
4	Set 1 to clear Ovf_PMC4 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to clear Ovf_PMC5 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to clear Ovf_PMC6 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to clear Ovf_PMC7 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to clear Ovf_FixedCtr0.	Thread
33	Set 1 to clear Ovf_FixedCtr1.	Thread
34	Set 1 to clear Ovf_FixedCtr2.	Thread
54:35	Reserved.	
55	Set 1 to clear Trace_ToPA_PMI.	Thread
57:56	Reserved.	
58	Set 1 to clear LBR_Frz.	Thread
59	Set 1 to clear CTR_Frz.	Thread
60	Set 1 to clear ASCI.	Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
61	Set 1 to clear Ovf_Uncore.	Thread
62	Set 1 to clear Ovf_BufDSSAVE.	Thread
63	Set 1 to clear CondChgd.	Thread
Register Address: 391H, 913	IA32_PERF_GLOBAL_STATUS_SET	
See Table 2-2 and Section 21.2.4, "Architectural Performance Monitoring Version 4."		
0	Set 1 to cause Ovf_PMC0 = 1.	Thread
1	Set 1 to cause Ovf_PMC1 = 1.	Thread
2	Set 1 to cause Ovf_PMC2 = 1.	Thread
3	Set 1 to cause Ovf_PMC3 = 1.	Thread
4	Set 1 to cause Ovf_PMC4 = 1 (if CPUID.0AH:EAX[15:8] > 4).	Thread
5	Set 1 to cause Ovf_PMC5 = 1 (if CPUID.0AH:EAX[15:8] > 5).	Thread
6	Set 1 to cause Ovf_PMC6 = 1 (if CPUID.0AH:EAX[15:8] > 6).	Thread
7	Set 1 to cause Ovf_PMC7 = 1 (if CPUID.0AH:EAX[15:8] > 7).	Thread
31:8	Reserved.	
32	Set 1 to cause Ovf_FixedCtr0 = 1.	Thread
33	Set 1 to cause Ovf_FixedCtr1 = 1.	Thread
34	Set 1 to cause Ovf_FixedCtr2 = 1.	Thread
54:35	Reserved.	
55	Set 1 to cause Trace_ToPA_PMI = 1.	Thread
57:56	Reserved.	
58	Set 1 to cause LBR_Frz = 1.	Thread
59	Set 1 to cause CTR_Frz = 1.	Thread
60	Set 1 to cause ASCI = 1.	Thread
61	Set 1 to cause Ovf_Uncore.	Thread
62	Set 1 to cause Ovf_BufDSSAVE.	Thread
63	Reserved.	
Register Address: 392H, 914	IA32_PERF_GLOBAL_INUSE	
See Table 2-2.		Thread
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W)		Thread
2:0	Event Code Select	
3	Reserved	
4	Event Code Select High	
7:5	Reserved.	
19:8	IDQ_Bubble_Length Specifier	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
22:20	IDQ_Bubble_Width Specifier	
63:23	Reserved.	
Register Address: 500H, 1280	IA32_SGX_SVN_STATUS	
Status and SVN Threshold of SGX Support for ACM (R/O)		Thread
0	Lock See Section 41.11.3, “Interactions with Authenticated Code Modules (ACMs).”	
15:1	Reserved.	
23:16	SGX_SVN_SINIT See Section 41.11.3, “Interactions with Authenticated Code Modules (ACMs).”	
63:24	Reserved.	
Register Address: 560H, 1376	IA32_RTIT_OUTPUT_BASE	
Trace Output Base Register (R/W) See Table 2-2.		Thread
Register Address: 561H, 1377	IA32_RTIT_OUTPUT_MASK_PTRS	
Trace Output Mask Pointers Register (R/W) See Table 2-2.		Thread
Register Address: 570H, 1392	IA32_RTIT_CTL	
Trace Control Register (R/W)		Thread
0	TraceEn	
1	CYCEn	
2	OS	
3	User	
6:4	Reserved, must be zero.	
7	CR3Filter	
8	ToPA Writing 0 will #GP if also setting TraceEn.	
9	MTCEn	
10	TSCEn	
11	DisRETC	
12	Reserved, must be zero.	
13	BranchEn	
17:14	MTCFreq	
18	Reserved, must be zero.	
22:19	CycThresh	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
23	Reserved, must be zero.	
27:24	PSBFreq	
31:28	Reserved, must be zero.	
35:32	ADDR0_CFG	
39:36	ADDR1_CFG	
63:40	Reserved, must be zero.	
Register Address: 571H, 1393	IA32_RTIT_STATUS	
Tracing Status Register (R/W)		Thread
0	FilterEn, writes ignored.	
1	ContexEn, writes ignored.	
2	TriggerEn, writes ignored.	
3	Reserved	
4	Error (R/W)	
5	Stopped	
31:6	Reserved, must be zero.	
48:32	PacketByteCnt	
63:49	Reserved, must be zero.	
Register Address: 572H, 1394	IA32_RTIT_CR3_MATCH	
Trace Filter CR3 Match Register (R/W)		Thread
4:0	Reserved	
63:5	CR3[63:5] value to match	
Register Address: 580H, 1408	IA32_RTIT_ADDR0_A	
Region 0 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 581H, 1409	IA32_RTIT_ADDR0_B	
Region 0 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 582H, 1410	IA32_RTIT_ADDR1_A	
Region 1 Start Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 583H, 1411	IA32_RTIT_ADDR1_B	
Region 1 End Address (R/W)		Thread
63:0	See Table 2-2.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
PP0 Energy Status (R/O) See Section 16.10.4, "PP0/PP1 RAPL Domains."		Package
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_COUNTER	
Platform Energy Counter (R/O) This MSR is valid only if both platform vendor hardware implementation and BIOS enablement support it. This MSR will read 0 if not valid.		Platform
31:0	Total energy consumed by all devices in the platform that receive power from integrated power delivery mechanism, included platform devices are processor cores, SOC, memory, add-on or peripheral devices that get powered directly from the platform power delivery means. The energy units are specified in the MSR_RAPL_POWER_UNIT.Energy_Status_Unit.	
63:32	Reserved.	
Register Address: 64EH, 1614	MSR_PPERF	
Productive Performance Count (R/O)		Thread
63:0	Hardware's view of workload scalability. See Section 16.4.5.1.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
	Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)	Package
0	PROCHOT Status (R0) When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	Residency State Regulation Status (R0) When set, frequency is reduced below the operating system request due to residency state regulation limit.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR Therm Alert Status (R0) When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	
7	VR Therm Design Current Status (R0) When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	Other Status (R0) When set, frequency is reduced below the operating system request due to electrical or other constraints.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	Max Turbo Limit Status (R0) When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	Turbo Transition Attenuation Status (R0) When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	Residency State Regulation Log When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th—13th Generation Intel® Core™ Processors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
24	Other Log When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Max Turbo Limit Log When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	Turbo Transition Attenuation Log When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 652H, 1618	MSR_PKG_HDC_CONFIG	
HDC Configuration (R/W)		Package
2:0	PKG_Cx_Monitor Configures Package Cx state threshold for MSR_PKG_HDC_DEEP_RESIDENCY.	
63: 3	Reserved.	
Register Address: 653H, 1619	MSR_CORE_HDC_RESIDENCY	
Core HDC Idle Residency (R/O)		Core
63:0	Core_Cx_Duty_Cycle_Cnt	
Register Address: 655H, 1621	MSR_PKG_HDC_SHALLOW_RESIDENCY	
Accumulate the cycles the package was in C2 state and at least one logical processor was in forced idle (R/O)		Package
63:0	Pkg_C2_Duty_Cycle_Cnt	
Register Address: 656H, 1622	MSR_PKG_HDC_DEEP_RESIDENCY	
Package Cx HDC Idle Residency (R/O)		Package
63:0	Pkg_Cx_Duty_Cycle_Cnt	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_CO	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Core-count Weighted C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 659H, 1625	MSR_ANY_CORE_C0	
Any Core C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if any processor core in the package is in C0.	
Register Address: 65AH, 1626	MSR_ANY_GFXE_C0	
Any Graphics Engine C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.	
Register Address: 65BH, 1627	MSR_CORE_GFXE_OVERLAP_C0	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
63:0	Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows.		Platform
14:0	Platform Power Limit #1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	Enable Platform Power Limit #1 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #1 over the time window specified by Power Limit #1 Time Window.	
16	Platform Clamping Limitation #1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #1 value. This bit is writeable only when CPUID.06H:EAX[4] is set.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
23:17	Time Window for Platform Power Limit #1 Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) $((1+(X/4)) \cdot (2^Y))$, where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:24	Reserved.	
46:32	Platform Power Limit #2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit # 1).	
47	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
48	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
62:49	Reserved.	
63	Lock. Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 690H, 1680	MSR_LASTBRANCH_16_FROM_IP	
Last Branch Record 16 From IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 19.12.		Thread
Register Address: 691H, 1681	MSR_LASTBRANCH_17_FROM_IP	
Last Branch Record 17 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 692H, 1682	MSR_LASTBRANCH_18_FROM_IP	
Last Branch Record 18 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 693H, 1683	MSR_LASTBRANCH_19_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 19 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 694H, 1684	MSR_LASTBRANCH_20_FROM_IP	
Last Branch Record 20 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 695H, 1685	MSR_LASTBRANCH_21_FROM_IP	
Last Branch Record 21 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 696H, 1686	MSR_LASTBRANCH_22_FROM_IP	
Last Branch Record 22 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 697H, 1687	MSR_LASTBRANCH_23_FROM_IP	
Last Branch Record 23 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 698H, 1688	MSR_LASTBRANCH_24_FROM_IP	
Last Branch Record 24 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 699H, 1689	MSR_LASTBRANCH_25_FROM_IP	
Last Branch Record 25 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69AH, 1690	MSR_LASTBRANCH_26_FROM_IP	
Last Branch Record 26 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69BH, 1691	MSR_LASTBRANCH_27_FROM_IP	
Last Branch Record 27 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69CH, 1692	MSR_LASTBRANCH_28_FROM_IP	
Last Branch Record 28 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69DH, 1693	MSR_LASTBRANCH_29_FROM_IP	
Last Branch Record 29 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69EH, 1694	MSR_LASTBRANCH_30_FROM_IP	
Last Branch Record 30 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 69FH, 1695	MSR_LASTBRANCH_31_FROM_IP	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 31 From IP (R/W) See description of MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 6B0H, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Processor Graphics (R/W) (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	Inefficient Operation Status (R0) When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	Inefficient Operation Log When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:29	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in the Ring Interconnect (R/W) (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT Status (R0) When set, frequency is reduced due to assertion of external PROCHOT.	
1	Thermal Status (R0) When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	Running Average Thermal Limit Status (R0) When set, frequency is reduced due to running average thermal limit.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	VR Therm Alert Status (R0) When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR Thermal Design Current Status (R0) When set, frequency is reduced due to VR TDC limit.	
8	Other Status (R0) When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	Package/Platform-Level Power Limiting PL1 Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL1.	
11	Package/Platform-Level PL2 Power Limiting Status (R0) When set, frequency is reduced due to package/Platform-level power limiting PL2/PL3.	
15:12	Reserved	
16	PROCHOT Log When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	Running Average Thermal Limit Log When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR Therm Alert Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR Thermal Design Current Log When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	Other Log When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
25	Reserved.	
26	Package/Platform-Level PL1 Power Limiting Log When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	Package/Platform-Level PL2 Power Limiting Log When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:28	Reserved.	
Register Address: 6D0H, 1744	MSR_LASTBRANCH_16_TO_IP	
Last Branch Record 16 To IP (R/W) One of 32 triplets of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction. See also: - Last Branch Record Stack TOS at 1C9H. - Section 19.12.		Thread
Register Address: 6D1H, 1745	MSR_LASTBRANCH_17_TO_IP	
Last Branch Record 17 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D2H, 1746	MSR_LASTBRANCH_18_TO_IP	
Last Branch Record 18 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D3H, 1747	MSR_LASTBRANCH_19_TO_IP	
Last Branch Record 19 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D4H, 1748	MSR_LASTBRANCH_20_TO_IP	
Last Branch Record 20 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D5H, 1749	MSR_LASTBRANCH_21_TO_IP	
Last Branch Record 21 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D6H, 1750	MSR_LASTBRANCH_22_TO_IP	
Last Branch Record 22 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D7H, 1751	MSR_LASTBRANCH_23_TO_IP	
Last Branch Record 23 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D8H, 1752	MSR_LASTBRANCH_24_TO_IP	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 24 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6D9H, 1753	MSR_LASTBRANCH_25_TO_IP	
Last Branch Record 25 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DAH, 1754	MSR_LASTBRANCH_26_TO_IP	
Last Branch Record 26 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DBH, 1755	MSR_LASTBRANCH_27_TO_IP	
Last Branch Record 27 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DCH, 1756	MSR_LASTBRANCH_28_TO_IP	
Last Branch Record 28 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DDH, 1757	MSR_LASTBRANCH_29_TO_IP	
Last Branch Record 29 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DEH, 1758	MSR_LASTBRANCH_30_TO_IP	
Last Branch Record 30 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 6DFH, 1759	MSR_LASTBRANCH_31_TO_IP	
Last Branch Record 31 To IP (R/W) See description of MSR_LASTBRANCH_0_TO_IP.		Thread
Register Address: 770H, 1904	IA32_PM_ENABLE	
See Section 16.4.2, “Enabling HWP.”		Package
Register Address: 771H, 1905	IA32_HWP_CAPABILITIES	
See Section 16.4.3, “HWP Performance Range and Dynamic Capabilities.”		Thread
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Section 16.4.4, “Managing HWP.”		Package
Register Address: 773H, 1907	IA32_HWP_INTERRUPT	
See Section 16.4.6, “HWP Notifications.”		Thread
Register Address: 774H, 1908	IA32_HWP_REQUEST	
See Section 16.4.4, “Managing HWP.”		Thread
7:0	Minimum Performance (R/W)	
15:8	Maximum Performance (R/W)	
23:16	Desired Performance (R/W)	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	Energy/Performance Preference (R/W)	
41:32	Activity Window (R/W)	
42	Package Control (R/W)	
63:43	Reserved.	
Register Address: 777H, 1911	IA32_HWP_STATUS	
See Section 16.4.5, “HWP Feedback.”		Thread
Register Address: D90H, 3472	IA32_BNDCFGS	
See Table 2-2.		Thread
Register Address: DA0H, 3488	IA32_XSS	
See Table 2-2.		Thread
Register Address: DB0H, 3504	IA32_PKG_HDC_CTL	
See Section 16.5.2, “Package level Enabling HDC.”		Package
Register Address: DB1H, 3505	IA32_PM_CTL1	
See Section 16.5.3, “Logical-Processor Level HDC Control.”		Thread
Register Address: DB2H, 3506	IA32_THREAD_STALL	
See Section 16.5.4.1, “IA32_THREAD_STALL.”		Thread
Register Address: DCOH, 3520	MSR_LBR_INFO_0	
Last Branch Record 0 Additional Information (R/W) One of 32 triplet of last branch record registers on the last branch record stack. This part of the stack contains flag, TSX-related and elapsed cycle information. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 19.9.1, “LBR Stack.”		Thread
Register Address: DC1H, 3521	MSR_LBR_INFO_1	
Last Branch Record 1 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC2H, 3522	MSR_LBR_INFO_2	
Last Branch Record 2 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC3H, 3523	MSR_LBR_INFO_3	
Last Branch Record 3 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC4H, 3524	MSR_LBR_INFO_4	
Last Branch Record 4 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC5H, 3525	MSR_LBR_INFO_5	
Last Branch Record 5 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: DC6H, 3526	MSR_LBR_INFO_6	
Last Branch Record 6 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC7H, 3527	MSR_LBR_INFO_7	
Last Branch Record 7 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC8H, 3528	MSR_LBR_INFO_8	
Last Branch Record 8 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DC9H, 3529	MSR_LBR_INFO_9	
Last Branch Record 9 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCAH, 3530	MSR_LBR_INFO_10	
Last Branch Record 10 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCBH, 3531	MSR_LBR_INFO_11	
Last Branch Record 11 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCCH, 3532	MSR_LBR_INFO_12	
Last Branch Record 12 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCDH, 3533	MSR_LBR_INFO_13	
Last Branch Record 13 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCEH, 3534	MSR_LBR_INFO_14	
Last Branch Record 14 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DCFH, 3535	MSR_LBR_INFO_15	
Last Branch Record 15 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDOH, 3536	MSR_LBR_INFO_16	
Last Branch Record 16 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD1H, 3537	MSR_LBR_INFO_17	
Last Branch Record 17 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD2H, 3538	MSR_LBR_INFO_18	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 18 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD3H, 3539	MSR_LBR_INFO_19	
Last Branch Record 19 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD4H, 3540	MSR_LBR_INFO_20	
Last Branch Record 20 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD5H, 3541	MSR_LBR_INFO_21	
Last Branch Record 21 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD6H, 3542	MSR_LBR_INFO_22	
Last Branch Record 22 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD7H, 3543	MSR_LBR_INFO_23	
Last Branch Record 23 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD8H, 3544	MSR_LBR_INFO_24	
Last Branch Record 24 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DD9H, 3545	MSR_LBR_INFO_25	
Last Branch Record 25 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDAH, 3546	MSR_LBR_INFO_26	
Last Branch Record 26 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDBH, 3547	MSR_LBR_INFO_27	
Last Branch Record 27 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDCH, 3548	MSR_LBR_INFO_28	
Last Branch Record 28 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDDH, 3549	MSR_LBR_INFO_29	
Last Branch Record 29 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDEH, 3550	MSR_LBR_INFO_30	

Table 2-39. Additional MSRs Supported by the 6th–13th Generation Intel® Core™ Processors, 1st–5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 E-Core Processors, Intel® Xeon® 6 P-Core Processors, and Intel® Series 2 Core™ Ultra Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Last Branch Record 30 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread
Register Address: DDFH, 3551	MSR_LBR_INFO_31	
Last Branch Record 31 Additional Information (R/W) See description of MSR_LBR_INFO_0.		Thread

Table 2-40 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_4EH, 06_5EH, 06_8EH, 06_9EH, or 06_66H.

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSELO	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	
Uncore Arb Unit, Counter 1 Event Select MSR		Package

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 706H, 1798	MSR_UNC_CBO_0_PERFCTRO	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 707H, 1799	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 716H, 1814	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 717H, 1815	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 726H, 1830	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 727H, 1831	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 736H, 1846	MSR_UNC_CBO_3_PERFCTRO	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 737H, 1847	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	

Table 2-40. Uncore PMU MSRs Supported by 6th Generation, 7th Generation, and 8th Generation Intel® Core™ Processors, and 8th generation Intel® Core™ i3 Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	Slice 3 select.	
4	Slice 4select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.1 MSRs Introduced in 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Table 2-41 lists additional MSRs for 7th generation and 8th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8EH or 06_9EH. For an MSR listed in Table 2-41 that also appears in the model-specific tables of prior generations, Table 2-41 supersedes prior generation tables.

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 80H, 128	MSR_TRACE_HUB_STH ACPIBAR_BASE	
NPK Address Used by AET Messages (R/W)		Package
0	Lock Bit If set, this MSR cannot be re-written anymore. Lock bit has to be set in order for the AET packets to be directed to NPK MMIO.	
17:1	Reserved.	
63:18	ACPIBAR_BASE_ADDRESS AET target address in NPK MMIO space.	
Register Address: 1F4H, 500	MSR_PRMRR_PHYS_BASE	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MemType PRMRR BASE MemType.	
11:3	Reserved.	

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
45:12	Base PRMRR Base Address.	
63:46	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask Control Register (R/W)		Core
9:0	Reserved.	
10	Lock Lock bit for the PRMRR.	
11	VLD Enable bit for the PRMRR.	
45:12	Mask PRMRR MASK bits.	
63:46	Reserved.	
Register Address: 1FBH, 507	MSR_PRMRR_VALID_CONFIG	
Valid PRMRR Configurations (R/W)		Core
0	1M supported MEE size.	
4:1	Reserved.	
5	32M supported MEE size.	
6	64M supported MEE size.	
7	128M supported MEE size.	
31:8	Reserved.	
Register Address: 2F4H, 756	MSR_UNCORE_PRMRR_PHYS_BASE ¹	
(R/W) The PRMRR range is used to protect the processor reserved memory from unauthorized reads and writes. Any IO access to this range is aborted. This register controls the location of the PRMRR range by indicating its starting address. It functions in tandem with the PRMRR mask register.		Package
11:0	Reserved.	
PAWIDTH-1:12	Range Base This field corresponds to bits PAWIDTH-1:12 of the base address memory range which is allocated to PRMRR memory.	
63:PAWIDTH	Reserved.	
Register Address: 2F5H, 757	MSR_UNCORE_PRMRR_PHYS_MASK ¹	
(R/W) This register controls the size of the PRMRR range by indicating which address bits must match the PRMRR base register value.		Package
9:0	Reserved.	
10	Lock Setting this bit locks all writeable settings in this register, including itself.	

Table 2-41. Additional MSRs Supported by the 7th Generation and 8th Generation Intel® Core™ Processors Based on Kaby Lake Microarchitecture and Coffee Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
11	Range_En Indicates whether the PRMRR range is enabled and valid.	
38:12	Range_Mask This field indicates which address bits must match PRMRR base in order to qualify as an PRMRR access.	
63:39	Reserved.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	

NOTES:

1. This MSR is specific to 7th generation and 8th generation Intel® Core™ processors.

2.17.2 MSRs Specific to 8th Generation Intel® Core™ i3 Processors

Table 2-42 lists additional MSRs for 8th generation Intel Core i3 processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H. For an MSR listed in Table 2-42 that also appears in the model-specific tables of prior generations, Table 2-42 supersedes prior generation tables.

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
17	SGX Launch Control Enable (R/WL) This bit must be set to enable runtime reconfiguration of SGX Launch Control via IA32_SGXLEPUBKEYHASHn MSR. Available only if CPUID.07H.00H:ECX[30] = 1.	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
18	SGX Global Functions Enable (R/WL)	
63:21	Reserved.	
Register Address: 350H, 848	MSR_BR_DETECT_CTRL	
Branch Monitoring Global Control (R/W)		
0	EnMonitoring Global enable for branch monitoring.	
1	EnExcept Enable branch monitoring event signaling on threshold trip. The branch monitoring event handler is signaled via the existing PMI signaling mechanism as programmed from the corresponding local APIC LVT entry.	
2	EnLBRFrz Enable LBR freeze on threshold trip. This will cause the LBR frozen bit 58 to be set in IA32_PERF_GLOBAL_STATUS when a triggering condition occurs and this bit is enabled.	
3	DisableInGuest When set to '1', branch monitoring, event triggering and LBR freeze actions are disabled when operating at VMX non-root operation.	
7:4	Reserved.	
17:8	WindowSize Window size defined by WindowCntSel. Values 0 - 1023 are supported. Once the Window counter reaches the WindowSize count both the Window Counter and all Branch Monitoring Counters are cleared.	
23:18	Reserved.	
25:24	WindowCntSel Window event count select: '00 = Instructions retired. '01 = Branch instructions retired '10 = Return instructions retired. '11 = Indirect branch instructions retired.	
26	CntAndMode When set to '1', the overall branch monitoring event triggering condition is true only if all enabled counters' threshold conditions are true. When '0', the threshold tripping condition is true if any enabled counters' threshold is true.	
63:27	Reserved.	
Register Address: 351H, 849	MSR_BR_DETECT_STATUS	
Branch Monitoring Global Status (R/W)		
0	Branch Monitoring Event Signaled When set to '1', Branch Monitoring event signaling is blocked until this bit is cleared by software.	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	LBRsValid This status bit is set to '1' if the LBR state is considered valid for sampling by branch monitoring software.	
7:2	Reserved.	
8	CntrHit0 Branch monitoring counter #0 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.	
9	CntrHit1 Branch monitoring counter #1 threshold hit. This status bit is sticky and once set requires clearing by software. Counter operation continues independent of the state of the bit.	
15:10	Reserved. Reserved for additional branch monitoring counters threshold hit status.	
25:16	CountWindow The current value of the window counter. The count value is frozen on a valid branch monitoring triggering condition. This is a 10-bit unsigned value.	
31:26	Reserved. Reserved for future extension of CountWindow.	
39:32	Count0 The current value of counter 0 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit0 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	
47:40	Count1 The current value of counter 1 updated after each occurrence of the event being counted. The count value is frozen on a valid branch monitoring triggering condition (in which case CntrHit1 will also be set). This is an 8-bit signed value (2's complement). Heuristic events which only increment will saturate and freeze at maximum value 0xFF (256). RET-CALL event counter saturate at maximum value 0x7F (+127) and minimum value 0x80 (-128).	
63:48	Reserved.	
Register Address: 354H–355H, 852–853	MSR_BR_DETECT_COUNTER_CONFIG_i	
Branch Monitoring Detect Counter Configuration (R/W)		
0	CntrEn Enable counter.	

**Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors
Based on Cannon Lake Microarchitecture (Contd.)**

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
7:1	CntrEvSel Event select (other values #GP) '0000000 = RETs. '0000001 = RET-CALL bias. '0000010 = RET mispredicts. '0000011 = Branch (all) mispredicts. '0000100 = Indirect branch mispredicts. '0000101 = Far branch instructions.	
14:8	CntrThreshold Threshold (an unsigned value of 0 to 127 supported). The value 0 of counter threshold will result in event signaled after every instruction. #GP if threshold is < 2.	
15	MispredEventCnt Mispredict events counting behavior: '0 = Mispredict events are counted in a window. '1 = Mispredict events are counted based on a consecutive occurrence. CntrThreshold is treated as # of consecutive mispredicts. This control bit only applies to events specified by CntrEvSel that involve a prediction (0000010, 0000011, 0000100). Setting this bit for other events is ignored.	
63:16	Reserved.	
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Package C3 Residency Counter (R/O)		Package
63:0	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.	
Register Address: 620H, 1568	MSR_RING_RATIO_LIMIT	
Ring Ratio Limit (R/W) This register provides Min/Max Ratio Limits for the LLC and Ring.		Package
6:0	MAX_Ratio This field is used to limit the max ratio of the LLC/Ring.	
7	Reserved.	
14:8	MIN_Ratio Writing to this field controls the minimum possible ratio of the LLC/Ring.	
63:15	Reserved.	
Register Address: 660H, 1632	MSR_CORE_C1_RESIDENCY	
Core C1 Residency Counter (R/O)		Core

Table 2-42. Additional MSRs Supported by the 8th Generation Intel® Core™ i3 Processors Based on Cannon Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:0	Value since last reset for the Core C1 residency. Counter rate is the Max Non-Turbo frequency (same as TSC). This counter counts in case both of the core's threads are in an idle state and at least one of the core's thread residency is in a C1 state or in one of its sub states. The counter is updated only after a core C state exit. Note: Always reads 0 if core C1 is unsupported. A value of zero indicates that this processor does not support core C1 or never entered core C1 level state.	
Register Address: 662H, 1634	MSR_CORE_C3_RESIDENCY	
Core C3 Residency Counter (R/O)		Core
63:0	Will always return 0.	

Table 2-43 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_66H.

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 394H, 916	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved	
22	Enable counting.	
63:23	Reserved.	
Register Address: 395H, 917	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
47:0	Current count.	
63:48	Reserved.	
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Report the number of C-Box units with performance counters, including processor cores and processor graphics.	
63:4	Reserved.	
Register Address: 3B0H, 946	MSR_UNC_ARB_PERFCTR0	
Uncore Arb Unit, Performance Counter 0		Package
Register Address: 3B1H, 947	MSR_UNC_ARB_PERFCTR1	
Uncore Arb Unit, Performance Counter 1		Package
Register Address: 3B2H, 944	MSR_UNC_ARB_PERFEVTSEL0	
Uncore Arb Unit, Counter 0 Event Select MSR		Package
Register Address: 3B3H, 945	MSR_UNC_ARB_PERFEVTSEL1	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore Arb unit, Counter 1 Event Select MSR		Package
Register Address: 700H, 1792	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 701H, 1793	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 702H, 1794	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 703H, 1795	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 708H, 1800	MSR_UNC_CBO_1_PERFEVTSELO	
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 709H, 1801	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 70AH, 1802	MSR_UNC_CBO_1_PERFCTR0	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 70BH, 1803	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 710H, 1808	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 711H, 1809	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 712H, 1810	MSR_UNC_CBO_2_PERFCTR0	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 713H, 1811	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 718H, 1816	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 719H, 1817	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 71AH, 1818	MSR_UNC_CBO_3_PERFCTR0	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 71BH, 1819	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 720H, 1824	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 721H, 1825	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 722H, 1826	MSR_UNC_CBO_4_PERFCTR0	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 723H, 1827	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 728H, 1832	MSR_UNC_CBO_5_PERFEVTSELO	
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 729H, 1833	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 72AH, 1834	MSR_UNC_CBO_5_PERFCTR0	
Uncore C-Box 5, Performance Counter 0		Package
Register Address: 72BH, 1835	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 730H, 1840	MSR_UNC_CBO_6_PERFEVTSELO	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 731H, 1841	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 732H, 1842	MSR_UNC_CBO_6_PERFCTR0	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 733H, 1843	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 738H, 1848	MSR_UNC_CBO_7_PERFEVTSELO	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 739H, 1849	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 73AH, 1850	MSR_UNC_CBO_7_PERFCTR0	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 73BH, 1851	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: E01H, 3585	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	
3	Slice 3 select.	
4	Slice 4select.	
18:5	Reserved.	
29	Enable all uncore counters.	

Table 2-43. Uncore PMU MSRs Supported by Intel® Core™ Processors Based on Cannon Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: E02H, 3586	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.3 MSRs Introduced in 10th Generation Intel® Core™ Processors

Table 2-44 lists additional MSRs for 10th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_7EH. For an MSR listed in Table 2-44 that also appears in the model-specific tables of prior generations, Table 2-44 supersedes prior generation tables.

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
28:0	Reserved.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 10.1.2.3, "Features to Disable Bus Locks."	
30	Reserved.	
31	Reserved.	
Register Address: 48H, 72	IA32_SPEC_CTRL	
See Table 2-2.		Core
Register Address: 49H, 73	IA32_PREDICT_CMD	
See Table 2-2.		Thread
Register Address: 8CH, 140	IA32_SGXLEPUBKEYHASH0	
See Table 2-2.		Thread
Register Address: 8DH, 141	IA32_SGXLEPUBKEYHASH1	
See Table 2-2.		Thread
Register Address: 8EH, 142	IA32_SGXLEPUBKEYHASH2	
See Table 2-2.		Thread
Register Address: 8FH, 143	IA32_SGXLEPUBKEYHASH3	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O) This MSR indicates if WRMSR 0x79 failed to configure PRM memory and gives a hint to debug BIOS.		Package
15:0	Error Codes (R/O)	Package
30:16	Reserved.	
31	MCU Partial Success (R/O) When set to 1, WRMSR 0x79 skipped part of the functionality during BIOS.	Thread
Register Address: A5H, 165	MSR_FIT_BIOS_ERROR	
FIT BIOS ERROR (R/W) Report error codes for debug in case the processor failed to parse the Firmware Table in BIOS. Can also be used to log BIOS information.		Thread
7:0	Error Codes (R/W) Error codes for debug.	
15:8	Entry Type (R/W) Failed FIT entry type.	
16	FIT MCU Entry (R/W) FIT contains MCU entry.	
62:17	Reserved.	
63	LOCK (R/W) When set to 1, writes to this MSR will be skipped.	
Register Address: 10BH, 267	IA32_FLUSH_CMD	
See Table 2-2.		Thread
Register Address: 151H, 337	MSR_BIOS_DONE	
BIOS Done (R/WO)		Thread
0	BIOS Done Indication (R/WO) Set by BIOS when it finishes programming the processor and wants to lock the memory configuration from changes by software that is running on this thread. Writes to the bit will be ignored if EAX[0] is 0.	Thread
1	Package BIOS Done Indication (R/O) When set to 1, all threads in the package have bit 0 of this MSR set.	Package
31:2	Reserved.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Write Data to a Crash Log Configuration		Thread
0	CDDIS: CrashDump_Disable If set, indicates that Crash Dump is disabled.	
63:1	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRM_BASE_0	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE: PRMRR BASE Memory Type.	
3	CONFIGURED: PRMRR BASE Configured.	
11:4	Reserved.	
51:12	BASE: PRMRR Base Address.	
63:52	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter Register 3 (R/W) Bit definitions are the same as found in IA32_FIXED_CTR0, offset 309H. See Table 2-2.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) Reports metrics directly. Software can check (and/or expose to its guests) the availability of PERF_METRICS feature using IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE (bit 15).		Thread
7:0	Retiring. Percent of utilized slots by uops that eventually retire (commit).	
15:8	Bad Speculation. Percent of wasted slots due to incorrect speculation, covering utilized by uops that do not retire, or recovery bubbles (unutilized slots).	
23:16	Frontend Bound. Percent of unutilized slots where front-end did not deliver a uop while back-end is ready.	
31:24	Backend Bound. Percent of unutilized slots where a uop was not delivered to back-end due to lack of back-end resources.	
63:32	Reserved.	
Register Address: 3F2H, 1010	MSR_PEBS_DATA_CFG	
PEBS Data Configuration (R/W) Provides software the capability to select data groups of interest and thus reduce the record size in memory and record generation latency. Hence, a PEBS record's size and layout vary based on the selected groups. The MSR also allows software to select LBR depth for branch data records.		Thread
0	Memory Info. Setting this bit will capture memory information such as the linear address, data source and latency of the memory access in the PEBS record.	
1	GPRs. Setting this bit will capture the contents of the General Purpose registers in the PEBS record.	
2	XMMs. Setting this bit will capture the contents of the XMM registers in the PEBS record.	
3	LBRs. Setting this bit will capture LBR TO, FROM, and INFO in the PEBS record.	
23:4	Reserved.	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	LBR Entries. Set the field to the desired number of entries - 1. For example, if the LBR_entries field is 0, a single entry will be included in the record. To include 32 LBR entries, set the LBR_entries field to 31 (0x1F). To ensure all PEBS records are 16-byte aligned, software can use LBR_entries that is multiple of 3.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	L1 Scrubbing Enable When set to 1, enable L1 scrubbing.	
31:1	Reserved.	
Register Address: 657H, 1623	MSR_FAST_UNCORE_MSRS_CTL	
Fast WRMSR/RDMSR Control MSR (R/W)		Thread
3:0	FAST_ACCESS_ENABLE: Bit 0: When set to '1', provides a hint for the hardware to enable fast access mode for the IA32_HWP_REQUEST MSR. This bit is sticky and is cleaned by the hardware only during reset time. This bit is valid only if FAST_UNCORE_MSRS_CAPABILITY[0] is set. Setting this bit will cause CPUID.06H:EAX[18] to be set.	
31:4	Reserved.	
Register Address: 65EH, 1630	MSR_FAST_UNCORE_MSRS_STATUS	
Indication of Uncore MSRs, Post Write Activates		Thread
0	Indicates whether the CPU is still in the middle of writing IA32_HWP_REQUEST MSR, even after the WRMSR instruction has retired. A value of 1 indicates the last write of IA32_HWP_REQUEST is still ongoing. A value of 0 indicates the last write of IA32_HWP_REQUEST is visible outside the logical processor. Software can use the status of this bit to avoid overwriting IA32_HWP_REQUEST.	
31:1	Reserved.	
Register Address: 65FH, 1631	MSR_FAST_UNCORE_MSRS_CAPABILITY	
Fast WRMSR/RDMSR Enumeration MSR (R/O)		Thread
3:0	MSRS_CAPABILITY: Bit 0: If set to '1', hardware supports the fast access mode for the IA32_HWP_REQUEST MSR.	
31:4	Reserved.	
Register Address: 772H, 1906	IA32_HWP_REQUEST_PKG	
See Table 2-2.		Package
Register Address: 775H, 1909	IA32_PECI_HWP_REQUEST_INFO	
See Table 2-2.		Package
Register Address: 777H, 1911	IA32_HWP_STATUS	

Table 2-44. MSRs Supported by the 10th Generation Intel® Core™ Processors (Ice Lake Microarchitecture) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Thread

2.17.4 MSRs Introduced in the 11th Generation Intel® Core™ Processors based on Tiger Lake Microarchitecture

Table 2-45 lists additional MSRs for 11th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_8CH or 06_8DH. The MSRs listed in Table 2-44 are also supported by these processors. For an MSR listed in Table 2-45 that also appears in the model-specific tables of prior generations, Table 2-45 supersedes prior generation tables.

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: A0H, 160	MSR_BIOS_MCU_ERRORCODE	
BIOS MCU ERRORCODE (R/O)		Package
15:0	Error Codes	
31:16	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) This MSR indicates if WRMSR 79H failed to configure PRM memory and gives a hint to debug BIOS.		Thread
30:0	Reserved.	
31	MCU Partial Success When set to 1, WRMSR 79H skipped part of the functionality during BIOS.	
63:32	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/O) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Package
1:0	Reserved.	
2	FUSA_SUPPORTED	
3	RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.	
4	Reserved.	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL (MSR address 33H).	
31:6	Reserved.	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTLSS3	
IA32_VMX_PROCBASED_CTLSS3 This MSR enumerates the allowed 1-settings of the third set of processor-based controls. Specifically, VM entry allows bit X of the tertiary processor-based VM-execution controls to be 1 if and only if bit X of the MSR is set to 1. If bit X of the MSR is cleared to 0, VM entry fails if control X and the “activate tertiary controls” primary processor-based VM-execution control are both 1.	Core	
0	LOADIWKEY This control determines whether executions of LOADIWKEY cause VM exits.	
63:1	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.	Package	
12:0	PL4 Value PL4 value in 0.125 A increments. This field is locked by VR_CURRENT_CONFIG[LOCK]. When the LOCK bit is set to 1b, this field becomes Read Only.	
30:13	Reserved.	
31	Lock Indication (LOCK) This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1b, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
62:32	Not in use.	
63	Reserved.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PL0_SSP	
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 6A8H, 1704	IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		
Register Address: 990H, 2448	IA32_COPY_STATUS ¹	
See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS ¹	
See Table 2-2.		Platform
Register Address: C82H, 3202	IA32_L2_QOS_CFG	
IA32_CR_L2_QOS_CFG This MSR provides software an enumeration of the parameters that L2 QoS (Intel RDT) support in any particular implementation.		Core
0	CDP_ENABLE When set to 1, it will enable the code and data prioritization for the L2 CAT/Intel RDT feature. When set to 0, code and data prioritization is disabled for L2 CAT/Intel RDT. See Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” for further details on CDP.	
31:1	Reserved.	
Register Address: D10H–D17H, 3220–3351	IA32_L2_QOS_MASK [0-7]	
IA32_CR_L2_QOS_MASK [0-7] Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Package
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	

Table 2-45. Additional MSRs Supported by the 11th Generation Intel® Core™ Processors Based on Tiger Lake Microarchitecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM ¹	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL ¹	
See Table 2-2.		Thread

NOTES:

1. Further details on Key Locker and usage of this MSR can be found here:

<https://software.intel.com/content/www/us/en/develop/download/intel-key-locker-specification.html>.

2.17.5 MSRs Introduced in the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Table 2-46 lists additional MSRs for 12th and 13th generation Intel Core processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH. Table 2-47 lists the MSRs unique to the processor P-core. Table 2-48 lists the MSRs unique to the processor E-core.

The MSRs listed in Table 2-44¹ and Table 2-45 are also supported by these processors. For an MSR listed in Table 2-46, Table 2-47, or Table 2-48 that also appears in the model-specific tables of prior generations, Table 2-46, Table 2-47, and Table 2-48 supersede prior generation tables.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 10.1.2.3, "Features to Disable Bus Locks."	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 10.1.2.3, "Features to Disable Bus Locks."	
30	Reserved.	
31	Reserved.	
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTL5	

1. MSRs at the following addresses are not supported in the 12th and 13th generation Intel Core processor E-core: 30CH, 329H, 541H, and 657H. The MSR at address 657H is not supported in the 12th and 13th generation Intel Core processor P-core.

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: C7H, 199	IA32_PMC6	
General Performance Counter 6 (R/W) See Table 2-2.		Core
Register Address: C8H, 200	IA32_PMC7	
General Performance Counter 7 (R/W) See Table 2-2.		Core
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/O) If CPUID.07H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Package
0	STLB_QOS_SUPPORTED When set to 1, the STLB QoS feature is supported and the STLB QoS MSRs (1A8FH - 1A97H) are accessible. When set to 0, access to these MSRs will #GP.	
1	Reserved.	
2	FUSA_SUPPORTED	
3	RSM_IN_CPL0_ONLY When set to 1, the RSM instruction is only allowed in CPL0 (#GP triggered in any CPL != 0). When set to 0, then any CPL may execute the RSM instruction.	
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.	
6	SNOOP_FILTER_QOS_SUPPORTED When set to 1, the Snoop Filter Qos Mask MSRs are supported. When set to 0, access to these MSRs will #GP.	
7	UC_STORE_THROTTLING_SUPPORTED When set 1, UC Store throttle capability exist through MSR_MEMORY_CTRL (33H) bit 27.	
31:8	Reserved.	
Register Address: E1H, 225	IA32_UMWAIT_CONTROL	
UMWAIT Control (R/W) See Table 2-2.		
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 18CH, 396	IA32_PERFEVTSEL6	
See Table 2-20.		Core
Register Address: 18DH, 397	IA32_PERFEVTSEL7	
See Table 2-20.		Core
Register Address: 195H, 405	IA32_OVERCLOCKING_STATUS	
Overclocking Status (R/O) IA32_ARCH_CAPABILITIES[bit 23] enumerates support for this MSR. See Table 2-2.		Package
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL2	
See Table 2-2.		
Register Address: 4C7H, 1223	IA32_A_PMC6	
Full Width Writable IA32_PMC6 Alias (R/W) See Table 2-2.		
Register Address: 4C8H, 1224	IA32_A_PMC7	
Full Width Writable IA32_PMC7 Alias (R/W) See Table 2-2.		
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY_COUNTER	
Module C6 Residency Counter (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Module
63:0	Time that this module is in module-specific C6 states since last reset. Counts at 1 Mhz frequency.	
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
Memory Encryption Capability MSR See Table 2-2.		
Register Address: 1200H–121FH, 4608–4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		

Table 2-46. Additional MSRs Supported by the 12th and 13th Generation Intel® Core™ Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H–151FH, 5376–5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H–161FH, 5632–5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		
Register Address: 17D2H, 6098	IA32_THREAD_FEEDBACK_CHAR	
Thread Feedback Characteristics (R/O) See Table 2-2.		
Register Address: 17D4H, 6100	IA32_HW_FEEDBACK_THREAD_CONFIG	
Hardware Feedback Thread Configuration (R/W) See Table 2-2.		
Register Address: 17DAH, 6106	IA32_HRESET_ENABLE	
History Reset Enable (R/W) See Table 2-2.		

The MSRs listed in Table 2-47 are unique to the 12th and 13th generation Intel Core processor P-core. These MSRs are not supported on the processor E-core.

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
Prefetch Disable Bits (R/W)		
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	Reserved.	

Table 2-47. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 3F7H, 1015	MSR_PEBS_FRONTEND	
FrontEnd Precise Event Condition Select (R/W) See Table 2-39.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core
Register Address: D10H–D17H, 3220–3351	IA32_L2_QOS_MASK_[0-7]	
IA32_CR_L2_QOS_MASK_[0-7] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Core
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	

The MSRs listed in Table 2-48 are unique to the 12th and 13th generation Intel Core processor E-core. These MSRs are not supported on the processor P-core.

Table 2-48. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D10H–D1FH, 3220–3359	IA32_L2_QOS_MASK_[0-15]	
IA32_CR_L2_QOS_MASK_[0-15] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Module

Table 2-48. MSRs Supported by 12th and 13th Generation Intel® Core™ Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
19:0	WAYS_MASK Setting a 1 in this bit X allows threads with CLOS <n> (where N is [0-7]) to allocate to way X in the MLC. Ones are only allowed to be written to ways that physically exist in the MLC (CPUID.04H.02H:EBX[31:22] will indicate this). Writing a 1 to a value beyond the highest way or a non-contiguous set of 1s will cause a #GP on the WRMSR to this MSR.	
31:20	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C6H, 5313–5318	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Core
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	

Table 2-49 lists the MSRs of uncore PMU for Intel processors with a CPUID Signature DisplayFamily_DisplayModel value of 06_97H, 06_9AH, 06_BAH, 06_B7H, or 06_BFH.

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 396H, 918	MSR_UNC_CBO_CONFIG	
Uncore C-Box Configuration Information (R/O)		Package
3:0	Specifies the number of C-Box units with programmable counters (including processor cores and processor graphics).	
63:4	Reserved.	
Register Address: 2000H, 8192	MSR_UNC_CBO_0_PERFEVTSELO	
Uncore C-Box 0, Counter 0 Event Select MSR		Package
Register Address: 2001H, 8193	MSR_UNC_CBO_0_PERFEVTSEL1	
Uncore C-Box 0, Counter 1 Event Select MSR		Package
Register Address: 2002H, 8194	MSR_UNC_CBO_0_PERFCTR0	
Uncore C-Box 0, Performance Counter 0		Package
Register Address: 2003H, 8195	MSR_UNC_CBO_0_PERFCTR1	
Uncore C-Box 0, Performance Counter 1		Package
Register Address: 2008H, 8200	MSR_UNC_CBO_1_PERFEVTSELO	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore C-Box 1, Counter 0 Event Select MSR		Package
Register Address: 2009H, 8201	MSR_UNC_CBO_1_PERFEVTSEL1	
Uncore C-Box 1, Counter 1 Event Select MSR		Package
Register Address: 200AH, 8202	MSR_UNC_CBO_1_PERFCTRO	
Uncore C-Box 1, Performance Counter 0		Package
Register Address: 200BH, 8203	MSR_UNC_CBO_1_PERFCTR1	
Uncore C-Box 1, Performance Counter 1		Package
Register Address: 2010H, 8208	MSR_UNC_CBO_2_PERFEVTSELO	
Uncore C-Box 2, Counter 0 Event Select MSR		Package
Register Address: 2011H, 8209	MSR_UNC_CBO_2_PERFEVTSEL1	
Uncore C-Box 2, Counter 1 Event Select MSR		Package
Register Address: 2012H, 8210	MSR_UNC_CBO_2_PERFCTRO	
Uncore C-Box 2, Performance Counter 0		Package
Register Address: 2013H, 8211	MSR_UNC_CBO_2_PERFCTR1	
Uncore C-Box 2, Performance Counter 1		Package
Register Address: 2018H, 8216	MSR_UNC_CBO_3_PERFEVTSELO	
Uncore C-Box 3, Counter 0 Event Select MSR		Package
Register Address: 2019H, 8217	MSR_UNC_CBO_3_PERFEVTSEL1	
Uncore C-Box 3, Counter 1 Event Select MSR		Package
Register Address: 201AH, 8218	MSR_UNC_CBO_3_PERFCTRO	
Uncore C-Box 3, Performance Counter 0		Package
Register Address: 201BH, 8219	MSR_UNC_CBO_3_PERFCTR1	
Uncore C-Box 3, Performance Counter 1		Package
Register Address: 2020H, 8224	MSR_UNC_CBO_4_PERFEVTSELO	
Uncore C-Box 4, Counter 0 Event Select MSR		Package
Register Address: 2021H, 8225	MSR_UNC_CBO_4_PERFEVTSEL1	
Uncore C-Box 4, Counter 1 Event Select MSR		Package
Register Address: 2022H, 8226	MSR_UNC_CBO_4_PERFCTRO	
Uncore C-Box 4, Performance Counter 0		Package
Register Address: 2023H, 8227	MSR_UNC_CBO_4_PERFCTR1	
Uncore C-Box 4, Performance Counter 1		Package
Register Address: 2028H, 8232	MSR_UNC_CBO_5_PERFEVTSELO	
Uncore C-Box 5, Counter 0 Event Select MSR		Package
Register Address: 2029H, 8233	MSR_UNC_CBO_5_PERFEVTSEL1	
Uncore C-Box 5, Counter 1 Event Select MSR		Package
Register Address: 202AH, 8234	MSR_UNC_CBO_5_PERFCTRO	
Uncore C-Box 5, Performance Counter 0		Package

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 202BH, 8235	MSR_UNC_CBO_5_PERFCTR1	
Uncore C-Box 5, Performance Counter 1		Package
Register Address: 2030H, 8240	MSR_UNC_CBO_6_PERFEVTSELO	
Uncore C-Box 6, Counter 0 Event Select MSR		Package
Register Address: 2031H, 8241	MSR_UNC_CBO_6_PERFEVTSEL1	
Uncore C-Box 6, Counter 1 Event Select MSR		Package
Register Address: 2032H, 8242	MSR_UNC_CBO_6_PERFCTRO	
Uncore C-Box 6, Performance Counter 0		Package
Register Address: 2033H, 8243	MSR_UNC_CBO_6_PERFCTR1	
Uncore C-Box 6, Performance Counter 1		Package
Register Address: 2038H, 8248	MSR_UNC_CBO_7_PERFEVTSELO	
Uncore C-Box 7, Counter 0 Event Select MSR		Package
Register Address: 2039H, 8249	MSR_UNC_CBO_7_PERFEVTSEL1	
Uncore C-Box 7, Counter 1 Event Select MSR		Package
Register Address: 203AH, 8250	MSR_UNC_CBO_7_PERFCTRO	
Uncore C-Box 7, Performance Counter 0		Package
Register Address: 203BH, 8251	MSR_UNC_CBO_7_PERFCTR1	
Uncore C-Box 7, Performance Counter 1		Package
Register Address: 2040H, 8256	MSR_UNC_CBO_8_PERFEVTSELO	
Uncore C-Box 8, Counter 0 Event Select MSR		Package
Register Address: 2041H, 8257	MSR_UNC_CBO_8_PERFEVTSEL1	
Uncore C-Box 8, Counter 1 Event Select MSR		Package
Register Address: 2042H, 8258	MSR_UNC_CBO_8_PERFCTRO	
Uncore C-Box 8, Performance Counter 0		Package
Register Address: 2043H, 8259	MSR_UNC_CBO_8_PERFCTR1	
Uncore C-Box 8, Performance Counter 1		Package
Register Address: 2048H, 8264	MSR_UNC_CBO_9_PERFEVTSELO	
Uncore C-Box 9, Counter 0 Event Select MSR		Package
Register Address: 2049H, 8265	MSR_UNC_CBO_9_PERFEVTSEL1	
Uncore C-Box 9, Counter 1 Event Select MSR		Package
Register Address: 204AH, 8266	MSR_UNC_CBO_9_PERFCTRO	
Uncore C-Box 9, Performance Counter 0		Package
Register Address: 204BH, 8267	MSR_UNC_CBO_9_PERFCTR1	
Uncore C-Box 9, Performance Counter 1		Package
Register Address: 2FD0H, 12240	MSR_UNC_ARB_0_PERFEVTSELO	
Uncore Arb Unit 0, Counter 0 Event Select MSR		Package
Register Address: 2FD1H, 12241	MSR_UNC_ARB_0_PERFEVTSEL1	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Uncore Arb Unit 0, Counter 1 Event Select MSR		Package
Register Address: 2FD2H, 12242	MSR_UNC_ARB_0_PERFCTRO	
Uncore Arb Unit 0, Performance Counter 0		Package
Register Address: 2FD3H, 12243	MSR_UNC_ARB_0_PERFCTR1	
Uncore Arb Unit 0, Performance Counter 1		Package
Register Address: 2FD4H, 12244	MSR_UNC_ARB_0_PERF_STATUS	
Uncore Arb Unit 0, Performance Status		Package
Register Address: 2FD5H, 12245	MSR_UNC_ARB_0_PERF_CTRL	
Uncore Arb Unit 0, Performance Control		Package
Register Address: 2FD8H, 12248	MSR_UNC_ARB_1_PERFEVTSELO	
Uncore Arb Unit 1, Counter 0 Event Select MSR		Package
Register Address: 2FD9H, 12249	MSR_UNC_ARB_1_PERFEVTSEL1	
Uncore Arb Unit 1, Counter 1 Event Select MSR		Package
Register Address: 2FDAH, 12250	MSR_UNC_ARB_1_PERFCTRO	
Uncore Arb Unit 1, Performance Counter 0		Package
Register Address: 2FDBH, 12251	MSR_UNC_ARB_1_PERFCTR1	
Uncore Arb Unit 1, Performance Counter 1		Package
Register Address: 2FDCB, 12252	MSR_UNC_ARB_1_PERF_STATUS	
Uncore Arb Unit 1, Performance Status		Package
Register Address: 2FDDH, 12253	MSR_UNC_ARB_1_PERF_CTRL	
Uncore Arb Unit 1, Performance Control		Package
Register Address: 2FDEH, 12254	MSR_UNC_PERF_FIXED_CTRL	
Uncore Fixed Counter Control (R/W)		Package
19:0	Reserved.	
20	Enable overflow propagation.	
21	Reserved.	
22	Enable counting.	
63:23	Reserved.	
Register Address: 2FDFH, 12255	MSR_UNC_PERF_FIXED_CTR	
Uncore Fixed Counter		Package
43:0	Current count.	
63:44	Reserved.	
Register Address: 2FF0H, 12272	MSR_UNC_PERF_GLOBAL_CTRL	
Uncore PMU Global Control		Package
0	Slice 0 select.	
1	Slice 1 select.	
2	Slice 2 select.	

Table 2-49. Uncore PMU MSRs Supported by 12th and 13th Generation Intel® Core™ Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	Slice 3 select.	
4	Slice 4 select.	
18:5	Reserved.	
29	Enable all uncore counters.	
30	Enable wake on PMI.	
31	Enable Freezing counter when overflow.	
63:32	Reserved.	
Register Address: 2FF2H, 12274	MSR_UNC_PERF_GLOBAL_STATUS	
Uncore PMU Main Status		Package
0	Fixed counter overflowed.	
1	An ARB counter overflowed.	
2	Reserved.	
3	A CBox counter overflowed (on any slice).	
63:4	Reserved.	

2.17.6 MSRs Introduced in the Intel® Xeon® Scalable Processor Family

The Intel® Xeon® Scalable Processor Family (CUID Signature DisplayFamily_DisplayModel value of 06_55H) supports the MSRs listed in Table 2-50.

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CUID Signature DisplayFamily_DisplayModel Value of 06_55H

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64 Processor (R/W) See Table 2-2.		Thread
0	Lock (R/WL)	
1	Enable VMX Inside SMX Operation (R/WL)	
2	Enable VMX Outside SMX Operation (R/WL)	
14:8	SENTER Local Functions Enables (R/WL)	
15	SENTER Global Functions Enable (R/WL)	
18	SGX Global Functions Enable (R/WL)	
20	LMCE_ENABLED (R/WL)	
63:21	Reserved.	
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	

**Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CUID Signature
DisplayFamily_DisplayModel Value of 06_55H (Contd.)**

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved.	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) See Table 2-26.	Package
22:16	Reserved.	
23	PPIN_CAP (R/O) See Table 2-26.	Package
27:24	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) See Table 2-26.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) See Table 2-26.	Package
30	Programmable TJ OFFSET (R/O) See Table 2-26.	Package
39:31	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) See Table 2-26.	Package
63:48	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states. See http://biosbits.org .		Core

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
2:0	Package C-State Limit (R/W) Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-state support) 001b: C2 010b: C6 (non-retention) 011b: C6 (retention) 111b: No Package C state limits. All C states supported by the processor are available.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W)	
14:11	Reserved.	
15	CFG Lock (R/W0)	
16	Automatic C-State Conversion Enable (R/W) If 1, the processor will convert HALT or MWAIT(C1) to MWAIT(C6).	
24:17	Reserved.	
25	C3 State Auto Demotion Enable (R/W)	
26	C1 State Auto Demotion Enable (R/W)	
27	Enable C3 Undemotion (R/W)	
28	Enable C1 Undemotion (R/W)	
29	Package C State Demotion Enable (R/W)	
30	Package C State Undemotion Enable (R/W)	
63:31	Reserved.	
Register Address: 179H, 377	IA32_MCG_CAP	
Global Machine Check Capability (R/O)		Thread
7:0	Count.	
8	MCG_CTL_P	
9	MCG_EXT_P	
10	MCP_CMCI_P	
11	MCG_TES_P	
15:12	Reserved.	
23:16	MCG_EXT_CNT	
24	MCG_SER_P	
25	MCG_EM_P	
26	MCG_ELOG_P	
63:27	Reserved.	
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
57:0	Reserved.	
58	SMM_Code_Access_Chk (SMM-RO) If set to 1 indicates that the SMM code access restriction is supported and a host-space interface is available to SMM handler.	
59	Long_Flow_Indication (SMM-RO) If set to 1 indicates that the SMM long flow indicator is supported and a host-space interface is available to SMM handler.	
63:60	Reserved.	
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Core
0	Thermal Status (R/O) See Table 2-2.	
1	Thermal Status Log (R/WC0) See Table 2-2.	
2	PROTCHOT # or FORCEPR# Status (R/O) See Table 2-2.	
3	PROTCHOT # or FORCEPR# Log (R/WC0) See Table 2-2.	
4	Critical Temperature Status (R/O) See Table 2-2.	
5	Critical Temperature Status Log (R/WC0) See Table 2-2.	
6	Thermal Threshold #1 Status (R/O) See Table 2-2.	
7	Thermal Threshold #1 Log (R/WC0) See Table 2-2.	
8	Thermal Threshold #2 Status (R/O) See Table 2-2.	
9	Thermal Threshold #2 Log (R/WC0) See Table 2-2.	
10	Power Limitation Status (R/O) See Table 2-2.	
11	Power Limitation Log (R/WC0) See Table 2-2.	
12	Current Limit Status (R/O) See Table 2-2.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
13	Current Limit Log (R/WC0) See Table 2-2.	
14	Cross Domain Limit Status (R/O) See Table 2-2.	
15	Cross Domain Limit Log (R/WC0) See Table 2-2.	
22:16	Digital Readout (R/O) See Table 2-2.	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O) See Table 2-2.	
31	Reading Valid (R/O) See Table 2-2.	
63:32	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R/O) See Table 2-26.	
27:24	TCC Activation Offset (R/W) See Table 2-26.	
63:28	Reserved.	
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
This register defines the ratio limits. RATIO[0:7] must be populated in ascending order. RATIO[i+1] must be less than or equal to RATIO[i]. Entries with RATIO[i] will be ignored. If any of the rules above are broken, the configuration is silently rejected. If the programmed ratio is:		Package
- Above the fused ratio for that core count, it will be clipped to the fuse limits (assuming !OC). - Below the min supported ratio, it will be clipped.		
7:0	RATIO_0 Defines ratio limits.	
15:8	RATIO_1 Defines ratio limits.	
23:16	RATIO_2 Defines ratio limits.	
31:24	RATIO_3 Defines ratio limits.	
39:32	RATIO_4 Defines ratio limits.	
47:40	RATIO_5 Defines ratio limits.	

**Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CUID Signature
DisplayFamily_DisplayModel Value of 06_55H (Contd.)**

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
55:48	RATIO_6 Defines ratio limits.	
63:56	RATIO_7 Defines ratio limits.	
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
This register defines the active core ranges for each frequency point. NUMCORE[0:7] must be populated in ascending order. NUMCORE[i+1] must be greater than NUMCORE[i]. Entries with NUMCORE[i] == 0 will be ignored. The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, the configuration is silently rejected.		Package
7:0	NUMCORE_0 Defines the active core ranges for each frequency point.	
15:8	NUMCORE_1 Defines the active core ranges for each frequency point.	
23:16	NUMCORE_2 Defines the active core ranges for each frequency point.	
31:24	NUMCORE_3 Defines the active core ranges for each frequency point.	
39:32	NUMCORE_4 Defines the active core ranges for each frequency point.	
47:40	NUMCORE_5 Defines the active core ranges for each frequency point.	
55:48	NUMCORE_6 Defines the active core ranges for each frequency point.	
63:56	NUMCORE_7 Defines the active core ranges for each frequency point.	
Register Address: 280H, 640	IA32_MCO_CTL2	
See Table 2-2.		Core
Register Address: 281H, 641	IA32_MC1_CTL2	
See Table 2-2.		Core
Register Address: 282H, 642	IA32_MC2_CTL2	
See Table 2-2.		Core
Register Address: 283H, 643	IA32_MC3_CTL2	
See Table 2-2.		Core
Register Address: 284H, 644	IA32_MC4_CTL2	
See Table 2-2.		Package
Register Address: 285H, 645	IA32_MC5_CTL2	
See Table 2-2.		Package
Register Address: 286H, 646	IA32_MC6_CTL2	
See Table 2-2.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 287H, 647	IA32_MC7_CTL2	
See Table 2-2.		Package
Register Address: 288H, 648	IA32_MC8_CTL2	
See Table 2-2.		Package
Register Address: 289H, 649	IA32_MC9_CTL2	
See Table 2-2.		Package
Register Address: 28AH, 650	IA32_MC10_CTL2	
See Table 2-2.		Package
Register Address: 28BH, 651	IA32_MC11_CTL2	
See Table 2-2.		Package
Register Address: 28CH, 652	IA32_MC12_CTL2	
See Table 2-2.		Package
Register Address: 28DH, 653	IA32_MC13_CTL2	
See Table 2-2.		Package
Register Address: 28EH, 654	IA32_MC14_CTL2	
See Table 2-2.		Package
Register Address: 28FH, 655	IA32_MC15_CTL2	
See Table 2-2.		Package
Register Address: 290H, 656	IA32_MC16_CTL2	
See Table 2-2.		Package
Register Address: 291H, 657	IA32_MC17_CTL2	
See Table 2-2.		Package
Register Address: 292H, 658	IA32_MC18_CTL2	
See Table 2-2.		Package
Register Address: 293H, 659	IA32_MC19_CTL2	
See Table 2-2.		Package
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MCO reports MC errors from the IFU module.		Core
Register Address: 403H, 1027	IA32_MCO_MISC	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC0 reports MC errors from the IFU module.		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 407H, 1031	IA32_MC1_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC1 reports MC errors from the DCU module.		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40BH, 1035	IA32_MC2_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC2 reports MC errors from the DTLB module.		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core
Register Address: 40FH, 1039	IA32_MC3_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC3 reports MC errors from the MLC module.		Core

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module.		Package
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 417H, 1047	IA32_MC5_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC5 reports MC errors from a link interconnect module.		Package
Register Address: 418H, 1048	IA32_MC6_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 419H, 1049	IA32_MC6_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41AH, 1050	IA32_MC6_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41BH, 1051	IA32_MC6_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC6 reports MC errors from the integrated I/O module.		Package
Register Address: 41CH, 1052	IA32_MC7_CTL	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41DH, 1053	IA32_MC7_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41EH, 1054	IA32_MC7_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 41FH, 1055	IA32_MC7_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC7 reports MC errors from the M2M 0.		Package
Register Address: 420H, 1056	IA32_MC8_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 421H, 1057	IA32_MC8_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 422H, 1058	IA32_MC8_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 423H, 1059	IA32_MC8_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC8 reports MC errors from the M2M 1.		Package
Register Address: 424H, 1060	IA32_MC9_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 425H, 1061	IA32_MC9_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 426H, 1062	IA32_MC9_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 427H, 1063	IA32_MC9_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package
Register Address: 428H, 1064	IA32_MC10_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 429H, 1065	IA32_MC10_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 42AH, 1066	IA32_MC10_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 42BH, 1067	IA32_MC10_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 42CH, 1068	IA32_MC11_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 42DH, 1069	IA32_MC11_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 42EH, 1070	IA32_MC11_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 42FH, 1071	IA32_MC11_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC9 - MC11 report MC errors from the CHA.	Package	
Register Address: 430H, 1072	IA32_MC12_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.	Package	
Register Address: 431H, 1073	IA32_MC12_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.	Package	
Register Address: 432H, 1074	IA32_MC12_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.	Package	
Register Address: 433H, 1075	IA32_MC12_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC12 report MC errors from each channel of a link interconnect module.	Package	
Register Address: 434H, 1076	IA32_MC13_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.	Package	
Register Address: 435H, 1077	IA32_MC13_STATUS	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 436H, 1078	IA32_MC13_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 437H, 1079	IA32_MC13_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 438H, 1080	IA32_MC14_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 439H, 1081	IA32_MC14_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43AH, 1082	IA32_MC14_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43BH, 1083	IA32_MC14_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43CH, 1084	IA32_MC15_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43DH, 1085	IA32_MC15_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43EH, 1086	IA32_MC15_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 43FH, 1087	IA32_MC15_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 440H, 1088	IA32_MC16_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 441H, 1089	IA32_MC16_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 442H, 1090	IA32_MC16_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 443H, 1091	IA32_MC16_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 444H, 1092	IA32_MC17_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 445H, 1093	IA32_MC17_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 446H, 1094	IA32_MC17_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 447H, 1095	IA32_MC17_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 448H, 1096	IA32_MC18_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 449H, 1097	IA32_MC18_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44AH, 1098	IA32_MC18_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44BH, 1099	IA32_MC18_MISC	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Banks MC13 through MC 18 report MC errors from the integrated memory controllers.		Package
Register Address: 44CH, 1100	IA32_MC19_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44DH, 1101	IA32_MC19_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44EH, 1102	IA32_MC19_ADDR	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 44FH, 1103	IA32_MC19_MISC	
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC19 reports MC errors from a link interconnect module.		Package
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 16.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 16.10.1, "RAPL Interfaces."	Package
63:20	Reserved.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Out of reset, the min_ratio and max_ratio fields represent the widest possible range of uncore frequencies. Writing to these fields allows software to control the minimum and the maximum frequency that hardware will select.		Package
63:15	Reserved.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
14:8	MIN_RATIO Writing to this field controls the minimum possible ratio of the LLC/Ring.	
7	Reserved.	
6:0	MAX_RATIO This field is used to limit the max ratio of the LLC/Ring.	
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
Reserved (R/O) Reads return 0.		Package
Register Address: C8DH, 3213	IA32_QM_EVTSEL	
Monitoring Event Select Register (R/W) If CPUID.07H.00H:EBX.RDT_M[12] = 1.		Thread
7:0	EventID (R/W) Event encoding: 0x00: No monitoring. 0x01: L3 occupancy monitoring. 0x02: Total memory bandwidth monitoring. 0x03: Local memory bandwidth monitoring. All other encoding reserved.	
31:8	Reserved.	
41:32	RMID (R/W)	
63:42	Reserved.	
Register Address: C8FH, 3215	IA32_PQR_ASSOC	
Resource Association Register (R/W)		Thread
9:0	RMID	
31:10	Reserved.	
51:32	CLOS (R/W)	
63: 52	Reserved.	
Register Address: C90H, 3216	IA32_L3_QOS_MASK_0	
L3 Class Of Service Mask - CLOS 0 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 0.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 0 enforcement.	
63:20	Reserved.	
Register Address: C91H, 3217	IA32_L3_QOS_MASK_1	
L3 Class Of Service Mask - CLOS 1 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 1.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 1 enforcement.	
63:20	Reserved.	
Register Address: C92H, 3218	IA32_L3_QOS_MASK_2	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
L3 Class Of Service Mask - CLOS 2 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 2 >= 2.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 2 enforcement.	
63:20	Reserved.	
Register Address: C93H, 3219	IA32_L3_QOS_MASK_3	
L3 Class Of Service Mask - CLOS 3 (R/W). If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 3.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 3 enforcement.	
63:20	Reserved.	
Register Address: C94H, 3220	IA32_L3_QOS_MASK_4	
L3 Class Of Service Mask - CLOS 4 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 4.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 4 enforcement.	
63:20	Reserved.	
Register Address: C95H, 3221	IA32_L3_QOS_MASK_5	
L3 Class Of Service Mask - CLOS 5 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 5.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 5 enforcement.	
63:20	Reserved.	
Register Address: C96H, 3222	IA32_L3_QOS_MASK_6	
L3 Class Of Service Mask - CLOS 6 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 6.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 6 enforcement.	
63:20	Reserved.	
Register Address: C97H, 3223	IA32_L3_QOS_MASK_7	
L3 Class Of Service Mask - CLOS 7 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 7.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 7 enforcement.	
63:20	Reserved.	
Register Address: C98H, 3224	IA32_L3_QOS_MASK_8	
L3 Class Of Service Mask - CLOS 8 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 8.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 8 enforcement.	
63:20	Reserved.	
Register Address: C99H, 3225	IA32_L3_QOS_MASK_9	
L3 Class Of Service Mask - CLOS 9 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 9.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 9 enforcement.	

Table 2-50. MSRs Supported by the Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_55H (Contd.)

Register Address: Hex, Decimal	Register Name (Former Register Name)	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: C9AH, 3226	IA32_L3_QOS_MASK_10	
L3 Class Of Service Mask - CLOS 10 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 10.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 10 enforcement.	
63:20	Reserved.	
Register Address: C9BH, 3227	IA32_L3_QOS_MASK_11	
L3 Class Of Service Mask - CLOS 11 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 11.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 11 enforcement.	
63:20	Reserved.	
Register Address: C9CH, 3228	IA32_L3_QOS_MASK_12	
L3 Class Of Service Mask - CLOS 12 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 12.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 12 enforcement.	
63:20	Reserved.	
Register Address: C9DH, 3229	IA32_L3_QOS_MASK_13	
L3 Class Of Service Mask - CLOS 13 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 13.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 13 enforcement.	
63:20	Reserved.	
Register Address: C9EH, 3230	IA32_L3_QOS_MASK_14	
L3 Class Of Service Mask - CLOS 14 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 14.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 14 enforcement.	
63:20	Reserved.	
Register Address: C9FH, 3231	IA32_L3_QOS_MASK_15	
L3 Class Of Service Mask - CLOS 15 (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] >= 15.		Package
0:19	CBM: Bit vector of available L3 ways for CLOS 15 enforcement.	
63:20	Reserved.	

2.17.7 MSRs Specific to the 3rd Generation Intel® Xeon® Scalable Processor Family Based on Ice Lake Microarchitecture

The 3rd generation Intel® Xeon® Scalable Processor Family based on Ice Lake microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_6AH or 06_6CH) support the MSRs listed in Table 2-51.

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 612H, 1554	MSR_PACKAGE_ENERGY_TIME_STATUS	
Package energy consumed by the entire CPU (R/W)		Package
31:0	Total amount of energy consumed since last reset.	
63:32	Total time elapsed when the energy was last updated. This is a monotonic increment counter with auto wrap back to zero after overflow. Unit is 10ns.	
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
Allows software to set power limits for the DRAM domain and measurement attributes associated with each limit.		Package
14:0	DRAM_PP_PWR_LIM: Power Limit[0] for DDR domain. Units = Watts, Format = 11.3, Resolution = 0.125W, Range = 0-2047.875W.	
15	PWR_LIM_CTRL_EN: Power Limit[0] enable bit for DDR domain.	
16	Reserved.	
23:17	CTRL_TIME_WIN: Power Limit[0] time window Y value, for DDR domain. Actual time_window for RAPL is: $(1/1024 \text{ seconds}) * (1 + (x/4)) * (2^y)$	
62:24	Reserved.	
63	PP_PWR_LIM_LOCK: When set, this entire register becomes read-only. This bit will typically be set by BIOS during boot.	
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
31:0	Energy in 15.3 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM Power Parameters (R/W)		Package
14:0	Spec DRAM Power (DRAM_TDP): The Spec power allowed for DRAM. The TDP setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
15	Reserved.	

Table 2-51. MSRs Supported by the 3rd Generation Intel® Xeon® Scalable Processor Family with a CPUID Signature DisplayFamily_DisplayModel Value of 06_6AH or 06_6CH (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30:16	Minimal DRAM Power (DRAM_MIN_PWR): The minimal power setting allowed for DRAM. Lower values will be clamped to this value. The minimum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
31	Reserved.	
46:32	Maximal Package Power (DRAM_MAX_PWR): The maximal power setting allowed for DRAM. Higher values will be clamped to this value. The maximum setting is typical (not guaranteed). The units for this value are defined in MSR_DRAM_POWER_INFO_UNIT[PWR_UNIT].	
47	Reserved.	
54:48	Maximal Time Window (DRAM_MAX_WIN): The maximal time window allowed for the DRAM. Higher values will be clamped to this value. $x = \text{PKG\_MAX\_WIN}[54:53]$ $y = \text{PKG\_MAX\_WIN}[52:48]$ The timing interval window is a floating-point number given by $1.x * \text{power}(2,y)$. The unit of measurement is defined in MSR_DRAM_POWER_INFO_UNIT[TIME_UNIT].	
62:55	Reserved.	
63	LOCK: Lock bit to lock the register.	
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
See Table 2-2.		
Register Address: 982H, 2434	IA32_TME_ACTIVATE	
See Table 2-2.		
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
See Table 2-2.		
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
See Table 2-2.		

2.17.8 MSRs Specific to the 4th and 5th Generation Intel® Xeon® Scalable Processor Families

The 4th generation Intel® Xeon® Scalable Processor Family based on Sapphire Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_8FH) and the 5th generation Intel® Xeon® Scalable Processor Family based on Emerald Rapids microarchitecture (CPUID Signature DisplayFamily_DisplayModel value of 06_CFH) both support the MSRs listed in Section 2.17, “MSRs In the 6th—13th Generation Intel® Core™ Proces-

sors, 1st—5th Generation Intel® Xeon® Scalable Processor Families, Intel® Core™ Ultra 7 Processors, 8th Generation Intel® Core™ i3 Processors, Intel® Xeon® E Processors, Intel® Xeon® 6 P-core processors, Intel® Xeon® 6 E-core processors, and Intel® Series 2 Core™ Ultra Processors,” including Table 2-52. For an MSR listed in Table 2-52 that also appears in the model-specific tables of prior generations, Table 2-52 supersedes prior generation tables.

Certain bit field positions may be related to the maximum physical address width, the value of which is expressed as “MAXPHYADDR” in Table 2-52. See Section 2.1 for an explanation of this value.

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register (R/W)		Core
27:0	Reserved.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 10.1.2.3, “Features to Disable Bus Locks.”	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 10.1.2.3, “Features to Disable Bus Locks.”	
31:30	Reserved.	
Register Address: A7H, 167	MSR_BIOS_DEBUG	
BIOS DEBUG (R/O) See Table 2-45.		Thread
Register Address: BCH, 188	IA32_MISC_PACKAGE_CTLs	
Power Filtering Control (R/W) IA32_ARCH_CAPABILITIES[bit 10] enumerates support for this MSR. See Table 2-2.		Package
Register Address: BFH, 191	IA32_PBOPT_CTRL	
IA32_PBOPT_OPT_CTRL If IA32_ARCH_CAPABILITIES[32] = 1 This MSR provides an architectural enumeration function for model-specific behavior.		
0	PREDICTION_CONTROL_BARRIER (R/W) When set to 0 (default), original Indirect Branch Predictor Barrier Target Array Return (IBPB TA RET) mitigation is enabled. When set to 1, alternative IBPB mitigation is enabled.	
63:1	Reserved.	
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	Reserved: returns zero.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Reserved: returns zero.	
2	INTEGRITY_CAPABILITIES When set to 1, the processor supports MSR_INTEGRITY_CAPABILITIES.	
3	RSM_IN_CPLD_ONLY Indicates that RSM will only be allowed in CPLD and will #GP for all non-CPLD privilege levels.	
4	UC_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 28 of MSR_MEMORY_CTRL (MSR address 33H).	
5	SPLIT_LOCK_DISABLE_SUPPORTED When read as 1, software can set bit 29 of MSR_MEMORY_CTRL.	
6	Reserved: returns zero.	
7	UC_STORE_THROTTLING_SUPPORTED Indicates that the snoop filter quality of service MSRs are supported on this core. This is based on the existence of a non-inclusive cache and the L2/MLC QoS feature supported.	
63:8	Reserved: returns zero.	
Register Address: E1H, 225	IA32_UWAIT_CONTROL	
UWAIT Control (R/W) See Table 2-2.		
Register Address: EDH, 237	MSR_RAR_CONTROL	
RAR Control (R/W)		Thread
29:0	Reserved.	
30	IGNORE_IF Allow RAR servicing at the RLP regardless of the value of RFLAGS.IF.	
31	ENABLE RAR events are recognized. When RAR is not enabled, RARs are dropped.	
63:32	Reserved.	
Register Address: EEH, 238	MSR_RAR_ACTION_VECTOR_BASE	
Pointer to RAR Action Vector (R/W)		Thread
5:0	Reserved.	
MAXPHYADDR-1:6	VECTOR_PHYSICAL_ADDRESS Pointer to the physical address of the 64B aligned RAR action vector.	
63:MAXPHYADDR	Reserved.	
Register Address: EFH, 239	MSR_RAR_PAYLOAD_TABLE_BASE	
Pointer to Base of RAR Payload Table (R/W)		Thread
11:0	Reserved.	
MAXPHYADDR-1:12	TABLE_PHYSICAL_ADDRESS Pointer to the base physical address of the 4K aligned RAR payload table.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:MAXPHYADDR	Reserved.	
Register Address: F0H, 240	MSR_RAR_INFO	
Read Only RAR Information (R/O)		Thread
31:0	Supported payload type bitmap. A value of 1 in bit position [i] indicates that payload type [i] is supported.	
37:32	Table Max Index Maximum supported payload table index.	
63:38	Always zero.	
Register Address: 105H, 261	MSR_CORE_BIST	
Core BIST (R/W) Controls Array BIST activation and status checking as part of FUSA.		Core
31:0	BIST_ARRAY Bitmap indicating which arrays to run BIST on (WRITE). Bitmap indicating which arrays were not processed, i.e., completion mask (READ).	
39:32	BANK Array bank of the [least significant set bit] array indicated in EAX to start BIST(WRITE). Array bank interrupted or failed (READ).	
47:40	DWORD Array dword of the [least significant set bit] array indicated in EAX to start BIST (WRITE). Array dword interrupted or failed (READ).	
62:48	Reserved.	
63	CTRL_RESULT Indicates whether WRMSR should signal Machine-Check upon BIST-error (WRITE). BIST result PASS(0)/FAIL(1) of the (least significant set bit) array indicated in EAX (READ).	
Register Address: 10AH, 266	IA32_ARCH_CAPABILITIES	
Enumeration of Architectural Features (R/O) See Table 2-2.		
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
Prefetch Disable Bits (R/W)		
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	Reserved.	
5	AMP_PREFETCH_DISABLE If 1, disables the L2 Adaptive Multipath Probability (AMP) prefetcher.	
63:6	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	
Primary Maximum Turbo Ratio Limit (R/W) See Table 2-46.		Package
Register Address: 1AEH, 430	MSR_TURBO_RATIO_LIMIT_CORES	
See Table 2-50.		Package
Register Address: 1C4H, 452	IA32_XFD	
Extended Feature Detect (R/W) See Table 2-2.		
Register Address: 1C5H, 453	IA32_XFD_ERR	
XFD Error Code (R/W) See Table 2-2.		
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
COPY_SCAN_HASHES (W)		Die
63:0	SCAN_HASH_ADDR Contains the linear address of the SCAN Test HASH Binary loaded into memory.	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
SCAN_HASHES_STATUS (R/O)		
15:0	CHUNK_SIZE Chunk size of the test in KB.	Die
23:16	NUM_CHUNKS Total number of chunks.	Die
31:24	Reserved: all zeros.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE The error-code refers to the LP that runs WRMSR(2C2H). 0x0: No error reported. 0x1: Attempt to copy scan-hashes when copy already in progress. 0x2: Secure Memory not set up correctly. 0x3: Scan-image header Image_info.ProgramID doesn't match RDMSR(2D9H)[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4: Reserved 0x5: Integrity check failed. 0x6: Re-install of scan test image attempted when current scan test image is in use by other LPs.	Thread
50:40	Reserved: set to all zeros.	
62:51	MAX_CORE_LIMIT Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	Die
63	Valid Valid bit is set when COPY_SCAN_HASHES has completed successfully.	Die
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
AUTHENTICATE_AND_COPY_CHUNK (W)		Die
7:0	CHUNK_INDEX Chunk Index, should be less than the total number of chunks defined by NUM_CHUNKS (MSR_SCAN_HASHES_STATUS[23:16]).	
63:8	CHUNK_ADDR Bits 63:8 of 256B aligned Linear address of scan chunk in memory.	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
CHUNKS_AUTHENTICATION_STATUS (R/O)		
7:0	VALID_CHUNKS Total number of Valid (authenticated) chunks.	Die
15:8	TOTAL_CHUNKS Total number of chunks.	Die
31:16	Reserved: all zeros.	
39:32	ERROR_CODE The error code refers to the LP that runs WRMSR(2C4H). 0x0: No error reported. 0x1: Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. 0x2: CHUNK authentication error. HASH of chunk did not match expected value.	Thread
63:40	Reserved: set to all zeros.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
ACTIVATE_SCAN (W)		Thread
7:0	CHUNK_START_INDEX Indicates chunk index to start from.	
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive).	
31:16	Reserved: all zeros.	
62:32	THREAD_WAIT_DELAY TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE If 1, then on scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. If 0, then no logging/no signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
SCAN_STATUS (R/O)		
7:0	CHUNK_NUM SCAN Chunk that was reached.	Core
15:8	CHUNK_STOP_INDEX Indicates what chunk index to stop at (inclusive). Maps to same field in WRMSR(ACTIVATE_SCAN).	Core
31:16	Reserved: return all zeros.	
39:32	ERROR_CODE 0x0: No error. 0x1: SCAN operation did not start. Other thread did not join in time. 0x2: SCAN operation did not start. Interrupt occurred prior to threads rendezvous. 0x3: SCAN operation did not start. Power Management conditions are inadequate to run Intel In-field Scan. 0x4: SCAN operation did not start. Non-valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5: SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6: SCAN operation did not start. Core not capable of performing SCAN currently. 0x8: SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Intel In-field Scan concurrently. MAX_CORE_LIMIT exceeded. 0x9: Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed.	Thread
61:40	Reserved: return all zeros.	
62	SCAN_CONTROL_ERROR Scan-System-Controller malfunction.	Core

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	SCAN_SIGNATURE_ERROR Core failed SCAN-SIGNATURE checking for this chunk.	Core
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
SCAN_MODULE_ID (R/O)		Module
31:0	RevID of the currently installed scan test image. Maps to Revision field in external header (offset 4).	
63:32	Reserved: return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
LAST_SAF_WP (R/O)		Core
31:0	LAST_WP Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. Available only if enumerated in MSR_INTEGRITY_CAPABILITIES[10:9].	
63:32	Reserved: return all zeros.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
INTEGRITY_CAPABILITIES (R/O)		Module
0	STARTUP_SCAN_BIST When set, supports Intel In-field Scan.	
3:1	Reserved: return all zeros.	
4	PERIODIC_SCAN_BIST When set, supports Intel In-field Scan.	
23:5	Reserved: return all zeros.	
31:24	ID of the scan programs supported for this part. WRMSR(2C2H) verifies this value against the corresponding value in the scan-image header, i.e., Image_info.	
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs," through Section 17.3.2.4, "IA32_MCi_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 413H, 1043	IA32_MC4_MISC	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Section 17.3.2.1, "IA32_MCI_CTL MSRs," through Section 17.3.2.4, "IA32_MCI_MISC MSRs." Bank MC4 reports MC errors from the PCU module. If SIGNAL_MCE is set, a Scan Status is logged in MC4_STATUS and MC4_MISC.		Package
Register Address: 492H, 1170	IA32_VMX_PROCBASED_CTL3	
Capability Reporting Register of Tertiary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		
Register Address: 493H, 1171	IA32_VMX_EXIT_CTL2	
Capability Reporting Register of Secondary VM-Exit Controls (R/O) See Table 2-2.		
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) Energy consumed by DRAM devices.		Package
31:0	Energy in 61 micro-joules. Requires BIOS configuration to enable DRAM RAPL mode 0 (Direct VR).	
63:32	Reserved.	
Register Address: 64DH, 1613	MSR_PLATFORM_ENERGY_STATUS	
Platform Energy Status (R/O)		Package
31:0	TOTAL_ENERGY_CONSUMED Total energy consumption in J (32.0), in 10nsec units.	
63:32	TIME_STAMP Time stamp (U32.0).	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W-L)		Package
16:0	POWER_LIMIT_1 The average power limit value that the platform must not exceed over a time window as specified by the Power_Limit_1_TIME field. The default value is the Thermal Design Power (TDP) and varies with product skus. The unit is specified in MSR_RAPL_POWER_UNIT.	
17	POWER_LIMIT_1_EN When set, the processor can apply control policies such that the platform average power does not exceed the Power_Limit_1 value over an exponential weighted moving average of the time window.	
18	CRITICAL_POWER_CLAMP_1 When set, the processor can go below the OS-requested P States to maintain the power below the specified Power_Limit_1 value.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
25:19	POWER_LIMIT_1_TIME This indicates the time window over which the Power_Limit_1 value should be maintained. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, and the unit is specified in MSR_RAPL_POWER_UNIT[Time Unit].	
31:26	Reserved.	
48:32	POWER_LIMIT_2 This is the Duration Power limit value that the platform must not exceed. The unit is specified in MSR_RAPL_POWER_UNIT.	
49	Enable Platform Power Limit #2 When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit #2 over the Short Duration time window.	
50	Platform Clamping Limitation #2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit #2 value.	
57:51	POWER_LIMIT_2_TIME This indicates the time window over which the Power_Limit_2 value should be maintained. This field has the same format as the POWER_LIMIT_1_TIME field.	
62:58	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 665H, 1637	MSR_PLATFORM_POWER_INFO	
Platform Power Information (R/W)		Package
16:0	MAX_PPL1 Maximum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
31:17	MIN_PPL1 Minimum PP L1 value. The unit is specified in MSR_RAPL_POWER_UNIT.	
48:32	MAX_PPL2 Maximum PP L2 value. The unit is specified in MSR_RAPL_POWER_UNIT.	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:49	MAX_TW Maximum time window. The unit is specified in MSR_RAPL_POWER_UNIT.	
62:56	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 666H, 1638	MSR_PLATFORM_RAPL_SOCKET_PERF_STATUS	
Platform RAPL Socket Performance Status (R/O)		Package
31:0	Count of limited performance due to platform RAPL limit.	
Register Address: 6A0H, 1696	IA32_U_CET	
Configure User Mode CET (R/W) See Table 2-2.		
Register Address: 6A2H, 1698	IA32_S_CET	
Configure Supervisor Mode CET (R/W) See Table 2-2.		
Register Address: 6A4H, 1700	IA32_PL0_SSP	
Linear address to be loaded into SSP on transition to privilege level 0. (R/W) See Table 2-2.		
Register Address: 6A5H, 1701	IA32_PL1_SSP	
Linear address to be loaded into SSP on transition to privilege level 1. (R/W) See Table 2-2.		
Register Address: 6A6H, 1702	IA32_PL2_SSP	
Linear address to be loaded into SSP on transition to privilege level 2. (R/W) See Table 2-2.		
Register Address: 6A7H, 1703	IA32_PL3_SSP	
Linear address to be loaded into SSP on transition to privilege level 3. (R/W) See Table 2-2.		
Register Address: 6A8H, 1704	IA32_INTERRUPT_SSP_TABLE_ADDR	
Linear address of a table of seven shadow stack pointers that are selected in IA-32e mode using the IST index (when not 0) from the interrupt gate descriptor. (R/W) See Table 2-2.		
Register Address: 6E1H, 1761	IA32_PKRS	
Specifies the PK permissions associated with each protection domain for supervisor pages (R/W) See Table 2-2.		
Register Address: 776H, 1910	IA32_HWP_CTL	
See Table 2-2.		
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
Memory Encryption Capability MSR See Table 2-2.		

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.		
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		
Register Address: 988H, 2440	IA32_UINTR_MISC	
User-Interrupt Target-Table Size and Notification Vector (R/W) See Table 2-2.		
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		
Register Address: C70H, 3184	MSR_B1_PMON_EVNT_SELO	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C71H, 3185	MSR_B1_PMON_CTR0	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C72H, 3186	MSR_B1_PMON_EVNT_SEL1	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C73H, 3187	MSR_B1_PMON_CTR1	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C74H, 3188	MSR_B1_PMON_EVNT_SEL2	
Uncore B-box 1 PerfMon event select MSR.		Package
Register Address: C75H, 3189	MSR_B1_PMON_CTR2	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C76H, 3190	MSR_B1_PMON_EVNT_SEL3	
Uncore B-box 1vPerfMon event select MSR.		Package
Register Address: C77H, 3191	MSR_B1_PMON_CTR3	
Uncore B-box 1 PerfMon counter MSR.		Package
Register Address: C82H, 3122	MSR_W_PMON_BOX_OVF_CTRL	
Uncore W-box PerfMon local box overflow control MSR.		Package
Register Address: C8FH, 3215	IA32_PQR_ASSOC	

Table 2-52. Additional MSRs Supported by the 4th and 5th Generation Intel® Xeon® Scalable Processor Families (CPUID Signature DisplayFamily_DisplayModel Values of 06_8FH and 06_CFH) (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		
Register Address: C90H—C9EH, 3216—3230	IA32_L3_QOS_MASK_0 through IA32_L3_QOS_MASK_14	
See Table 2-50.		Package
Register Address: D10H—D17H, 3344—3351	IA32_L2_QOS_MASK_[0-7]	
IA32_CR_L2_QOS_MASK_[0-7] If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. See Table 2-2.		Core
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		
Register Address: 1200H—121FH, 4608—4639	IA32_LBR_x_INFO	
Last Branch Record Entry X Info Register (R/W) See Table 2-2.		
Register Address: 1406H, 5126	IA32_MCU_CONTROL	
See Table 2-2.		
Register Address: 14CEH, 5326	IA32_LBR_CTL	
Last Branch Record Enabling and Configuration Register (R/W) See Table 2-2.		
Register Address: 14CFH, 5327	IA32_LBR_DEPTH	
Last Branch Record Maximum Stack Depth Register (R/W) See Table 2-2.		
Register Address: 1500H—151FH, 5376—5407	IA32_LBR_x_FROM_IP	
Last Branch Record Entry X Source IP Register (R/W) See Table 2-2.		
Register Address: 1600H—161FH, 5632—5663	IA32_LBR_x_TO_IP	
Last Branch Record Entry X Destination IP Register (R/W) See Table 2-2.		

2.17.9 MSRs Introduced in the Intel® Core™ Ultra 7 Processor Supporting Performance Hybrid Architecture

Table 2-53 lists additional MSRs for the Intel Core Ultra 7 processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_AAH. Table 2-54 lists the MSRs unique to the processor P-core. Table 2-55 lists the MSRs unique to the processor E-core.

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CTRL	
Memory Control Register		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor will only allow one in-progress UC store at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC lock will cause a #GP(0) exception. See Section 10.1.2.3, "Features to Disable Bus Locks."	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will cause an #AC(0) exception. See Section 10.1.2.3, "Features to Disable Bus Locks."	
63:30	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
Feature Activation (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread. See Table 2-2.		
Register Address: 80H, 128	MSR_TRACE_HUB_STH ACPIBAR_BASE	
MSR_TRACE_HUB_STH ACPIBAR_BASE (R/W) This register is used by BIOS to program Trace Hub STH base address that will be used by AET messages.		Thread
0	LOCK Lock bit. If set, this MSR cannot be re-written anymore. The lock bit has to be set in order for the AET packets to be directed to Trace Hub MMIO.	
17:1	Reserved.	
45:18	ADDRESS AET target address in Trace Hub MMIO space.	
63:46	Reserved.	
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration (R/W)		Core

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3:0	PKG_C_STATE_LIMIT Specifies the lowest processor-specific C-state code name (consuming the least power) for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings may be supported: 0000b: C0/C1 (no package C-state support) 0001b: C2 0010b: C3 0011b: C6 0100b: C7 0101b: C7s 0110b: C8 0111b: C9 1000b: C10	
7:4	MAX_CORE_C_STATE Possible values are: 0000—reserved; 0001—C1; 0010—C3, 0011—C6.	
9:8	Reserved.	
10	IO_MWAIT_REDIRECTION_ENABLE When set, will map IO_read instructions sent to IO registers PMG_IO_BASE_ADDR.PMB0+0/1/2 to MWAIT(C2,3,4) instructions; applies to deepc4 too.	
14:11	Reserved.	
15	CFG_LOCK When set, locks bits 15:0 of this register for further writes, until the next reset occurs.	
24:16	Reserved.	
25	C3_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.	
26	C1_STATE_AUTO_DEMOTION_ENABLE When set, processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	ENABLE_C3_UNDEMOTION Enable Un-Demotion from Demoted C3.	
28	ENABLE_C1_UNDEMOTION Enable Un-Demotion from Demoted C1.	
29	ENABLE_PKGC_AUTODEMOTION Enable Package C-State Auto-Demotion. It enables use of the history of past package C-state depth and residence, as a factor in determining C-State depth.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	ENABLE_PKGC_UNDEMOTION Enable Package C-State Un-Demotion. It enables considering cases where demotion was the incorrect decision in determining C-State depth.	
31	TIMED_MWAIT_ENABLE When set, enables Timed MWAIT feature. MWAIT would #GP on attempts to do setup MWAIT timer if this bit is not set.	
63:32	Reserved.	
Register Address: E4H, 228	MSR_IO_CAPTURE_BASE	
IO Capture Base (R/W) Power Management IO Redirection in C-state. See http://biosbits.org .		Core
15:0	LVL_2_BASE_ADDRESS Specifies the base address visible to software for IO redirection. If MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.	
18:16	CST_RANGE Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PKG_CST_CONFIG_CONTROL.IO_MWAIT_REDIRECTION_ENABLE: 000b—C3 is the max C-State to include. 001b—C6 is the max C-State to include. 010b—C7 is the max C-State to include.	
63:19	Reserved.	
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Feature Configuration (R/W)		Core
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
1	AESNI_DISABLE This bit disables Advanced Encryption Standard feature on this processor core. To disable AES, BIOS will write '11 to this MSR on every core.	
63:2	Reserved.	
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Feature Enable (R/W) Miscellaneous enables for thread specific features.		Thread
0	CPUID_GP_ON_CPL_GT_0 Causes CPUID to #GP if CPL greater than 0 and not in SMM.	
63:1	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target (R/W) Legacy register holding temperature related constants for Platform use.		Package
6:0	TCC Offset Time Window Describes the RATL averaging time window.	
7	TCC Offset Clamping Bit When enabled will allow RATL throttling below P1.	
15:8	Temperature Control Offset Fan Temperature Target Offset (a.k.a. T-Control) indicates the relative offset from the Thermal Monitor Trip Temperature at which fans should be engaged.	
23:16	TCC Activation Temperature The minimum temperature at which PROCHOT# will be asserted. The value is degrees C.	
30:24	TCC Activation Offset Specifies a temperature offset in degrees C from the temperature target (bits 23:16). PROCHOT# will assert at the offset target temperature. Write is permitted only if MSR_PLATFORM_INFO[30] is set.	
31	LOCKED When set, this entire register becomes read-only.	
63:2	Reserved.	
Register Address: 1A4H, 420	MSR_PREFETCH_CONTROL	
PREFETCH Control (R/W) Prefetch disable bits.		Thread
0	L2_HARDWARE_PREFETCHER_DISABLE If 1, disables the L2 hardware prefetcher, which fetches additional lines of code or data into the L2 cache.	
1	L2_ADJACENT_CACHE_LINE_PREFETCHER_DISABLE If 1, disables the adjacent cache line prefetcher, which fetches the cache line that comprises a cache line pair (128 bytes).	
2	DCU_HARDWARE_PREFETCHER_DISABLE If 1, disables the L1 data cache prefetcher, which fetches the next cache line into L1 data cache.	
3	DCU_IP_PREFETCHER_DISABLE If 1, disables the L1 data cache IP prefetcher, which uses sequential load history (based on instruction pointer of previous loads) to determine whether to prefetch additional lines.	
4	DCU_NEXT_PAGE_PREFETCH_DISABLE If 1, disables Next Page prefetcher.	
5	AMP_PREFETCH_DISABLE If 1, disables L2 Adaptive Multipath Probability (AMP) prefetcher.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
6	LLC_PAGE_PREFETCH_DISABLE If 1, disables the LLC Page prefetcher.	
7	AOP_PREFETCH_DISABLE	
8	STREAM_PREFETCH_CODE_FETCH_DISABLE	
63:9	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
OFFCORE_RSP_0 (R/W) Offcore Response Event Select Register		Thread
0	TRUE_DEMAND_CACHE_LOAD Demand Data Rd = DCU reads (includes partials) that is not tagged homeless.	
1	DEMAND_RFO Demand Instruction fetch = IFU Fetches. ItoM or RFO that is not tagged homeless.	
2	DEMAND_CODE_READ Demand Instruction fetch = IFU Fetches. CRd or CRd_UC.	
3	CORE_MODIFIED_WRITEBACK WBMtoI or WBMtoE.	
4	HW_PREFETCH_MLC_LOAD L2 prefetcher requests triggered by reads from MEC (except those triggered by I-side).	
5	HW_PREFETCH_MLC_RFO L2 prefetcher requests triggered by RFOs.	
6	HW_PREFETCH_MLC_CODE L2 prefetcher requests triggered by I-side requests.	
7	HW_PREFETCH_LLC_LOAD LLC prefetch requests triggered by DRd.	
8	HW_PREFETCH_LLC_RFO LLC prefetch requests triggered by RFO.	
9	HW_PREFETCH_LLC_CODE LLC prefetch requests triggered by CRd.	
10	L1_HWPREFETCH Covers Hardware PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
11	ALL_STREAMING_STORE Write Combining. WCIL or WCILF.	
12	CORE_NON_MODIFIED_WB WBEFtoI or WBEFtoE.	
13	LLC_PREFETCH LLC prefetch of load/code/RFO.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
14	L1_SWPREFETCH Covers Software PFRFO, PFNEAR, PFMED, PFFAR, PFHW, PFNTA, PFNPP, PFIPP including the homeless versions.	
15	OTHER Includes CLFlush, CLFlushOPT, CLDemote, CLWB, Enqueue SetMonitor, PortIn, IntA, Lock, SplitLock, Unlock, SpCyc, ClrMonitor, PortOut, IntPriUp, IntLog, IntPhy, EOI, RdCurr, WbStol, LLCWBInv, LLCInv, NOP, PCOMMIT.	
16	ANY_RESP Match on any response.	
17	SUPPLIER_NONE No Supplier Details. DATA_PRE [6:3] = 0.	
18	LLC_HIT_M_STATE LLC/L3, M-state, DATA_PRE [6:3] = 2.	
19	LLC_HIT_E_STATE LLC/L3, E-state, DATA_PRE [6:3] = 4.	
20	LLC_HIT_S_STATE LLC/L3, S-state, DATA_PRE [6:3] = 6.	
21	LLC_HIT_F_STATE LLC/L3, F-state, DATA_PRE [6:3] = 8.	
22	FAR_MEM_LOCAL Far Memory, Local, DATA_PRE [6:3] = 1.	
23	FAR_MEM_REMOTE_0_HOP Far Memory, Remote 0-hop, DATA_PRE [6:3] = 3.	
24	FAR_MEM_REMOTE_1_HOP Far Memory, Remote 1-hop, DATA_PRE [6:3] = 5.	
25	FAR_MEM_REMOTE_2_PLUS_HOP Far Memory, Rem 2+ hop, DATA_PRE [6:3] = 7.	
26	NEAR_MEM_MISS_LOCAL_NODE LLC Miss Local Node. Near Memory, Local DATA_PRE [6:3] = E.	
27	NEAR_MEM_REMOTE_0_HOP Near Memory, Remote 0-hop, DATA_PRE [6:3] = B	
28	NEAR_MEM_REMOTE_1_HOP Near Memory, Remote 1-hop, DATA_PRE [6:3] = D.	
29	NEAR_MEM_REMOTE_2_PLUS_HOP Near Memory, Remote 2+ hop, DATA_PRE [6:3] = F.	
30	SPL_HIT Snoop Info: SPL-hit, DATA_PRE [2:0] = 6.	
31	SNOOP_NONE No details as to Snoop-related info. Snoop Info: None, DATA_PRE [2:0] = 0.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
32	NOT_NEEDED No snoop was needed to satisfy the request. Snoop Info: Not needed, DATA_PRE [2:0] = 1.	
33	MISS No snoop was needed to satisfy the request. Snoop Info: Miss, DATA_PRE [2:0] = 2.	
34	HIT_NO_FWD A snoop was needed and it Hits in at least one snooped cache. Hit denotes a cache-line was valid before snoop effect. Snoop Info: Hit No Fwd, DATA_PRE [2:0] = 3.	
35	HIT_EF_WITH_FWD A snoop was needed and data was Forwarded from a remote socket. Snoop Info: Hit EF w/Fwd, DATA_PRE [2:0] = 4.	
36	HITM A snoop was needed and it HitMed in local or remote cache. HitM denotes a cache-line was modified before snoop effect. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
37	NON_DRAM Target was non-DRAM system address. Snoop Info: HitM, DATA_PRE [2:0] = 5.	
38	GO_ERR GO-ERR, RspData[3:0] = 0100.	
39	GO_NO_GO GO-NoGO, RspData[3:0] = 0111.	
40	INPKG_MEM_LOCAL In-package Memory, Local, DATA_PRE [6:3] = 9.	
41	INPKG_MEM_NONLOCAL In-package Memory, Non-Local, DATA_PRE [6:3] = C.	
43:42	Reserved.	
44	UC_LOAD PRd or UCRdF.	
45	UC_STORE WiL.	
46	PARTIAL_STREAMING_STORES WCiL.	
47	FULL_STREAMING_STORES WCiLF.	
48	L1_MODIFIED_WB EVICTIION EXTTYPE from MEC.	
49	L2_MODIFIED_WB WBMtol or WBMtoE.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
50	PSMI MemPushWr_NS (PSMI only).	
51	ITOM ItoM.	
63:52	Reserved.	
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
OFFCORE_RSP_1 (R/W) Offcore Response Event Select Register. See MSR_OFFCORE_RSP_0 (at 1A6H).		Thread
Register Address: 1AAH, 426	MSR_MISC_PWR_MGMT	
Miscellaneous Power Management Control (R/W) Various model-specific features enumeration. See http://biosbits.org .		Package
0	Reserved.	
1	ENABLE_HWP_VOTING_RIGHT When set (1), The CPU will take into account thread HWP requests for threads that have voting rights only (ignores thread requests if they do not have voting rights). When reset(0), The CPU will take into account all thread HWP requests, even for threads that don't have voting rights. Setting this bit will cause the HWP Base feature bit to be reported in CPUID as present; clearing will cause it to be reported as non-present.	
5:2	Reserved.	
6	ENABLE_HWP Setting this bit will cause the HWP Base feature bit to report as present in CPUID; clearing this bit will cause CPUID to report the feature as non-present.	
7	ENABLE_HWP_INTERRUPT Setting this bit will cause the HWP Interrupt feature CPUID.06H:EAX[8] bit to report as present; clearing will report as non-present.	
8	ENABLE_OUT_OF_BAND_AUTONOMOUS Setting this bit will cause the HWP Autonomous feature bit to report as present; clearing will report as non-present.	
11:9	Reserved.	
12	ENABLE_HWP_EPP Enable HWP EPP. Setting this bit (1) will cause the HWP CPUID.06H:EAX[10] Energy Performance Preference bit to report as present (1); clearing will report as non-present (0).	
13	LOCK Setting this bit will prevent the BIOS specific bits from changing until the next reset. i.e., only Bits [0,22] which are meant for OS use can be changed once the LOCK bit is set.	
63:14	Reserved.	
Register Address: 1ADH, 429	MSR_PRIMARY_TURBO_RATIO_LIMIT	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Primary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 1F1H, 497	MSR_CRASHLOG_CONTROL	
Crash Log Control (R/W) Write data to a Crash Log configuration.		Thread
0	CDDIS CrashDump_Disable: If set, indicates that Crash Dump is disabled.	
1	EN_GPRS Collect GPRs on a crash dump. Only meaningful when CDDIS is zero.	
2	EN_GPRS_IN_SMM Collect GPRs in SMM on a crash dump. Only meaningful when CDDIS is zero. EN_GPRS will override this control,	
3	TRIPLE_FAULT_SHUTDOWN Collect a crash log on a triple fault shutdown. Only meaningful when CDDIS is zero.	
63:4	Reserved.	
Register Address: 1F5H, 501	MSR_PRMRR_PHYS_MASK	
Processor Reserved Memory Range Register - Physical Mask (R/W)		Core
9:0	Reserved.	
10	LOCK Once set, this bit prevents software from modifying the PRMRR.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
11	VALID This bit serves as the enable for the PRMRR; the PRMRR must be LOCKed before it can be enabled.	
19:12	Reserved.	
45:20	MASK PRMRR Address Mask.	
63:46	Reserved.	
Register Address: 1FCH, 508	MSR_POWER_CTL	
Power Control Register (R/W) See http://biosbits.org .		Package
0	ENABLE_BIDIR_PROCHOT Used to enable or disable the response to PROCHOT# input. When set/enabled, the platform can force the CPU to throttle to a lower power condition such as Pn/Pm by asserting proshot#. When clear/disabled (default), the CPU ignores the status of the proshot input signal.	
1	C1E_ENABLE When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).	
2	SAPM_IMC_C2_POLICY This bit determines if self-refresh activation is allowed when entering Package C2 State. If it is set to 0b, PCODE will keep the FORCE_SR_OFF bit asserted in Package C2 State and allow its negation according to the defined latency negotiations with the PCH and Display Engine in Package C3 and deeper states. Otherwise, self-refresh is allowed in Package C2 State.	
3	FAST_BRK_SNP_EN This bit controls the VID swing rate for the OTHER_SNP_WAKE events that are detected by the iMPH. This is the event that is detected by the iMPH when a non-DMI snoopable request is observed while UCLK domain is not functional. 0b: Use slow VID swing rate. 1b: Use fast VID swing rate.	
17:4	Reserved.	
18	PWR_PERF_PLTFRM_OVR Power performance platform override.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
19	EE_TURBO_DISABLE Setting this bit disables the P-States energy efficiency optimization. Default value is 0. Disable/enable the energy efficiency optimization in P-State legacy mode (when IA32_PM_ENABLE[HWP_ENABLE] = 0), has an effect only in the turbo range or into PERF_MIN_CTL value if it is not zero set. In HWP mode (IA32_PM_ENABLE[HWP_ENABLE] == 1), has an effect between the OS desired or OS maximize to the OS minimize performance setting.	
20	RTH_DISABLE Setting this bit disables the Race to Halt optimization and avoids this optimization limitation to execute below the most efficient frequency ratio. Default value is 0 for processors that support Race to Halt optimization.	
21	DIS_PROCHOT_OUT Prochot output disable.	
22	PROCHOT_RESPONSE Prochhot configurable response enable.	
23	VR_THERM_ALERT_DISABLE_LOCK When set to 1, locks PROCHOT related bits of this MSR. Once set, a reset is required to clear this bit.	
24	VR_THERM_ALERT_DISABLE When set to 1, disables the VR_THERMAL_ALERT signaling.	
25	DISABLE_RING_EE Disable Ring EE.	
26	DISABLE_SA_OPTIMIZATION Disable SA optimization.	
27	DISABLE_OOK Disable OOK.	
28	DISABLE_AUTONOMOUS Disable HWP autonomous mode.	
29	Reserved.	
30	CSTATE_PREWAKE_DISABLE C-state pre-wake disable.	
63:31	Reserved.	
Register Address: 2A0H, 672	MSR_PRMRR_BASE_0	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
45:20	BASE PRMRR base address.	
63:46	Reserved.	
Register Address: 474H, 1140	IA32_MC29_CTL	
MC29_CTL. See Table 2-2.		Package
Register Address: 475H, 1141	IA32_MC29_STATUS	
MC29_STATUS. See Table 2-2.		Package
Register Address: 476H, 1142	IA32_MC29_ADDR	
MC29_ADDR. See Table 2-2.		Package
Register Address: 477H, 1143	IA32_MC29_MISC	
MC29_MISC. See Table 2-2.		Package
Register Address: 478H, 1144	IA32_MC30_CTL	
MC30_CTL. See Table 2-2.		Package
Register Address: 479H, 1145	IA32_MC30_STATUS	
MC30_STATUS. See Table 2-2.		Package
Register Address: 47AH, 1146	IA32_MC30_ADDR	
MC30_ADDR. See Table 2-2.		Package
Register Address: 47BH, 1147	IA32_MC30_MISC	
MC30_MISC. See Table 2-2.		Package
Register Address: 47CH, 1148	IA32_MC31_CTL	
MC31_CTL. See Table 2-2.		Package
Register Address: 47DH, 1149	IA32_MC31_STATUS	
MC31_STATUS. See Table 2-2.		Package
Register Address: 47EH, 1150	IA32_MC31_ADDR	
MC31_ADDR. See Table 2-2.		Package
Register Address: 47FH, 1151	IA32_MC31_MISC	
MC31_MISC. See Table 2-2.		Package
Register Address: 4E0H, 1248	MSR_SMM_FEATURE_CONTROL	
Enhanced SMM Feature Control (R/W) Reports SMM capability enhancement.		Package
0	LOCK When set, locks this register from further changes.	
1	SMM_CPU_SAVE_EN If 0, SMI/RSM will save/restore state in SMRAM If 1, SMI/RSM will save/restore state from SRAM.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	SMM_CODE_CHK_EN When clear (default) none of the logical processors are prevented from executing SMM code outside the ranges defined by the SMRR. When set, any logical processor in the package that attempts to execute SMM code not within the ranges defined by the SMRR will assert an unrecoverable MCE.	
63:3	Reserved.	
Register Address: 601H, 1537	MSR_VR_CURRENT_CONFIG	
Power Limit 4 (PL4) (R/W) Package-level maximum power limit (in Watts). It is a proactive, instantaneous limit.		Package
15:0	CURRENT_LIMIT PL4 Value in 0.125 A increments. This field is locked by MSR_VR_CURRENT_CONFIG.LOCK. When the LOCK bit is set to 1, this field becomes Read Only.	
30:16	Reserved.	
31	LOCK This bit will lock the CURRENT_LIMIT settings in this register and will also lock this setting. This means that once set to 1, the CURRENT_LIMIT setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 620H, 1568	MSR_UNCORE_RATIO_LIMIT	
Uncore Ratio Limit (R/W) Min/Max Ratio Limits for Uncore LLC and Ring.		Package
6:0	MAX_CLR_RATIO Maximum allowed ratio for the Ring and Last Level Cache (LLC).	
7	Reserved.	
14:8	MIN_CLR_RATIO Minimum allowed ratio for the Ring and Last Level Cache (LLC).	
63:15	Reserved.	
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
MSR_PPO_POWER_LIMIT (R/W) PPO RAPL power unit control.		Package
14:0	IA_PP_PWR_LIM This is the power limitation on the IA cores power plane. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[PWR_UNIT].	
15	PWR_LIM_CTRL_EN This bit must be set in order to limit the power of the IA cores power plane. 0b: IA cores power plane power limitation is disabled. 1b: IA cores power plane power limitation is enabled.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	PP_CLAMP_LIM Power Plane Clamping limitation; allow going below P1. 0b: PBM is limited between P1 and P0. 1b: PBM can go below P1.	
23:17	CTRL_TIME_WIN x = CTRL_TIME_WIN[23:22] y = CTRL_TIME_WIN[21:17] The timing interval window is Floating Point number given by $1.x * \text{power}(2,y)$. The unit of measurement is defined in PACKAGE_POWER_SKU_UNIT_MSR[TIME_UNIT]. The maximal time window is bounded by PACKAGE_POWER_SKU_MSR[PKG_MAX_WIN]. The minimum time window is 1 unit of measurement (as defined above).	
30:24	Reserved.	
31	PP_PWR_LIM_LOCK When set, all settings in this register are locked and are treated as Read Only.	
63:32	Reserved.	
Register Address: 64FH, 1615	MSR_CORE_PERF_LIMIT_REASONS	
Core Performance Limit Reasons Indicator of Frequency Clipping in Processor Cores. (Frequency refers to processor core frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced below the operating system request due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced below the operating system request due to a thermal event.	
3:2	Reserved.	
4	RSR_LIMIT (R/O) Residency State Regulation Status. When set, frequency is reduced below the operating system request due to residency state regulation limit.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced below the operating system request due to Running Average Thermal Limit (RATL).	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced below the operating system request due to a thermal alert from a processor Voltage Regulator (VR).	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
7	VR_TDC (R/O) VR Therm Design Current Status. When set, frequency is reduced below the operating system request due to VR thermal design current limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced below the operating system request due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced below the operating system request due to package/platform-level power limiting PL2/PL3.	
12	MAX_TURBO_LIMIT (R/O) Max Turbo Limit Status. When set, frequency is reduced below the operating system request due to multi-core turbo limits.	
13	TURBO_ATTEN (R/O) Turbo Transition Attenuation Status. When set, frequency is reduced below the operating system request due to Turbo transition attenuation. This prevents performance degradation due to frequent operating ratio changes.	
15:14	Reserved.	
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
19:18	Reserved.	
20	RSR_LIMIT_LOG (R/W) Residency State Regulation Log. When set, indicates that the Residency State Regulation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
21	RATL_LOG (R/W) Running average thermal limit Log, RW, When set by PCODE indicates that Running average thermal limit has cause IA frequency clipping. Software should write to this bit to clear the status in this bit.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR TDC Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the Other Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package or Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package or Platform Level PL2/PL3 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	MAX_TURBO_LIMIT_LOG (R/W) Max Turbo Limit Log. When set, indicates that the Max Turbo Limit Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
29	TURBO_ATTEN_LOG (R/W) Turbo Transition Attenuation Log. When set, indicates that the Turbo Transition Attenuation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:30	Reserved.	
Register Address: 650H, 1616	MSR_SECONDARY_TURBO_RATIO_LIMIT	
Secondary Maximum Turbo Ratio Limit (R/W) Software can configure these limits when MSR_PLATFORM_INFO[28] = 1. Specifies Maximum Ratio Limit for each group. Maximum ratio for groups with more cores must decrease monotonically.		Package
7:0	MAX_TURBO_GROUP_0: Maximum turbo ratio limit with 1 core active.	
15:8	MAX_TURBO_GROUP_1: Maximum turbo ratio limit with 2 cores active.	
23:16	MAX_TURBO_GROUP_2: Maximum turbo ratio limit with 3 cores active.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	MAX_TURBO_GROUP_3: Maximum turbo ratio limit with 4 cores active.	
39:32	MAX_TURBO_GROUP_4: Maximum turbo ratio limit with 5 cores active.	
47:40	MAX_TURBO_GROUP_5: Maximum turbo ratio limit with 6 cores active.	
55:48	MAX_TURBO_GROUP_6: Maximum turbo ratio limit with 7 cores active.	
63:56	MAX_TURBO_GROUP_7: Maximum turbo ratio limit with 8 cores active.	
Register Address: 65CH, 1628	MSR_PLATFORM_POWER_LIMIT	
Platform Power Limit Control (R/W) Allows platform BIOS to limit power consumption of the platform devices to the specified values. The Long Duration power consumption is specified via Platform_Power_Limit_1 and Platform_Power_Limit_1_Time. The Short Duration power consumption limit is specified via the Platform_Power_Limit_2 with duration chosen by the processor. The processor implements an exponential-weighted algorithm in the placement of the time windows.		Package
14:0	POWER_LIMIT_1 Average Power limit value which the platform must not exceed over a time window as specified by Power_Limit_1_TIME field. The default value is the Thermal Design Power (a.k.a TDP) and varies with product skus. The unit is specified in MSR_RAPLPOWER_UNIT.	
15	POWER_LIMIT_1_EN When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit 1 over the time window specified by Power Limit 1 Time Window.	
16	CRITICAL_POWER_CLAMP_1 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit 1 value.	
23:17	POWER_LIMIT_1_TIME Specifies the duration of the time window over which Platform Power Limit 1 value should be maintained for sustained long duration. This field is made up of two numbers from the following equation: Time Window = (float) ((1+(X/4))*(2^Y)), where: X = POWER_LIMIT_1_TIME[23:22] Y = POWER_LIMIT_1_TIME[21:17] The maximum allowed value in this field is defined in MSR_PKG_POWER_INFO[PKG_MAX_WIN]. The default value is 0DH, The unit is specified in MSR_RAPLPOWER_UNIT[Time Unit]	
31:24	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
46:32	POWER_LIMIT_2 Average Power limit value which the platform must not exceed over the Short Duration time window chosen by the processor. The recommended default value is 1.25 times the Long Duration Power Limit (i.e., Platform Power Limit 1).	
47	POWER_LIMIT_2_EN When set, enables the processor to apply control policy such that the platform power does not exceed Platform Power limit 2 over the Short Duration time window.	
48	CRITICAL_POWER_CLAMP_2 When set, allows the processor to go below the OS requested P states in order to maintain the power below specified Platform Power Limit 2 value.	
62:49	Reserved.	
63	LOCK Setting this bit will lock all other bits of this MSR until system RESET.	
Register Address: 6BOH, 1712	MSR_GRAPHICS_PERF_LIMIT_REASONS	
MSR_GRAPHICS_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Processor Graphics. (Frequency refers to processor graphics frequency.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
12	INEFFICIENT_OPERATION (R/O) Inefficient Operation Status. When set, processor graphics frequency is operating below target frequency.	
15:13	Reserved.	
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log. When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	RATL_LOG (R/W) Running Average Thermal Limit Log. When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
28	INEFFICIENT_OPERATION_LOG (R/W) Inefficient Operation Log. When set, indicates that the Inefficient Operation Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:29	Reserved.	
Register Address: 6B1H, 1713	MSR_RING_PERF_LIMIT_REASONS	
MSR_RING_PERF_LIMIT_REASONS Indicator of Frequency Clipping in the Ring Interconnect. (Frequency refers to ring interconnect in the uncore.)		Package
0	PROCHOT (R/O) PROCHOT Status. When set, frequency is reduced due to assertion of external PROCHOT.	
1	THERMAL (R/O) Thermal Status. When set, frequency is reduced due to a thermal event.	
4:2	Reserved.	
5	RATL (R/O) Running Average Thermal Limit Status. When set, frequency is reduced due to running average thermal limit.	
6	VR_THERMALERT (R/O) VR Therm Alert Status. When set, frequency is reduced due to a thermal alert from a processor Voltage Regulator.	
7	VR_TDC (R/O) VR Thermal Design Current Status. When set, frequency is reduced due to VR TDC limit.	
8	OTHER (R/O) Other Status. When set, frequency is reduced due to electrical or other constraints.	
9	Reserved.	
10	PBM_PL1 (R/O) Package/Platform-Level Power Limiting PL1 Status. When set, frequency is reduced due to package/platform-level power limiting PL1.	
11	PBM_PL2 (R/O) Package/Platform-Level PL2 Power Limiting Status. When set, frequency is reduced due to package/platform-level power limiting PL2/PL3.	
15:12	Reserved.	

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
16	PROCHOT_LOG (R/W) PROCHOT Log. When set, indicates that the PROCHOT Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
17	THERMAL_LOG (R/W) Thermal Log. When set, indicates that the Thermal Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
20:18	Reserved.	
21	RATL_LOG (R/W) Running Average Thermal Limit Log. When set, indicates that the RATL Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
22	VR_THERMALERT_LOG (R/W) VR Therm Alert Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
23	VR_TDC_LOG (R/W) VR Thermal Design Current Log. When set, indicates that the VR Therm Alert Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
24	OTHER_LOG (R/W) Other Log. When set, indicates that the OTHER Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
25	Reserved.	
26	PBM_PL1_LOG (R/W) Package/Platform-Level PL1 Power Limiting Log. When set, indicates that the Package/Platform Level PL1 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
27	PBM_PL2_LOG (R/W) Package/Platform-Level PL2 Power Limiting Log. When set, indicates that the Package/Platform Level PL2 Power Limiting Status bit has asserted since the log bit was last cleared. This log bit will remain set until cleared by software writing 0.	
63:28	Reserved.	
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core

Table 2-53. Additional MSRs Supported by the Intel® Core™ Ultra 7 Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field and must be zero on a WRMSR.	
63:40	Reserved.	

The MSRs listed in Table 2-54 are unique to the Intel Core Ultra 7 processor P-core. These MSRs are not supported on the processor E-core.

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W)		Thread
47:0	FIXED_COUNTER Top-down Microarchitecture Analysis unhalting number of available slots counter.	
63:48	Reserved.	
Register Address: 329H, 809	MSR_PERF_METRICS	
Performance Metrics (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	
23:16	FRONTEND_BOUND Percent of Unutilized slots where Front-end did not deliver a uop while Back-end is ready.	
31:24	BACKEND_BOUND Percent of Unutilized slots where a uop was not delivered to Back-end due to lack of Back-end resources.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	

Table 2-54. MSRs Supported by the Intel® Core™ Ultra 7 Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W) See Table 2-47.		Thread
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W) See Table 2-44.		Core

The MSRs listed in Table 2-48 are unique to the Intel Core Ultra 7 processor E-core. These MSRs are not supported on the processor P-core.

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
SAF Control (W/O) Extension to SAF.		Package
0	INVALIDATE_CURRENT_STRIDE Invalidate all chunks in current stride.	
63:1	Reserved.	
Register Address: D18H–D1FH, 3352–3359	IA32_L2_MASK_[8-15]	
IA32_L2_MASK_[8-15] (R/W) If CPUID.10H.01H:EDX.CAT_MAX_CLOS[15:0] ≥ 0. Controls MLC (L2) Intel RDT allocation. For more details on CAT/RDT, see Chapter 19, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features.”		Module
15:0	WAY_MASK Capacity Bit Mask. Available ways vectors for class of service of IA core. ‘1 in bit indicates allocation to the way is allowed. ‘0 indicates allocation to the way is not allowed.	
31:16	Reserved.	
Register Address: 1309H–130BH, 4873–4875	MSR_RELOAD_FIXED_CTRx	
Reload value for IA32_FIXED_CTRx (R/W)		Thread
47:0	Value loaded into IA32_FIXED_CTRx when a PEBS record is generated while PEBS_EN_FIXEDx = 1 and PEBS_OUTPUT = 01B in IA32_PEBES_ENABLE, and FIXED_CTRx is overflowed.	
63:48	Reserved.	
Register Address: 14C1H–14C8H, 5313 –5320	MSR_RELOAD_PMCx	
Reload value for IA32_PMCx (R/W)		Thread

Table 2-55. MSRs Supported by the Intel® Core™ Ultra 7 Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
47:0	Value loaded into IA32_PMCx when a PEBS record is generated while PEBS_EN_PMCx = 1 and PEBS_OUTPUT = 01B in IA32_PEBS_ENABLE, and PMCx is overflowed.	
63:48	Reserved.	
Register Address: 1A8EH, 6798	MSR_STLB_FILL_TRANSLATION	
STLB Fill Translation (W/O) STLB QoS MSR to fill translations into STLB.		Core
3:0	CLOS Class of service to use for the fill.	
9:4	Reserved.	
10	X Set to 1 when LA is to an executable page.	
11	RW Set to 1 when LA is to a writeable page.	
63:12	LA Logical address to use for fill.	

2.17.10 MSRs Introduced in the Intel® Xeon® 6 P-Core Processors

Table 2-56 lists additional MSRs for the Intel Xeon 6 P-core processors. Intel Xeon 6 P-core processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_ADH or 06_AEH.

For an MSR listed in Table 2-56 that also appears in the model-specific tables of prior generations, Table 2-56 supersedes prior generation tables.

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 33H, 51	MSR_MEMORY_CONTROL	
MSR_MEMORY_CONTROL (R/W) Disables split locks, which are locked instructions that split a cache line.		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor allows one in-progress, post-retirement UC stores at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a fault. If clear to 0, UC load locks proceed normally.	
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will trigger an #AC fault. If clear to 0, split locks proceed normally	
63:30	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
SMI Counter (R/W)		Thread
31:0	SMI_COUNT Running count of SMI events since the last reset.	
63:32	Reserved.	
Register Address: 39H, 57	MSR_SOCKET_ID	
Socket ID (R/W) Reassigns the package-specific portions of the APIC ID. This MSR is used on scalable DP and high-end MP platforms to resolve legacy-mode APIC ID conflicts.		Package
10:0	PACKAGE_ID: Holds package ID. This reflects the upper bits of the APIC ID.	
63:11	Reserved.	
Register Address: 7AH, 122	IA32_FEATURE_ACTIVATION	
IA32_FEATURE_ACTIVATION (R/W) Implements Feature Activation command. WRMSR to this address activates all 'activatable' features on this thread. See Table 2-2.		Thread
Register Address: 7BH, 123	IA32_MCU_ENUMERATION	
IA32_MCU_ENUMERATION (R/O) Enumeration of architectural features. See Table 2-2.		Package
Register Address: 7CH, 124	IA32_MCU_STATUS	
IA32_MCU_STATUS (R/O) Communicates results from the previous patch loads. See Table 2-2.		Package
Register Address: 82H, 130	IA32_FZM_RANGE_INDEX	
IA32_FZM_RANGE_INDEX (R/W) Index and Domain handle for a valid FZM region. Programmed by SW and used by other FRM MSRs FZM Range Index register to R/W Domain Index. See Table 2-2.		Thread
Register Address: 83H, 131	IA32_FZM_DOMAIN_CONFIG	
IA32_FZM_DOMAIN_CONFIG (R/O) Bit mask of valid regions within the domain identified by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 84H, 132	IA32_FZM_RANGE_STARTADDR	
IA32_FZM_RANGE_STARTADDR (R/O) Start address of the FZM range pointed to by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 85H, 133	IA32_FZM_RANGE_ENDADDR	
IA32_FZM_RANGE_ENDADDR (R/O) End address of the specified domain in FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 86H, 134	IA32_FZM_RANGE_WRITESTATUS	
IA32_FZM_RANGE_WRITESTATUS (R/O) Write status of the FZM range pointed to by FZM_RANGE_INDEX. See Table 2-2.		Thread
Register Address: 87H, 135	IA32_MKTME_KEYID_PARTITIONING	
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX. See Table 2-2.		Package

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 90H, 144	IA32_SGXLEPUBKEYHASH4	
IA32_SGXLEPUBKEYHASH4 (R/W) See Table 2-2.		Thread
Register Address: 91H, 145	IA32_SGXLEPUBKEYHASH5	
IA32_SGXLEPUBKEYHASH5 (R/W) See Table 2-2.		Thread
Register Address: 98H, 152	MSR_SEAM_WBINVDP	
SEAM WBINVDP (R/W) Allows software to WBINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 99H, 153	MSR_SEAM_WBNOINVDP	
SEAM WBNOINVDP (R/W) Allows software to WBNOINVDP sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 9AH, 154	MSR_SEAM_INTR_PENDING	
SEAM Interrupt Pending (R/O) Report out some event pending bits.		Thread
0	INTR Interrupt is pending.	
1	NMI NMI is pending.	
2	SMI SMI is pending.	
4:3	OTHER_EVENTS Other events pending.	
63:5	Reserved.	
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Control (R/W) The SMM Monitor Configuration involves SMM code specifying the MSEG location and enabling dual-monitor treatment by writing to the corresponding MSR. See Table 2-2.		Thread
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	STLB_QOS When set to 1, processor supports STLB QoS.	
1	Reserved.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
2	INTEGRITY_SUPPORTED When set to 1, processor supports Functional Safety. Specific FUSA capabilities are enumerated in MSR_FUSA_CAPABILITIES.	
3	RSM_IN_CPLO_ONLY Intel System Resources Defense: When set to 1, RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.	
4	UC_LOCK_DISABLE When set to 1, processor supports UC load lock disable.	
5	SPLIT_LOCK_DISABLE When set to 1, processor supports #AC on split locks.	
6	SNP_FILTER_QOS When set to 1, processor supports Snoop Filter Quality of Service MSRs.	
7	UC_STORE_THROTTLING When set to 1, processor supports UC store throttling through MSR_MEMORY_CTRL[UC_STORE_THROTTLE].	
63:8	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 105H, 261	MSR_ARRAY_BIST	
MSR_ARRAY_BIST (R/W) Triggered by writing and reading an MSR that can be written by Ring 0 software.		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	ARRAY_LIST: Bit map which indicates which arrays to run MarchC- BIST ▪ Bit[0] MLC Data ▪ Bit[1] MLC Tag ▪ Bit[2] C6SRAM Data (NOP for WRMSR – used for reporting error only) ▪ Bit[3] PMA BIST (NOP for WRMSR – used for reporting error only) ▪ Bit[4] STLB Data ▪ Bit[5] IFU Data ▪ Bit[6] STLB Tag ▪ Bit[7] DCU Data ▪ Bit[8] DSB Data ▪ Bit[9] TMUL Data ▪ Bit[10] UROM pointer0 ▪ Bit[11] UROM pointer1-3 ▪ Bit[12] UROM pointer4-7 ▪ Bit[13] UROM unique0 ▪ Bit[14] UROM unique1/2 The WRMSR will run PBIST on all the arrays indicated in the bitmap, starting from the LSB. NOTE2: C6SRAM[Bit 2] and PMA[Bit 3] are only for reporting and do not execute BIST (done by EDX[15:0]uCode during Fusa-Reset).	
46:32	Reserved.	
62:47	Reserved.	
63	SIGNAL_MCE: Signal MCERR upon BIST failure.	
Register Address: 105H, 261	MSR_ARRAY_BIST_STATUS	
MSR_ARRAY_BIST_STATUS (R/O)		Core
31:0	ARRAY_COMPLETION_MASK Bitmap indicating which arrays from the ARRAY_BIST.ARRAY_LIST was not processed. 1 means not tested and 0 means tested.	
62:32	Reserved. Returns all 0s.	
63	PASS_FAIL: 0 means Pass on all arrays in the WRMSR(ARRAY_BIST.ARRAY_LIST) 1 means Fail on the LSB array in the RDMSR(ARRAY_BIST_STATUS.ARRAY_COMPLETION_MASK).	
Register Address: 122H, 290	IA32_TSX_CTRL	
IA32_TSX_CTRL (R/W) See Table 2-2.		Thread
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Miscellaneous enables for thread-specific features. (R/W)		Thread

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
63:1	Reserved.	
Register Address: 1E0H, 480	IA32_LER_INFO	
IA32_LER_INFO (R/W) Last Event Record Destination IP Register. See Table 2-2.		Thread
Register Address: 1F9H, 505	IA32_CPU_DCA_CAP	
IA32_CPU_DCA_CAP (R/O) See Table 2-2.		Thread
Register Address: 2A1H, 673	MSR_PRMRR_BASE_1	
MSR_PRMRR_BASE_1 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register.		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
19:4	Reserved.	
51:20	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A2H, 674	MSR_PRMRR_BASE_2	
MSR_PRMRR_BASE_2 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A3H, 675	MSR_PRMRR_BASE_3	
MSR_PRMRR_BASE_3 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A4H, 676	MSR_PRMRR_BASE_4	
MSR_PRMRR_BASE_4 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A5H, 677	MSR_PRMRR_BASE_5	
MSR_PRMRR_BASE_5 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A6H, 678	MSR_PRMRR_BASE_6	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_PRMRR_BASE_6 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2A7H, 679	MSR_PRMRR_BASE_7	
MSR_PRMRR_BASE_7 (R/W) Processor Reserved Memory Range Register - Physical Base Control Register. See MSR_PRMRR_BASE_1 (2A1H) for reference; similar format.		Core
Register Address: 2B8H, 696	MSR_COPY_SBFT_HASHES	
MSR_COPY_SBFT_HASHES (W/O)		Module
63:0	SBFT_PROGRAM_SOURCE_ADDR EDX:EAX contains the linear address base of the SBFT Binary loaded into memory.	
Register Address: 2B9H, 697	MSR_SBFT_HASHES_STATUS	
MSR_COPY_SBFT_HASHES (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE - EDX[7:0] The error code refers to the LP that runs WRMSR(2B8H). 0x0: Reserved. 0x1: Attempt to copy SBFT-hashtes when copy already in progress. 0x2: Secure Memory not set up correctly. 0x3: Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4: Reserved. 0x5: Integrity check failed. 0x6: WRMSR(0x2B8) (ACTIVATE_SBAF) Reinstall of SBFT test image attempted when current SBFT test image is in use by other LPs. 0x7: Aborted due to #PF (Page Fault). 0x8: Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, stride value must be >= 1.	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	MAX_CORE_LIMIT EDX[30:19] - Maximum Number of Cores that can run SBFTAFSBAF simultaneously -1. 0 means 1 core at a time.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	Valid. EDX[31] - Valid bit is set when COPY_SBFT_HASHES completed successfully.	
Register Address: 2BAH, 698	MSR_AUTHENTICATE_AND_COPY_SBFT_CHUNK	
MSR_AUTHENTICATE_AND_COPY_SBFT_CHUNK (w/o)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2BBH, 699	MSR_SBFT_CHUNKS_AUTHENTICATION_STATUS	
MSR_SBFT_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	NUM_VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No error reported. 0x1 - Attempt to authenticate a CHUNK already marked as authentic or is currently being installed by another core. 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. 0x3 - Aborted due to #PF. 0x4 - Chunk Outside the current Stride. 0x5 - Interrupted.	
47:40	Reserved. EDX[15:8] - Set to all zeros.	
63:48	CURRENT_MAX_BUNDLE_INDX EDX[31:16] - Maximum Bundle Index in current stride.	
Register Address: 2BCH, 700	MSR_ACTIVATE_SBFT	
MSR_ACTIVATE_SBFT (w/o)		Core
13:0	SBFT_BUNDLE_INDEX EAX[13:0] - Indicates SBFT Bundle Index to start from.	
15:14	SBFT_PRGM_INDEX EAX[15:14] - Indicates what SBFT Program index to run.	
31:16	Reserved. Set to all zeros.	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC-based delay to allow threads to rendezvous.	
63	Reserved. EDX[31] - Must be set to 0. #GP fault otherwise.	
Register Address: 2BDH, 701	MSR_SBFT_STATUS	
MSR_SBFT_STATUS (R/O)		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
13:0	SBFT_BUNDLE_INDEX EAX[13:0] - SBFT Bundle that was executed.	
15:14	SBFT_PGM_INDEX EAX[15:14] - Indicates what SBFT Program index that was last ran. Maps to same field in WRMSR(ACTIVATE_SBFT). On a test pass this field will be 2'b00.	
31:16	Reserved. EAX[31:16] - Return all zeros.	
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SBFT operation did not start. Other thread could not join. 0x2 - SBFT operation did not start. Interrupt occurred prior to SBFT coordination. 0x3 - Reserved. 0x4 - SBFT operation did not start. Non-valid SBFT BUNDLES in the SBFT_BUNDLE_INDEX. 0x5 - SBFT operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SBFT operation did not start. Core is not capable of performing SBFT currently. 0x7 - Reserved. 0x8 - SBFT operation did not start. Exceeded number of Logical Processors (LP) allowed to run SBFT-At-Field concurrently. 0x9 - SBFT operation did not start. Interrupt occurred or timer about to expire. 0xA - SBFT operation did not start. SBFT_PGM_INDEX is not valid. 0xB - SBFT operation aborted due to corrupted chunk. 0xC - SBFT operation did not start. TAP Data error. 0xD - SBFT operation did not start. SBFT program is not valid. All other error codes are reserved.	
60:40	Reserved. EDX[28:8] - Return all zeros.	
61	TEST_FAIL EDX[29:29] - Architectural Signature failed. Last thread executed HLT and completed SBFT and EBX != 0xACED.	
63:62	SBFT_STATUS EDX[31:30] - SBFT status (result of running SBAF). 00 - PASS. 10 - INTERRUPTED. 01 - FAILED SIGNATURE CHECK. 11 - FAILED.	
Register Address: 2BEH, 702	MSR_SBFT_MODULE_ID	
MSR_SBFT_MODULE_ID (R/O)		Module
31:0	SBFT-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2BFH, 703	MSR_SBFTAF_LAST_WP	
MSR_SBFTAF_LAST_WP (R/O)		Module
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SBFT) was executed. Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved.	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	
MSR_COPY_SCAN_HASHES (W/O)		Module
63:0	SCAN_HASH-ADDR EDX:EAX contains the linear address of the SCAN Test HASH Binary loaded into memory	
Register Address: 2C3H, 707	MSR_SCAN_HASHES_STATUS	
MSR_SCAN_HASHES_STATUS (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE EDX[7:0] - The error code refers to the LP that runs WRMSR(2C2H). ▪ 0x0 - Reserved. ▪ 0x1 - Attempt to copy scan-hashes when copy already in progress. ▪ 0x2 - Secure Memory not set up correctly. ▪ 0x3 - Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. ▪ 0x4 - Reserved. ▪ 0x5 - Integrity check failed. ▪ 0x6 - WRMSR(0x2C6) Re-install of scan test image attempted when current scan test image is in use by other LPs. ▪ 0x7 - Aborted due to #PF (Page Fault). ▪ 0x8 - Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, the stride value must be >= 1.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	NAME EDX[30:19] - Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	
63	VALID EDX[31] - Valid bit is set when COPY_SCAN_HASHES completed.	
Register Address: 2C4H, 708	MSR_AUTHENTICATE_AND_COPY_CHUNK	
MSR_AUTHENTICATE_AND_COPY_CHUNK (R/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2C5H, 709	MSR_CHUNKS_AUTHENTICATION_STATUS	
MSR_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] ▪ 0x0 - No-error reported. ▪ 0x1 - Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. ▪ 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. ▪ 0x3 - Aborted due to #PF (Page Fault). ▪ 0x4 - Chunk Outside the current Stride.	
63:40	Reserved. EDX[31:8] - Set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
MSR_ACTIVATE_SCAN (W/O)		Core
15:0	CHUNK_START_INDEX EAX[15:0] - Indicates Chunk Index from which to start.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive).	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC based delay to allow threads to rendezvous.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63	SIGNAL_MCE EDX[31] - If 1: On scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. - If 0: Don't no-logging/no-signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
MSR_SCAN_STATUS (R/O)		Core
15:0	CHUNK_NUM EAX[15:0] - SCAN Chunk that was reached.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive). - Maps to same field in WRMSR(ACTIVATE_SCAN).	
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SCAN operation did not start. Other thread could not join. 0x2 - SCAN operation did not start. Interrupt occurred prior to SCAN coordination. 0x3 - SCAN operation did not start. Power Management conditions are inadequate to run SAF. 0x4 - SCAN operation did not start. Non valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5 - SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SCAN operation did not start. Core not capable of performing SCAN currently. 0x7 - Debug Mode. Scan-At-Field results not to be trusted. 0x8 - SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Scan-At-Field concurrently. MAX_CORE_LIMIT exceeded. 0x9 - Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed. 0xB - Scan operation aborted due to corrupted chunk. 0xC - Scan operation did not start. All other error codes are reserved.	
61:40	Reserved. EDX[29:8] - Return all zeros.	
62	SCAN_CONTROL_ERROR EDX[30] - SCAN error in the Scan-At-Field controller. - Non ECC error.	
63	SCAN_SIGNATURE_ERROR EDX[31] - SCAN SIGNATURE error in the SCAN pattern fetched from main memory. - Non ECC error.	
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_SCAN_MODULE_ID (R/O)		Module
31:0	SCAN-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2C9H, 713		
MSR_LAST_SAF_WP		
MSR_LAST_SAF_WP (R/O)		Module
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. - Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved. EDX[7:0]	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2D9H, 729		
MSR_INTEGRITY_CAPABILITIES		
MSR_INTEGRITY_CAPABILITIES (R/O) Enumerates features supported in Functional Safety.		Thread
0	STARTUP_SCAN_BIST When set to 1, processor supports Startup SCAN BIST.	
1	STARTUP_MEM_BIST When set to 1, processor supports Startup MEM BIST.	
2	PERIODIC_MEM_BIST When set to 1, processor supports Periodic MEM BIST.	
3	LOCKSTEP When set to 1, processor supports Lock Step Mode.	
4	PERIODIC_SCAN_BIST When set to 1, processor supports Periodic SCAN BIST.	
5	PLL_LOSS_DETECT When set to 1, processor supports PLL LOSS detection.	
6	PWR_LOSS_DETECT When set to 1, processor supports Power Loss detection.	
7	PERRINJ When set to 1, processor supports FUSA PERRINJ.	
8	SBFT_AT_FIELD When set to 1, processor supports SBFT-At-Field.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
10:9	SAF_GEN_REV 00 = REV1; 01 = REV2; 10 = REV3; 11 = REV4.	
14:11	Reserved.	
15	PRESERVE_MEMORY_NEEDED When set to 1, processor supports FUSARR_BASE/MASK MSRs.	
20:16	TID_BIT_SHIFT Number of bits to shift right on x2APICID to get a unique topology ID of all logical processors that share a scan test engine.	
21	ALL_LP_JOIN_NEEDED All logical processors that share scan test engine need to be tested together and must join using MSR_ACTIVATE_SCAN.	
23:22	Reserved.	
31:24	PATTERN_ID Processor scan pattern ID. ID of the startup and periodic scan programs supported for this part.	
63:32	Reserved.	
Register Address: 30CH, 780	IA32_FIXED_CTR3	
Fixed-Function Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 4D0H, 1232	IA32_MCG_EXT_CTL	
IA32_MCG_EXT_CTL (R/W) See Table 2-2.		Thread
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
MSR_SAF_CTRL (W/O)		Core
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride. - Clears only the VALID_CHUNKS field on a RDMSR(CHUNKS_AUTHENTICATION_STATUS).	
63:1	Reserved.	
Register Address: 4F8H, 1272	MSR_SBFT_CTRL	
MSR_SBFT_CTRL (W/O)		Module
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride.	
63:1	Reserved. EDX[31:0], EAX[31:1]	
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Microarchitectural Control (R/W)		Thread

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 541H, 1345	MSR_CORE_UARCH_CTL	
Core Microarchitecture Control MSR (R/W)		Core
0	SCRUB_DIS L1 scrubbing disable.	
63:1	Reserved.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY	
MSR_MC6_RESIDENCY (R/O) Time spent in the Module C6-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).		Module
63:0	RESIDENCY Time that this module is in module-specific C6 states since last reset.	
Register Address: 6E1H, 1761	IA32_PKRS	
IA32_PKRS (R/W) Specifies the PK permissions associated with each protection domain for supervisor pages. See Table 2-2.		Thread
Register Address: 7A3H, 1955	IA32_MCU_EXT_SERVICE	
MCU Extended Service MSR (R/O) If IA32_ARCH_CAPABILITIES[22] = 1. See Table 2-2.		Module
Register Address: 7A4H, 1956	IA32_MCU_ROLLBACK_MIN_ID	
Minimal MCU Revision ID for Rollback (R/O) See Table 2-2.		Module
Register Address: 7BOH, 1968	IA32_ROLLBACK_SIGN_ID_0	
Rollback ID 0 (R/O) See Table 2-2.		Module
Register Address: 7B1H, 1969	IA32_ROLLBACK_SIGN_ID_1	
Rollback ID 1 (R/O) See Table 2-2.		Module
Register Address: 7B2H, 1970	IA32_ROLLBACK_SIGN_ID_2	
Rollback ID 2 (R/O) See Table 2-2.		Module
Register Address: 7B3H, 1971	IA32_ROLLBACK_SIGN_ID_3	
Rollback ID 3 (R/O) See Table 2-2.		Module
Register Address: 7B4H, 1972	IA32_ROLLBACK_SIGN_ID_4	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Rollback ID 4 (R/O) See Table 2-2.		Module
Register Address: 7B5H, 1973	IA32_ROLLBACK_SIGN_ID_5	
Rollback ID 5 (R/O) See Table 2-2.		Module
Register Address: 7B6H, 1974	IA32_ROLLBACK_SIGN_ID_6	
Rollback ID 6 (R/O) See Table 2-2.		Module
Register Address: 7B7H, 1975	IA32_ROLLBACK_SIGN_ID_7	
Rollback ID 7 (R/O) See Table 2-2.		Module
Register Address: 7B8H, 1976	IA32_ROLLBACK_SIGN_ID_8	
Rollback ID 8 (R/O) See Table 2-2.		Module
Register Address: 7B9H, 1977	IA32_ROLLBACK_SIGN_ID_9	
Rollback ID 9 (R/O) See Table 2-2.		Module
Register Address: 7BAH, 1978	IA32_ROLLBACK_SIGN_ID_10	
Rollback ID 10 (R/O) See Table 2-2.		Module
Register Address: 7BBH, 1979	IA32_ROLLBACK_SIGN_ID_11	
Rollback ID 11 (R/O) See Table 2-2.		Module
Register Address: 7BCH, 1980	IA32_ROLLBACK_SIGN_ID_12	
Rollback ID 12 (R/O) See Table 2-2.		Module
Register Address: 7BDH, 1981	IA32_ROLLBACK_SIGN_ID_13	
Rollback ID 13 (R/O) See Table 2-2.		Module
Register Address: 7BEH, 1982	IA32_ROLLBACK_SIGN_ID_14	
Rollback ID 14 (R/O) See Table 2-2.		Module
Register Address: 7BFH, 1983	IA32_ROLLBACK_SIGN_ID_15	
Rollback ID 15 (R/O) See Table 2-2.		Module
Register Address: 981H, 2433	IA32_TME_CAPABILITY	
IA32_TME_CAPABILITY (R/O) See Table 2-2.		Package
Register Address: 982H, 2434	IA32_TME_ACTIVATE	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_TME_ACTIVATE (R/W) See Table 2-2.		Package
Register Address: 983H, 2435	IA32_TME_EXCLUDE_MASK	
Intel TME Exclude Mask (R/W) See Table 2-2.		Package
Register Address: 984H, 2436	IA32_TME_EXCLUDE_BASE	
Intel TME Exclude Base (R/W) See Table 2-2.		Package
Register Address: 985H, 2437	IA32_UINTR_RR	
User Interrupt Request Register (R/W) See Table 2-2.		Thread
Register Address: 986H, 2438	IA32_UINTR_HANDLER	
User Interrupt Handler Address (R/W) See Table 2-2.		Thread
Register Address: 987H, 2439	IA32_UINTR_STACKADJUST	
User Interrupt Stack Adjustment (R/W) See Table 2-2.		Thread
Register Address: 988H, 2440	IA32_UINTR_NV	
User-Interrupt Size and Notification Vector (R/W) See Table 2-2.		Thread
Register Address: 989H, 2441	IA32_UINTR_PD	
User Interrupt PID Address (R/W) See Table 2-2.		Thread
Register Address: 98AH, 2442	IA32_UINTR_TT	
User-Interrupt Target Table (R/W) See Table 2-2.		Thread
Register Address: 990H, 2448	IA32_COPY_STATUS	
IA32_COPY_STATUS (R/O) See Table 2-2.		Thread
Register Address: 991H, 2449	IA32_IWKEYBACKUP_STATUS	
IA32_IWKEYBACKUP_STATUS (R/O) See Table 2-2.		Package
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS] (R/O)		Core
31:0	Reserved.	

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	
63:40	Reserved.	
Register Address: C84H, 3204	MSR_MBA_CFG	
Memory Bandwidth Allocation (MBA) Configuration (R/W)		Package
1:0	Reserved.	
2	RAMBAE Resource Aware MBA Enable.	
63:3	Reserved.	
Register Address: CA0H, 3232	MSR_RMID_SNC_CONFIG	
RMID_SNC_CONFIG (R/W)		Package
0	RMID_LOCALIZED_DISTRIBUTION_MODE_ENABLE If set, Localized RMID distribution mode is enabled. If Clear, RMID Sharing mode is enabled.	
63:1	Reserved.	
Register Address: D50H, 3408	IA32_L2_QOS_EXT_BW_THRTL_0	
Memory Bandwidth Enforcement for COS0 (R/W) See Table 2-2.		Package
Register Address: D51H, 3409	IA32_L2_QOS_EXT_BW_THRTL_1	
Memory Bandwidth Enforcement for COS1 (R/W) See Table 2-2.		Package
Register Address: D52H, 3410	IA32_L2_QOS_EXT_BW_THRTL_2	
Memory Bandwidth Enforcement for COS2 (R/W) See Table 2-2.		Package
Register Address: D53H, 3411	IA32_L2_QOS_EXT_BW_THRTL_3	
Memory Bandwidth Enforcement for COS3 (R/W) See Table 2-2.		Package
Register Address: D54H, 3412	IA32_L2_QOS_EXT_BW_THRTL_4	
Memory Bandwidth Enforcement for COS4 (R/W) See Table 2-2.		Package
Register Address: D55H, 3413	IA32_L2_QOS_EXT_BW_THRTL_5	
Memory Bandwidth Enforcement for COS5 (R/W) See Table 2-2.		Package
Register Address: D56H, 3414	IA32_L2_QOS_EXT_BW_THRTL_6	
Memory Bandwidth Enforcement for COS6 (R/W) See Table 2-2.		Package

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: D57H, 3415	IA32_L2_QOS_EXT_BW_THRTL_7	
Memory Bandwidth Enforcement for COS7 (R/W) See Table 2-2.		Package
Register Address: D58H, 3416	IA32_L2_QOS_EXT_BW_THRTL_8	
Memory Bandwidth Enforcement for COS8 (R/W) See Table 2-2.		Package
Register Address: D59H, 3417	IA32_L2_QOS_EXT_BW_THRTL_9	
Memory Bandwidth Enforcement for COS9 (R/W) See Table 2-2.		Package
Register Address: D5AH, 3418	IA32_L2_QOS_EXT_BW_THRTL_10	
Memory Bandwidth Enforcement for COS10 (R/W) See Table 2-2.		Package
Register Address: D5BH, 3419	IA32_L2_QOS_EXT_BW_THRTL_11	
Memory Bandwidth Enforcement for COS11 (R/W) See Table 2-2.		Package
Register Address: D5CH, 3420	IA32_L2_QOS_EXT_BW_THRTL_12	
Memory Bandwidth Enforcement for COS12 (R/W) See Table 2-2.		Package
Register Address: D5DH, 3421	IA32_L2_QOS_EXT_BW_THRTL_13	
Memory Bandwidth Enforcement for COS13 (R/W) See Table 2-2.		Package
Register Address: D5EH, 3422	IA32_L2_QOS_EXT_BW_THRTL_14	
Memory Bandwidth Enforcement for COS14 (R/W) See Table 2-2.		Package
Register Address: D91H, 3473	IA32_COPY_LOCAL_TO_PLATFORM	
See Table 2-2.		Thread
Register Address: D92H, 3474	IA32_COPY_PLATFORM_TO_LOCAL	
See Table 2-2.		Thread
Register Address: D93H, 3475	IA32_PASID	
See Table 2-2.		Thread
Register Address: 1400H, 5120	IA32_SEAMRR_BASE	
SEAM Memory Range Register for TDx - Base Address (R/W) See Table 2-2.		Core
Register Address: 1401H, 5121	IA32_SEAMRR_MASK	
SEAM Memory Range Register for TDX (R/W) See Table 2-2.		Core
Register Address: 1A8FH, 6799	MSR_STLB_QOS_INFO	
STLB_QOS_INFO (R/O) STLB QoS MASK configuration.		Core

Table 2-56. Additional MSRs Supported by the Intel® Xeon® 6 P-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
5:0	NCLOS Number of CLOS supported for STLB resource using minus-1 notation.	
15:6	Reserved.	
19:16	4K_2M_CBM Length of capacity bitmask for 4K and 2M pages using minus-1 notation.	
28:20	Reserved.	
29	STLB_FILL_TRANSLATION_MSR_SUPPORTED MSR interface to fill STLB translations supported.	
30	4K_2M_ALIAS Indicates that 4K/2M pages alias into the same structure.	
63:31	Reserved.	
Register Address: 1B01H, 6913	IA32_UARCH_MISC_CTL	
IA32_UARCH_MISC_CTL (R/W) See Table 2-2.		Thread

2.17.11 MSRs Introduced in the Intel® Xeon® 6 E-Core Processors

Table 2-57 lists additional MSRs for the Intel Xeon 6 E-core processors. Intel Xeon 6 E-core processors have a CPUID Signature DisplayFamily_DisplayModel value of 06_AFH.

For an MSR listed in Table 2-57 that also appears in the model-specific tables of prior generations, Table 2-57 supersedes prior generation tables.

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2FH, 47	IA32_BARRIER	
BARRIER (R/O) The IA32_BARRIER MSR ensures ordered execution by acting like LFENCE, controlling the sequencing of subsequent MSR reads after prior MSR reads and instructions. See Table 2-2.		Core
Register Address: 33H, 51	MSR_MEMORY_CONTROL	
Memory Control (R/W) Disables split locks, which are locked instructions that split a cache line.		Core
26:0	Reserved.	
27	UC_STORE_THROTTLE If set to 1, when enabled, the processor allows one in-progress, post-retirement UC stores at a time.	
28	UC_LOCK_DISABLE If set to 1, a UC load lock will trigger a fault. If clear to 0, UC load locks proceed normally.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
29	SPLIT_LOCK_DISABLE If set to 1, a split lock will trigger an #AC fault. If clear to 0, split locks proceed normally.	
63:30	Reserved.	
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/W)		Thread
31:0	SMI_COUNT Running count of SMI events since the last reset.	
63:32	Reserved.	
Register Address: 39H, 57	MSR_SOCKET_ID	
Socket ID (R/W) Reassigns the package-specific portions of the APIC ID. This MSR is used on scalable DP and high-end MP platforms to resolve legacy-mode APIC ID conflicts.		Package
10:0	PACKAGE_ID Holds package ID. This reflects the upper bits of the APIC ID.	
63:11	Reserved.	
Register Address: 7BH, 123	IA32_MCU_ENUMERATION	
Enumeration of Architectural Features (R/O) See Table 2-2.		Package
Register Address: 7CH, 124	IA32_MCU_STATUS	
MCU Status (R/O) Communicates results from the previous patch loads. See Table 2-2.		Package
Register Address: 87H, 135	IA32_MKTME_KEYID_PARTITIONING	
MKTME KEY ID Partitioning (R/O) Enumerates the number of activated KeyIDs for Intel TME-MK and Intel TDX. See Table 2-2.		Package
Register Address: 98H, 152	MSR_SEAM_WBINVDP	
SEAM WBINVDP (R/W) Allows software to WBINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 99H, 153	MSR_SEAM_WBNINVD	
SEAM WBNINVD (R/W) Allows software to WBNINVD sections of the LLC.		Thread
63:0	HANDLE Caches sub-block to invalidate.	
Register Address: 9AH, 154	MSR_SEAM_INTR_PENDING	
SEAM Interrupt Pending (R/O) Report out some event pending bits.		Thread
0	INTR Interrupt is pending.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	NMI NMI is pending.	
2	SMI SMI is pending.	
4:3	OTHER_EVENTS Other events pending.	
63:5	Reserved.	
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Control (R/W) The SMM Monitor Configuration involves SMM code specifying the MSEG location and enabling dual-monitor treatment by writing to the corresponding MSR. See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information (R/O) Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
15:8	MAX_NON_TURBO_LIM_RATIO This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	
25:16	Reserved.	
26	DCU_16K_MODE_AVAIL 0b: Indicates that the part does not support the 16K DCU mode. 1b: Indicates that the part supports 16K DCU mode.	
27	Reserved.	
28	PRG_TURBO_RATIO_EN Programmable Turbo Ratios per number of Active Cores. 0 = Programming Not Allowed. 1 = Programming Allowed.	
34:29	Reserved.	
35	BIOS_GUARD_ENABLE Indicates whether the BIOS Guard feature is enabled in the CPU.	
36	PEG2DMIDIS_EN 0 = PEG2DMIDIS is disabled. 1 = PEG2DMIDIS is enabled.	
39:37	Reserved.	
47:40	MAX_EFFICIENCY_RATIO Maximum Efficiency Ratio. This is given in units of 100 MHz.	
58:48	Reserved.	
59	SMM_SUPOVR_STATE_LOCK_ENABLE When set, indicates that the CPU supports MSR SMM_SUPOVR_STATE_LOCK and the Hardware Shield feature.	
63:60	Reserved.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: CFH, 207	IA32_CORE_CAPABILITIES	
IA32 Core Capabilities Register (R/W) If CPUID.07H.00H:EDX[30] = 1. This MSR provides an architectural enumeration function for model-specific behavior.		Core
0	STLB_QOS When set to 1, processor supports STLB QoS.	
1	Reserved.	
2	INTEGRITY_SUPPORTED When set to 1, processor supports Functional Safety. Specific FUSA capabilities are enumerated in MSR_FUSA_CAPABILITIES.	
3	RSM_IN_CPLO_ONLY Intel System Resources Defense: When set to 1, RSM will only be allowed in CPLO and will #GP for all non-CPLO privilege levels.	
4	UC_LOCK_DISABLE When set to 1, processor supports UC load lock disable.	
5	SPLIT_LOCK_DISABLE When set to 1, processor supports #AC on split locks.	
6	SNP_FILTER_QOS When set to 1, processor supports Snoop Filter Quality of Service MSRs.	
7	UC_STORE_THROTTLING When set to 1, processor supports UC store throttling through MSR_MEMORY_CTRL[UC_STORE_THROTTLE].	
63:8	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 140H, 320	MSR_FEATURE_ENABLES	
Miscellaneous Enables for Thread-Specific Features (R/W)		Thread
0	AESNI_LOCK Once this bit is set, writes to this register will not be allowed.	
63:1	Reserved.	
Register Address: 1BOH, 432	IA32_ENERGY_PERF_BIAS	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_ENERGY_PERF_BIAS (R/W) See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
IA32_PACKAGE_THERM_STATUS See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
IA32_PACKAGE_THERM_INTERRUPT (R/W) See Table 2-2.		Package
Register Address: 2A1H, 673	MSR_PRMRR_BASE_1	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A2H, 674	MSR_PRMRR_BASE_2	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2A3H, 675	MSR_PRMRR_BASE_3	
Processor Reserved Memory Range Register - Physical Base Control Register (R/W)		Core
2:0	MEMTYPE Memory Type for PRMRR accesses.	
3	CONFIGURED PRMRR base configured.	
11:4	Reserved.	
51:12	BASE PRMRR Base address.	
63:52	Reserved.	
Register Address: 2C2H, 706	MSR_COPY_SCAN_HASHES	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_COPY_SCAN_HASHES (W/O)		Module
63:0	SCAN_HASH-ADDR EDX:EAX contains the linear address of the SCAN Test HASH Binary loaded into memory	
Register Address: 2C3H, 707		MSR_SCAN_HASHES_STATUS
MSR_SCAN_HASHES_STATUS (R/O)		Core
15:0	CHUNK_SIZE EAX[15:0] - Chunk size of the test in KB.	
31:16	TOTAL_NUM_CHUNKS EAX[31:16] - Total number of chunks.	
39:32	ERROR_CODE EDX[7:0] - The error code refers to the LP that runs WRMSR(2C2H). 0x0 - Reserved. 0x1 - Attempt to copy scan-hashees when copy already in progress. 0x2 - Secure Memory not set up correctly. 0x3 - Scan-Image Header Image_info.ProgramID does not match MSR_INTEGRITY_CAPABILITIES[31:24], or scan-image header Processor-Signature doesn't match F/M/S, or scan-image header Processor-Flags doesn't match PlatformID. 0x4 - Reserved. 0x5 - Integrity check failed. 0x6 - WRMSR(0x2C6) Re-install of scan test image attempted when current scan test image is in use by other LPs. 0x7 - Aborted due to #PF (Page Fault). 0x8 - Unable to generate a Random Value.	
48:40	NUM_CHUNKS_IN_STRIDE EDX[16:8] - Number of Chunks in stride. This is the number of chunks that are installed. 0 in this field means that the CPU does not support strides, otherwise, the stride value must be >= 1	
50:49	Reserved. EDX[18:17] - Set to all zeros.	
62:51	NAME EDX[30:19] - Maximum Number of cores that can run Intel® In-field Scan simultaneously minus 1. 0 means 1 core at a time.	
63	VALID EDX[31] - Valid bit is set when COPY_SCAN_HASHES completed.	
Register Address: 2C4H, 708		MSR_AUTHENTICATE_AND_COPY_CHUNK
MSR_AUTHENTICATE_AND_COPY_CHUNK(R/O)		Core
63:0	BASE_CHUNK_TABLE_ADDR EDX:EAX[63:0] - Linear Address pointing to the CHUNK TABLE (TABLE_BASE).	
Register Address: 2C5H, 709		MSR_CHUNKS_AUTHENTICATION_STATUS

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_CHUNKS_AUTHENTICATION_STATUS (R/O)		Core
15:0	VALID_CHUNKS EAX[15:0] - Total number of Valid (authenticated) chunks.	
31:16	NUM_CHUNKS_IN_STRIDE EAX[31:16] - Number of Chunks in Stride.	
39:32	ERROR_CODE EDX[7:0] ▪ 0x0 - No-error reported. ▪ 0x1 - Attempt to authenticate a CHUNK which is already marked as authentic or is currently being installed by another core. ▪ 0x2 - CHUNK authentication error. HASH of chunk did not match expected value. ▪ 0x3 - Aborted due to #PF (Page Fault). ▪ 0x4 - Chunk Outside the current Stride.	
63:40	Reserved. EDX[31:8] - Set to all zeros.	
Register Address: 2C6H, 710	MSR_ACTIVATE_SCAN	
MSR_ACTIVATE_SCAN (W/O)		Core
15:0	CHUNK_START_INDEX EAX[15:0] - Indicates Chunk Index from which to start.	
31:16	CHUNK_STOP_INDEX EAX[31:16] - Indicates what chunk index to stop at (inclusive).	
62:32	THREAD_WAIT_DELAY EDX[30:0] - TSC based delay to allow threads to rendezvous.	
63	SIGNAL_MCE EDX[31] ▪ If 1: On scan-error log MC in MC4_STATUS and signal MCE if machine check signaling enabled in MC4_CTL[0]. ▪ If 0: Don't no-logging/no-signaling.	
Register Address: 2C7H, 711	MSR_SCAN_STATUS	
MSR_SCAN_STATUS (R/O)		Core
15:0	CHUNK_NUM EAX[15:0] - SCAN Chunk that was reached.	
31:16	CHUNK_STOP_INDEX EAX[31:16] ▪ Indicates what chunk index to stop at (inclusive). ▪ Maps to same field in WRMSR(ACTIVATE_SCAN).	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
39:32	ERROR_CODE EDX[7:0] 0x0 - No Error. 0x1 - SCAN operation did not start. Other thread could not join. 0x2 - SCAN operation did not start. Interrupt occurred prior to SCAN coordination. 0x3 - SCAN operation did not start. Power Management conditions are inadequate to run SAF. 0x4 - SCAN operation did not start. Non valid chunks in the range CHUNK_STOP_INDEX : CHUNK_START_INDEX. 0x5 - SCAN operation did not start. Mismatch in arguments between threads T0/T1. 0x6 - SCAN operation did not start. Core not capable of performing SCAN currently. 0x7 - Debug Mode. Scan-At-Field results not to be trusted. 0x8 - SCAN operation did not start. Exceeded number of Logical Processors (LP) allowed to run Scan-At-Field concurrently. MAX_CORE_LIMIT exceeded. 0x9 - Interrupt occurred. Scan operation aborted prematurely, not all chunks requested have been executed. 0xB - Scan operation aborted due to corrupted chunk. 0xC - Scan operation did not start. All other error codes are reserved.	
61:40	Reserved. EDX[29:8] - Return all zeros.	
62	SCAN_CONTROL_ERROR EDX[30] - SCAN error in the Scan-At-Field controller. - Non ECC error.	
63	SCAN_SIGNATURE_ERROR EDX[31] - SCAN SIGNATURE error in the SCAN pattern fetched from main memory. - Non ECC error.	
Register Address: 2C8H, 712	MSR_SCAN_MODULE_ID	
MSR_SCAN_MODULE_ID (R/O)		Module
31:0	SCAN-AT-FIELD_REVID EAX[31:0] - Maps to Revision field in external header (offset 4).	
40:32	CURRENT_STRIDE_INDEX EDX[8:0] - Stride Index.	
63:41	Reserved. EDX[31:9] - Return all zeros.	
Register Address: 2C9H, 713	MSR_LAST_SAF_WP	
MSR_LAST_SAF_WP (R/O)		Module

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:0	LAST_WP EAX[31:0] - Provides information about the core when the last WRMSR(ACTIVATE_SCAN) was executed. - Available only if enumerated in INTEGRITY_CAPABILITIES[10:9].	
39:32	Reserved. EDX[7:0]	
63:40	Reserved. EDX[31:8] - Return all zeros.	
Register Address: 2D6H, 726	MSR_TRIGGER_PERIODIC_MEM_BIST	
MSR_TRIGGER_PERIODIC_MEM_BIST (W/O)		Core
0	SIGNAL_MCE EAX[0] - If 1, then signal MCE on fail if machine check signaling enabled in the corresponding MCI_CTL. If 0 then don't signal machine checks.	
7:1	ARRAY_BANK EAX[7:1] - Reserved.	
15:8	TST_STEP_PARAM EAX[15:8] 0: Test All Arrays, or Test Arrays in STEPs of NUM_STEPS.	
31:16	Reserved. EAX[31:16]	
63:32	Reserved. EAX[31:0]	
Register Address: 2D7H, 727	MSR_PERIODIC_MEM_BIST_STATUS	
MSR_PERIODIC_MEM_BIST_STATUS (R/O)		Core
0	MEM_BIST_STATUS 0: PASS. 1: FAIL.	
63:1	Reserved.	
Register Address: 2D9H, 729	MSR_INTEGRITY_CAPABILITIES	
MSR_INTEGRITY_CAPABILITIES (R/O) Enumerates features supported in Functional Safety.		Thread
0	STARTUP_SCAN_BIST When set to 1, processor supports Startup SCAN BIST.	
1	STARTUP_MEM_BIST When set to 1, processor supports Startup MEM BIST.	
2	PERIODIC_MEM_BIST When set to 1, processor supports Periodic MEM BIST.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
3	LOCKSTEP When set to 1, processor supports Lock Step Mode.	
4	PERIODIC_SCAN_BIST When set to 1, processor supports Periodic SCAN BIST.	
5	PLL_LOSS_DETECT When set to 1, processor supports PLL LOSS detection.	
6	PWR_LOSS_DETECT When set to 1, processor supports Power Loss detection.	
7	PERRINJ When set to 1, processor supports FUSA PERRINJ.	
8	SBFT_AT_FIELD When set to 1, processor supports SBFT-At-Field.	
10:9	SAF_GEN_REV 00 = REV1; 01 = REV2; 10 = REV3; 11 = REV4.	
14:11	Reserved.	
15	PRESERVE_MEMORY_NEEDED When set to 1, processor supports FUSARR_BASE/MASK MSRs.	
20:16	TID_BIT_SHIFT Number of bits to shift right on x2APICID to get a unique topology ID of all logical processors that share a scan test engine.	
21	ALL_LP_JOIN_NEEDED All logical processors that share scan test engine need to be tested together and must join using MSR_ACTIVATE_SCAN.	
23:22	Reserved.	
31:24	PATTERN_ID Processor scan pattern ID. ID of the startup and periodic scan programs supported for this part.	
63:32	Reserved.	
Register Address: 2DCH, 732	IA32_INTEGRITY_STATUS	
IA32_INTEGRITY_STATUS (R/O) Provides status information for integrity features. See Table 2-2.		Thread
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
MSR_PKG_C6_RESIDENCY (R/O)		Package
63:0	Package C6 Residency Counter	
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
MSR_PKG_C7_RESIDENCY (R/O)		Package
63:0	Package C7 Residency Counter	
Register Address: 3FCH, 1020	MSR_CORE_C3_RESIDENCY	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MSR_CORE_C3_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C3 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 3FDH, 1021	MSR_CORE_C6_RESIDENCY	
MSR_CORE_C6_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C6 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 3FEH, 1022	MSR_CORE_C7_RESIDENCY	
MSR_CORE_C7_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Core
63:0	CORE C7 Residency Counter Time spent in the Core C-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).	
Register Address: 4F0H, 1264	MSR_SAF_CTRL	
MSR_SAF_CTRL (W/O)		Core
0	INVALIDATE_CURRENT_STRIDE EAX[0] - Write of 1 invalidates the currently installed stride. - Clears only the VALID_CHUNKS field on a RDMSR(CHUNKS_AUTHENTICATION_STATUS).	
63:1	Reserved.	
Register Address: 664H, 1636	MSR_MC6_RESIDENCY	
MSR_MC6_RESIDENCY (R/O) Time spent in the Module C6-State. Provided in units compatible to P1 clock frequency (Guaranteed / Maximum Core Non-Turbo Frequency).		Module
63:0	RESIDENCY Time that this module is in module-specific C6 states since last reset.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Thread
Register Address: 7A3H, 1955	IA32_MCU_EXT_SERVICE	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
MCU Extended Service (R/W) See Table 2-2.		Module
Register Address: 7A4H, 1956	IA32_MCU_ROLLBACK_MIN_ID	
Minimal MCU Revision ID (R/O) See Table 2-2.		Module
Register Address: 7A5H, 1957	IA32_MCU_STAGING_MBOX_ADDR	
IA32_MCU_STAGING_MBOX_ADDR (R/O) See Table 2-2.		Package
Register Address: 7B0H, 1968	IA32_ROLLBACK_SIGN_ID_0	
Rollback ID 0 (R/O) See Table 2-2.		Module
Register Address: 7B1H, 1969	IA32_ROLLBACK_SIGN_ID_1	
Rollback ID 1 (R/O) See Table 2-2.		Module
Register Address: 7B2H, 1970	IA32_ROLLBACK_SIGN_ID_2	
Rollback ID 2 (R/O) See Table 2-2.		Module
Register Address: 7B3H, 1971	IA32_ROLLBACK_SIGN_ID_3	
Rollback ID 3 (R/O) See Table 2-2.		Module
Register Address: 7B4H, 1972	IA32_ROLLBACK_SIGN_ID_4	
Rollback ID 4 (R/O) See Table 2-2.		Module
Register Address: 7B5H, 1973	IA32_ROLLBACK_SIGN_ID_5	
Rollback ID 5 (R/O) See Table 2-2.		Module
Register Address: 7B6H, 1974	IA32_ROLLBACK_SIGN_ID_6	
Rollback ID 6 (R/O) See Table 2-2.		Module
Register Address: 7B7H, 1975	IA32_ROLLBACK_SIGN_ID_7	
Rollback ID 7 (R/O) See Table 2-2.		Module
Register Address: 7B8H, 1976	IA32_ROLLBACK_SIGN_ID_8	
Rollback ID 8 (R/O) See Table 2-2.		Module
Register Address: 7B9H, 1977	IA32_ROLLBACK_SIGN_ID_9	
Rollback ID 9 (R/O) See Table 2-2.		Module
Register Address: 7BAH, 1978	IA32_ROLLBACK_SIGN_ID_10	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Rollback ID 10 (R/O) See Table 2-2.		Module
Register Address: 7BBH, 1979	IA32_ROLLBACK_SIGN_ID_11	
Rollback ID 11 (R/O) See Table 2-2.		Module
Register Address: 7BCH, 1980	IA32_ROLLBACK_SIGN_ID_12	
Rollback ID 12 (R/O) See Table 2-2.		Module
Register Address: 7BDH, 1981	IA32_ROLLBACK_SIGN_ID_13	
Rollback ID 13 (R/O) See Table 2-2.		Module
Register Address: 7BEH, 1982	IA32_ROLLBACK_SIGN_ID_14	
Rollback ID 14 (R/O) See Table 2-2.		Module
Register Address: 7BFH, 1983	IA32_ROLLBACK_SIGN_ID_15	
Rollback ID 15 (R/O) See Table 2-2.		Module
Register Address: 988H, 2440	IA32_UINTR_NV	
User Interrupt Size and Notification Vector (R/W) See Table 2-2.		Thread
Register Address: 9FBH, 2555	IA32_TME_CLEAR_SAVED_KEY	
IA32_TME_CLEAR_SAVED_KEY (R/W) See Table 2-2.		Package
Register Address: 9FFH, 2559	MSR_CORE_MKTME_ACTIVATE	
MSR_CORE_MKTME_ACTIVATE (R/O) MSR to read TME_ACTIVATE[MK_TME_KEYID_BITS].		Core
31:0	Reserved.	
35:32	READ_MK_TME_KEYID_BITS This value will be returned on a RDMSR, but must be zero on a WRMSR.	
39:36	TDX_RESERVED_KEYID_BITS (read only) The number of key identifier bits allocated to TDX usage. This is a read-only field. #GP on a non-zero write.	
63:40	Reserved.	
Register Address: C84H, 3204	MSR_MBA_CFG	
Memory Bandwidth Allocation (MBA) Configuration (R/W)		Package
1:0	Reserved.	
2	RAMBAE Resource Aware MBA Enable.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:3	Reserved.	
Register Address: CA0H, 3232	MSR_RMID_SNC_CONFIG	
MSR_RMID_SNC_CONFIG (R/W)		Package
0	RMID_LOCALIZED_DISTRIBUTION_MODE_ENABLE If set, Localized RMID distribution mode is enabled. If Clear, RMID Sharing mode is enabled.	
63:1	Reserved.	
Register Address: D50H, 3408	IA32_L2_QOS_EXT_BW_THRTL_0	
Memory Bandwidth Enforcement for COS0 (R/W) See Table 2-2.		Package
Register Address: D51H, 3409	IA32_L2_QOS_EXT_BW_THRTL_1	
Memory Bandwidth Enforcement for COS1 (R/W) See Table 2-2.		Package
Register Address: D52H, 3410	IA32_L2_QOS_EXT_BW_THRTL_2	
Memory Bandwidth Enforcement for COS2 (R/W) See Table 2-2.		Package
Register Address: D53H, 3411	IA32_L2_QOS_EXT_BW_THRTL_3	
Memory Bandwidth Enforcement for COS3 (R/W) See Table 2-2.		Package
Register Address: D54H, 3412	IA32_L2_QOS_EXT_BW_THRTL_4	
Memory Bandwidth Enforcement for COS4 (R/W) See Table 2-2.		Package
Register Address: D55H, 3413	IA32_L2_QOS_EXT_BW_THRTL_5	
Memory Bandwidth Enforcement for COS5 (R/W) See Table 2-2.		Package
Register Address: D56H, 3414	IA32_L2_QOS_EXT_BW_THRTL_6	
Memory Bandwidth Enforcement for COS6 (R/W) See Table 2-2.		Package
Register Address: D57H, 3415	IA32_L2_QOS_EXT_BW_THRTL_7	
Memory Bandwidth Enforcement for COS7 (R/W) See Table 2-2.		Package
Register Address: D58H, 3416	IA32_L2_QOS_EXT_BW_THRTL_8	
Memory Bandwidth Enforcement for COS8 (R/W) See Table 2-2.		Package
Register Address: D59H, 3417	IA32_L2_QOS_EXT_BW_THRTL_9	
Memory Bandwidth Enforcement for COS9 (R/W) See Table 2-2.		Package
Register Address: D5AH, 3418	IA32_L2_QOS_EXT_BW_THRTL_10	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Memory Bandwidth Enforcement for COS10 (R/W) See Table 2-2.		Package
Register Address: D5BH, 3419	IA32_L2_QOS_EXT_BW_THRTL_11	
Memory Bandwidth Enforcement for COS11 (R/W) See Table 2-2.		Package
Register Address: D5CH, 3420	IA32_L2_QOS_EXT_BW_THRTL_12	
Memory Bandwidth Enforcement for COS12 (R/W) See Table 2-2.		Package
Register Address: D5DH, 3421	IA32_L2_QOS_EXT_BW_THRTL_13	
Memory Bandwidth Enforcement for COS13 (R/W) See Table 2-2.		Package
Register Address: D5EH, 3422	IA32_L2_QOS_EXT_BW_THRTL_14	
Memory Bandwidth Enforcement for COS14 (R/W) See Table 2-2.		Package
Register Address: E00H, 3584	IA32_QOS_CORE_BW_THRTL_0	
CBA Levels Based on COS for Bandwidth Throttling (R/W) See Table 2-2.		Thread
Register Address: E01H, 3585	IA32_QOS_CORE_BW_THRTL_1	
CBA Levels Based on COS for Bandwidth Throttling (R/W) See Table 2-2.		Thread
Register Address: 1400H, 5120	IA32_SEAMRR_BASE	
SEAM Memory Range Register for TDX - Base Address (R/W) See Table 2-2.		Core
Register Address: 1401H, 5121	IA32_SEAMRR_MASK	
SEAM Memory Range Register for TDX (R/W) See Table 2-2.		Core
Register Address: 1A8FH, 6799	MSR_STLB_QOS_INFO	
STLB_QOS_INFO (R/O) STLB QoS MASK configuration.		Core
5:0	NCLOS Number of CLOS supported for STLB resource using minus-1 notation.	
15:6	Reserved.	
19:16	4K_2M_CBM Length of capacity bitmask for 4K and 2M pages using minus-1 notation.	
28:20	Reserved.	
29	STLB_FILL_TRANSLATION_MSR_SUPPORTED MSR interface to fill STLB translations supported.	

Table 2-57. Additional MSRs Supported by the Intel® Xeon® 6 E-Core Processors (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
30	4K_2M_ALIAS Indicates that 4K/2M pages alias into the same structure.	
63:31	Reserved.	

2.17.12 MSRs Introduced in the Intel® Series 2 Core™ Ultra Processor Supporting Performance Hybrid Architecture

Table 2-58 lists additional MSRs for the Intel Series 2 Core Ultra processor with a CPUID Signature DisplayFamily_DisplayModel value of 06_BDH. Table 2-59 lists the MSRs unique to the processor P-core. Table 2-60 lists the MSRs unique to the processor E-core.

For an MSR listed in Table 2-58, Table 2-59, or Table 2-60 that also appears in the model-specific tables of prior generations, Table 2-58, Table 2-59, and Table 2-60 supersede prior generation tables.

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 601H, 1537	MSR_PKG_POWER_LIMIT_4	
Package Power Limit 4 (R/W) Package-level maximum power limit (in Watts).		Package
15:0	POWER_LIMIT_4 PL4 Value in 0.125 W increments. This field is locked by PKG_POWER_LIMIT_4.LOCK. When the LOCK bit is set to 1, this field becomes Read Only. If the value is 0, PL4 limit is disabled.	
30:16	Reserved.	
31	LOCK This bit will lock the POWER_LIMIT_4 settings in this register and will also lock this setting. This means that once set to 1, the POWER_LIMIT_4 setting and this bit become Read Only until the next Warm Reset.	
63:32	Reserved.	
Register Address: 630H, 1584	MSR_PKG_C8_RESIDENCY	
MSR_PKG_C8_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
59:0	Package C8 Residency Counter Value since last reset that this package is in processor-specific C8 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 631H, 1585	MSR_PKG_C9_RESIDENCY	
MSR_PKG_C9_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C9 Residency Counter Value since last reset that this package is in processor-specific C9 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 632H, 1586	MSR_PKG_C10_RESIDENCY	
MSR_PKG_C10_RESIDENCY (R/O) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.		Package
59:0	Package C10 Residency Counter Value since last reset that this package is in processor-specific C10 states. Count at the same frequency as the TSC.	
63:60	Reserved.	
Register Address: 651H, 1617	MSR_SECONDARY_TURBO_RATIO_LIMIT_CORES	
SECONDARY_TURBO_RATIO_LIMIT_CORES (R/W) This register defines the active core ranges for each frequency point. - NUMCORE[0:7] must be populated in ascending order. - NUMCORE[i+1] must be greater than NUMCORE[i]. - Entries with NUMCORE[i] == 0 will be ignored. - The last valid entry must have NUMCORE >= the number of cores in the SKU. If any of the rules above are broken, we will silently reject the configuration.		Package
7:0	CORE_COUNT_0 Defines the active core ranges for each frequency point.	
15:8	CORE_COUNT_1 Defines the active core ranges for each frequency point.	
23:16	CORE_COUNT_2 Defines the active core ranges for each frequency point.	
31:24	CORE_COUNT_3 Defines the active core ranges for each frequency point.	
39:32	CORE_COUNT_4 Defines the active core ranges for each frequency point.	
47:40	CORE_COUNT_5 Defines the active core ranges for each frequency point.	
55:48	CORE_COUNT_6 Defines the active core ranges for each frequency point.	

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:56	CORE_COUNT_7 Defines the active core ranges for each frequency point.	
Register Address: 658H, 1624	MSR_WEIGHTED_CORE_CO	
Core-Count Weighted C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 659H, 1625	MSR_ANY_CORE_CO	
Any Core C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is weighted by the number of processor cores in the package that reside in C0. If N cores are simultaneously in C0, then each cycle the counter increments by N.	
Register Address: 65AH, 1626	MSR_ANY_GFXE_CO	
Any Graphics Engine C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is one if any processor graphic device's compute engines are in C0.	
Register Address: 65BH, 1627	MSR_CORE_GFXE_OVERLAP_CO	
Core and Graphics Engine Overlapped C0 Residency (R/O)		Package
63:0	DATA Increment at the same rate as the TSC. The increment each cycle is one if at least one compute engine of the processor graphics is in C0 and at least one processor core in the package is also in C0.	
Register Address: C88H, 3208	IA32_RESOURCE_PRIORITY	
Thread scope Resource Priority Enable (R/W) See Table 2-2.		Thread
Register Address: C89H, 3209	IA32_RESOURCE_PRIORITY_PKG	
IA32_RESOURCE_PRIORITY_PKG (R/W) See Table 2-2.		Package
Register Address: 1900H, 6400	IA32_PMC_GPO_CTR	
Full Width Writable General Performance Counter 0 (R/W) See Table 2-2.		Thread
Register Address: 1901H, 6401	IA32_PMC_GPO_CFG_A	
IA32_PMC_GPO_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 0. See Table 2-2.		Thread

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1904H, 6404	IA32_PMC_GP1_CTR	
Full Width Writable General Performance Counter 1 (R/W) See Table 2-2.		Thread
Register Address: 1905H, 6405	IA32_PMC_GP1_CFG_A	
IA32_PMC_GP1_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 1. See Table 2-2.		Thread
Register Address: 1908H, 6408	IA32_PMC_GP2_CTR	
Full Width Writable General Performance Counter 2 (R/W) See Table 2-2.		Thread
Register Address: 1909H, 6409	IA32_PMC_GP2_CFG_A	
IA32_PMC_GP2_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 2. See Table 2-2.		Thread
Register Address: 190CH, 6412	IA32_PMC_GP3_CTR	
Full Width Writable General Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 190DH, 6413	IA32_PMC_GP3_CFG_A	
IA32_PMC_GP3_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 3. See Table 2-2.		Thread
Register Address: 1910H, 6416	IA32_PMC_GP4_CTR	
Full Width Writable General Performance Counter 4 (R/W) See Table 2-2.		Thread
Register Address: 1911H, 6417	IA32_PMC_GP4_CFG_A	
IA32_PMC_GP4_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 4. See Table 2-2.		Thread
Register Address: 1914H, 6420	IA32_PMC_GP5_CTR	
Full Width Writable General Performance Counter 5 (R/W) See Table 2-2.		Thread
Register Address: 1915H, 6421	IA32_PMC_GP5_CFG_A	
IA32_PMC_GP5_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 5. See Table 2-2.		Thread
Register Address: 1918H, 6424	IA32_PMC_GP6_CTR	
Full Width Writable General Performance Counter 6 (R/W) See Table 2-2.		Thread
Register Address: 1919H, 6425	IA32_PMC_GP6_CFG_A	

Table 2-58. Additional MSRs Supported by the Intel® Series 2 Core™ Ultra Processors Supporting Performance Hybrid Architecture (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
IA32_PMC_GP6_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 6. See Table 2-2.		Thread
Register Address: 191CH, 6428	IA32_PMC_GP7_CTR	
Full Width Writable General Performance Counter 7 (R/W) See Table 2-2.		Thread
Register Address: 191DH, 6429	IA32_PMC_GP7_CFG_A	
IA32_PMC_GP7_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 7. See Table 2-2.		Thread
Register Address: 1980H, 6528	IA32_PMC_FX0_CTR	
IA32_PMC_FX0_CTR (R/W) Fixed-Function Performance Counter 0 - Instructions Retired. See Table 2-2.		Thread
Register Address: 1984H, 6532	IA32_PMC_FX1_CTR	
IA32_PMC_FX1_CTR (R/W) Fixed-Function Performance Counter 1 - Unhalted core clock cycles. See Table 2-2.		Thread
Register Address: 1988H, 6536	IA32_PMC_FX2_CTR	
IA32_PMC_FX2_CTR (R/W) Fixed-Function Performance Counter 2 - Unhalted core reference cycles. See Table 2-2.		Thread

The MSRs listed in Table 2-59 are unique to the Intel Series 2 Core Ultra processor P-core. These MSRs are not supported on the processor E-core.

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: C9H, 201	IA32_PMC8	
General Performance Counter 8 (R/W) See Table 2-2.		Thread
Register Address: CAH, 202	IA32_PMC9	
General Performance Counter 9 (R/W) See Table 2-2.		Thread
Register Address: 18EH, 398	IA32_PERFEVTSEL8	
Performance Event Select Register 8 (R/W) See Table 2-2.		Thread
Register Address: 18FH, 399	IA32_PERFEVTSEL9	
Performance Event Select Register 9 (R/W) See Table 2-2.		Thread
Register Address: 30CH, 780	IA32_FIXED_CTR3	

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Fixed-Function Performance Counter 3 (R/W) See Table 2-2.		Thread
Register Address: 329H, 809	MSR_PERF_METRICS	
MSR_PERF_METRICS (R/W) This register provides built-in support for Top-down Micro-architecture Analysis (TMA) metrics. It exposes the four TMA Level 1 metrics where the lower 32 bits are divided into four 8 bit fields, each of which is an integer percentage of the total TOPDOWN.SLOTS (as reported by fixed-function counter 3).		Thread
7:0	RETIRING Percent of utilized by uops that eventually retire (commit).	
15:8	BAD_SPECULATION Percent of Wasted due to incorrect speculation, covering Utilized by uops that do not retire, or Recovery Bubbles (unutilized slots).	
23:16	FRONTEND_BOUND Percent of Unutilized slots where Front-end did not deliver a uop while Back-end is ready.	
31:24	BACKEND_BOUND Percent of Unutilized slots where a uop was not delivered to Back-end due to lack of Back-end resources.	
39:32	MULTI_UOPS Frontend bound.	
47:40	BRANCH_MISPREDICTS Frontend bound.	
55:48	FRONTEND_LATENCY Frontend bound.	
63:56	MEMORY_BOUND Frontend bound.	
Register Address: 4C9H, 1225	IA32_A_PMC8	
Full Width Writable IA32_PMC8 Alias (R/W) See Table 2-2.		Thread
Register Address: 4CAH, 1226	IA32_A_PMC9	
Full Width Writable IA32_PMC9 Alias (R/W) See Table 2-2.		Thread
Register Address: 540H, 1344	MSR_THREAD_UARCH_CTL	
Thread Uarch Control (R/W)		Thread
0	WB_MEM_STRM_LD_DISABLE Disable streaming behavior for MOVNTDQA loads to WB memory type. If set, these accesses will be treated like regular cacheable loads (Data will be cached).	
63:1	Reserved.	
Register Address: 540H, 1344	MSR_CORE_UARCH_CTL	
Core Uarch Control (R/W)		Core

Table 2-59. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor P-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
0	SCRUB_DIS L1 scrubbing disable.	
63:1	Reserved.	
Register Address: 1920H, 6432	IA32_PMC_GP8_CTR	
Full Width Writable General Performance Counter 8 (R/W) See Table 2-2.		Thread
Register Address: 1921H, 6433	IA32_PMC_GP8_CFG_A	
IA32_PMC_GP8_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 8. See Table 2-2.		Thread
Register Address: 1924H, 6436	IA32_PMC_GP9_CTR	
Full Width Writable General Performance Counter 9 (R/W) See Table 2-2.		Thread
Register Address: 1925H, 6437	IA32_PMC_GP9_CFG_A	
IA32_PMC_GP9_CFG_A (R/W) Performance Event Select Register used to control the operation of the General Performance Counter 9. See Table 2-2.		Thread
Register Address: 198CH, 6540	IA32_PMC_FX3_CTR	
IA32_PMC_FX3_CTR (R/W) See Table 2-2.		Thread

The MSRs listed in Table 2-60 are unique to the Intel Series 2 Core Ultra processor E-core. These MSRs are not supported on the processor P-core.

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 2DCH, 732	IA32_INTEGRITY_STATUS	
Status Information for Integrity Features (R/O) See Table 2-2.		Thread
Register Address: 30DH, 781	IA32_FIXED_CTR4	
Fixed-Function Performance Counter 4 - Top-down Bad Speculation (R/W) See Table 2-2.		Thread
Register Address: 30EH, 782	IA32_FIXED_CTR5	
Fixed-Function Performance Counter 5 - Top-down Frontend Bound (R/W) See Table 2-2.		Thread
Register Address: 30FH, 783	IA32_FIXED_CTR6	
Fixed-Function Performance Counter 6 - Top-down Retiring (R/W) See Table 2-2.		Thread
Register Address: D18H, 3352	IA32_L2_MASK_8	

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
L2 CAT Mask for COS8 (R/W) See Table 2-2.		Module
Register Address: D19H, 3353	IA32_L2_MASK_9	
L2 CAT Mask for COS9 (R/W) See Table 2-2.		Module
Register Address: D1AH, 3354	IA32_L2_MASK_10	
L2 CAT Mask for COS10 (R/W) See Table 2-2.		Module
Register Address: D1BH, 3355	IA32_L2_MASK_11	
L2 CAT Mask for COS11 (R/W) See Table 2-2.		Module
Register Address: D1CH, 3356	IA32_L2_MASK_12	
L2 CAT Mask for COS12 (R/W) See Table 2-2.		Module
Register Address: D1DH, 3357	IA32_L2_MASK_13	
L2 CAT Mask for COS13 (R/W) See Table 2-2.		Module
Register Address: D1EH, 3358	IA32_L2_MASK_14	
L2 CAT Mask for COS14 (R/W) See Table 2-2.		Module
Register Address: D1FH, 3359	IA32_L2_MASK_15	
L2 CAT Mask for COS15 (R/W) See Table 2-2.		Module
Register Address: 1878H, 6264	MSR_WORK_CONSERVING_CLOS	
Work Conserving CLOS (R/W)		Module
0	WC_VALID WC Valid Bit that indicates WC MSR has been setup. This bit must be set for the WC algorithm to be enabled.	
7:1	Reserved.	
11:8	CLOS_START_PRI1 Starting CLOS range for priority 1.	
15:12	CLOS_END_PRI1 Ending CLOS range for priority 1.	
19:16	CLOS_START_PRI2 Starting CLOS range for priority 2.	
23:20	CLOS_END_PRI2 Ending CLOS range for priority 2.	
27:24	CLOS_START_PRI3 Starting CLOS range for priority 3.	

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:28	CLOS_END_PRI3 Ending CLOS range for priority 3.	
63:32	Reserved.	
Register Address: 1903H, 6403	IA32_PMC_GP0_CFG_C	
IA32_PMC_GP0_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1907H, 6407	IA32_PMC_GP1_CFG_C	
IA32_PMC_GP1_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 190AH, 6410	IA32_PMC_GP2_CFG_B	
IA32_PMC_GP2_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 190BH, 6411	IA32_PMC_GP2_CFG_C	
IA32_PMC_GP2_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 190EH, 6414	IA32_PMC_GP3_CFG_B	
IA32_PMC_GP3_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 190FH, 6415	IA32_PMC_GP3_CFG_C	
IA32_PMC_GP3_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1912H, 6418	IA32_PMC_GP4_CFG_B	
IA32_PMC_GP4_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 1913H, 6419	IA32_PMC_GP4_CFG_C	
IA32_PMC_GP4_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1916H, 6422	IA32_PMC_GP5_CFG_B	
IA32_PMC_GP5_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 1917H, 6423	IA32_PMC_GP5_CFG_C	
IA32_PMC_GP5_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 191AH, 6426	IA32_PMC_GP6_CFG_B	
IA32_PMC_GP6_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 191BH, 6427	IA32_PMC_GP6_CFG_C	
IA32_PMC_GP6_CFG_C (R/W) See Table 2-2.		Thread

Table 2-60. MSRs Supported by the Intel® Series 2 Core™ Ultra Processor E-core (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 191EH, 6430	IA32_PMC_GP7_CFG_B	
IA32_PMC_GP7_CFG_B (R/W) See Table 2-2.		Thread
Register Address: 191FH, 6431	IA32_PMC_GP7_CFG_C	
IA32_PMC_GP7_CFG_C (R/W) See Table 2-2.		Thread
Register Address: 1982H, 6530	IA32_PMC_FX0_CFG_B	
Fixed-Function Counter Reload Configuration Register (R/W) See Table 2-2.		Thread
Register Address: 1983H, 6531	IA32_PMC_FX0_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 0 (R/W) See Table 2-2.		Thread
Register Address: 1986H, 6534	IA32_PMC_FX1_CFG_B	
Fixed-Function Counter Reload Configuration Register (R/W) See Table 2-2.		Thread
Register Address: 1987H, 6535	IA32_PMC_FX1_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 1 (R/W) See Table 2-2.		Thread
Register Address: 198BH, 6539	IA32_PMC_FX2_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 2 (R/W) See Table 2-2.		Thread
Register Address: 1990H, 6544	IA32_PMC_FX4_CTR	
Fixed-Function Performance Counter 4 - Top-down Bad Speculation (R/W) See Table 2-2.		Thread
Register Address: 1993H, 6547	IA32_PMC_FX4_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 4 (R/W) See Table 2-2.		Thread
Register Address: 1994H, 6548	IA32_PMC_FX5_CTR	
Fixed-Function Performance Counter 5 - Top-down Frontend Bound (R/W) See Table 2-2.		Thread
Register Address: 1997H, 6551	IA32_PMC_FX5_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 5 (R/W) See Table 2-2.		Thread
Register Address: 1998H, 6552	IA32_PMC_FX6_CTR	
Fixed-Function Performance Counter 5 - Top-down Bad Retiring (R/W) See Table 2-2.		Thread
Register Address: 199BH, 6555	IA32_PMC_FX6_CFG_C	
Extended Perf Event Selector for Fixed-Function Counter 6 (R/W) See Table 2-2.		Thread

2.18 MSRS IN THE INTEL® XEON PHI™ PROCESSOR 3200/5200/7200 SERIES AND THE INTEL® XEON PHI™ PROCESSOR 7215/7285/7295 SERIES

The Intel® Xeon Phi™ processor 3200, 5200, 7200 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_57H, supports the MSR interfaces listed in Table 2-61. These processors are based on the Knights Landing microarchitecture. The Intel® Xeon Phi™ processor 7215, 7285, 7295 series, with a CPUID Signature DisplayFamily_DisplayModel value of 06_85H, supports the MSR interfaces listed in Table 2-61 and Table 2-62. These processors are based on the Knights Mill microarchitecture. Some MSRs are shared between a pair of processor cores, and the scope is marked as module.

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 0H, 0	IA32_P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 1H, 1	IA32_P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors."		Module
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, "Monitor/Mwait Address Range Determination." See Table 2-2.		Thread
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, "Time-Stamp Counter," and Table 2-2.		Thread
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2.		Package
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 12.4.4, "Local APIC Status and Location," and Table 2-2.		Thread
Register Address: 34H, 52	MSR_SMI_COUNT	
SMI Counter (R/O)		Thread
31:0	SMI Count (R/O)	
63:32	Reserved.	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in Intel 64Processor (R/W) See Table 2-2.		Thread
0	Lock. (R/WL)	
1	Reserved.	
2	Enable VMX outside SMX operation. (R/WL)	
Register Address: 3BH, 59	IA32_TSC_ADJUST	
Per-Logical-Processor TSC ADJUST (R/W) See Table 2-2.		Thread
Register Address: 4EH, 78	IA32_PPIN_CTL (MSR_PPIN_CTL)	
Protected Processor Inventory Number Enable Control (R/W)		Package
0	LockOut (R/WO) See Table 2-2.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	Enable_PPIN (R/W) See Table 2-2.	
63:2	Reserved	
Register Address: 4FH, 79	IA32_PPIN (MSR_PPIN)	
Protected Processor Inventory Number (R/O)		Package
63:0	Protected Processor Inventory Number (R/O) See Table 2-2.	
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG	
BIOS Update Trigger Register (W) See Table 2-2.		Core
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID	
BIOS Update Signature ID (R/W) See Table 2-2.		Thread
Register Address: C1H, 193	IA32_PMC0	
Performance Counter Register See Table 2-2.		Thread
Register Address: C2H, 194	IA32_PMC1	
Performance Counter Register See Table 2-2.		Thread
Register Address: CEH, 206	MSR_PLATFORM_INFO	
Platform Information Contains power management and other model specific features enumeration. See http://biosbits.org .		Package
7:0	Reserved.	
15:8	Maximum Non-Turbo Ratio (R/O) This is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.	Package
27:16	Reserved.	
28	Programmable Ratio Limit for Turbo Mode (R/O) When set to 1, indicates that Programmable Ratio Limit for Turbo mode is enabled. When set to 0, indicates Programmable Ratio Limit for Turbo mode is disabled.	Package
29	Programmable TDP Limit for Turbo Mode (R/O) When set to 1, indicates that TDP Limit for Turbo mode is programmable. When set to 0, indicates TDP Limit for Turbo mode is not programmable.	Package
39:30	Reserved.	
47:40	Maximum Efficiency Ratio (R/O) This is the minimum ratio (maximum efficiency) that the processor can operate, in units of 100MHz.	Package
63:48	Reserved.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: E2H, 226	MSR_PKG_CST_CONFIG_CONTROL	
C-State Configuration Control (R/W)		Package
2:0	Package C-State Limit (R/W) Specifies the lowest C-state for the package. This feature does not limit the processor core C-state. The power-on default value from bit[2:0] of this register reports the deepest package C-state the processor is capable to support when manufactured. It is recommended that BIOS always read the power-on default value reported from this bit field to determine the supported deepest C-state on the processor and leave it as default without changing it. 000b - C0/C1 (No package C-state support) 001b - C2 010b - C6 (non retention)* 011b - C6 (Retention)* 100b - Reserved 101b - Reserved 110b - Reserved 111b - No package C-state limit. All C-States supported by the processor are available. Note: C6 retention mode provides more power saving than C6 non-retention mode. Limiting the package to C6 non retention mode does prevent the MSR_PKG_C6_RESIDENCY counter (MSR 3F9h) from being incremented.	
9:3	Reserved.	
10	I/O MWAIT Redirection Enable (R/W) When set, will map IO_read instructions sent to IO registers at MSR_PMG_IO_CAPTURE_BASE[15:0] to MWAIT instructions.	
14:11	Reserved.	
15	CFG Lock (R/O) When set, locks bits [15:0] of this register for further writes until the next reset occurs.	
25	Reserved.	
26	C1 State Auto Demotion Enable (R/W) When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.	
27	Reserved.	
28	C1 State Auto Undemotion Enable (R/W) When set, enables Undemotion from Demoted C1.	
29	PKG C-State Auto Demotion Enable (R/W) When set, enables Package C state demotion.	
63:30	Reserved.	
Register Address: E4H, 228	MSR_PMG_IO_CAPTURE_BASE	
Power Management IO Capture Base (R/W)		Tile

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
15:0	LVL_2 Base Address (R/W) Microcode will compare IO-read zone to this base address to determine if an MWAIT(C2/3/4) needs to be issued instead of the IO-read. Should be programmed to the chipset Plevel_2 IO address.	
22:16	C-State Range (R/W) The IO-port block size in which IO-redirection will be executed (0-127). Should be programmed based on the number of LVLx registers existing in the chipset.	
63:23	Reserved.	
Register Address: E7H, 231	IA32_MPERF	
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: E8H, 232	IA32_APERF	
Actual Performance Frequency Clock Count (R/W) See Table 2-2.		Thread
Register Address: FEH, 254	IA32_MTRRCAP	
Memory Type Range Register (R/O) See Table 2-2.		Core
Register Address: 13CH, 316	MSR_FEATURE_CONFIG	
AES Configuration (RW-L) Privileged post-BIOS agent must provide a #GP handler to handle unsuccessful read of this MSR.		Core
1:0	AES Configuration (RW-L) Upon a successful read of this MSR, the configuration of AES instruction set availability is as follows: 11b: AES instructions are not available until next RESET. Otherwise, AES instructions are available. Note, the AES instruction set is not available if read is unsuccessful. If the configuration is not 01b, AES instructions can be mis-configured if a privileged agent unintentionally writes 11b.	
63:2	Reserved.	
Register Address: 140H, 320	MISC_FEATURE_ENABLES	
MISC_FEATURE_ENABLES		Thread
0	Reserved.	
1	User Mode MONITOR and MWAIT (R/W) If set to 1, the MONITOR and MWAIT instructions do not cause invalid-opcode exceptions when executed with CPL > 0 or in virtual-8086 mode. If MWAIT is executed when CPL > 0 or in virtual-8086 mode, and if EAX indicates a C-state other than C0 or C1, the instruction operates as if EAX indicated the C-state C1.	
63:2	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Thread
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Thread
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Thread
Register Address: 17AH, 378	IA32_MCG_STATUS	
See Table 2-2.		Thread
Register Address: 17DH, 381	MSR_SMM_MCA_CAP	
Enhanced SMM Capabilities (SMM-RO) Reports SMM capability Enhancement. Accessible only while in SMM.		Thread
31:0	Bank Support (SMM-RO) One bit per MCA bank. If the bit is set, that bank supports Enhanced MCA (Default all 0; does not support EMCA).	
55:32	Reserved.	
56	Targeted SMI (SMM-RO) Set if targeted SMI is supported.	
57	SMM_CPU_SVRSTR (SMM-RO) Set if SMM SRAM save/restore feature is supported.	
58	SMM_CODE_ACCESS_CHK (SMM-RO) Set if SMM code access check feature is supported.	
59	Long_Flow_Indication (SMM-RO) If set to 1, indicates that the SMM long flow indicator is supported and a host-space interface available to SMM handler.	
63:60	Reserved.	
Register Address: 186H, 390	IA32_PERFECTSELO	
Performance Monitoring Event Select Register (R/W) See Table 2-2.		Thread
7:0	Event Select.	
15:8	UMask.	
16	USR.	
17	OS.	
18	Edge.	
19	PC.	
20	INT.	
21	AnyThread.	
22	EN.	
23	INV.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
31:24	CMASK.	
63:32	Reserved.	
Register Address: 187H, 391	IA32_PERFEVTSEL1	
See Table 2-2.		Thread
Register Address: 198H, 408	IA32_PERF_STATUS	
See Table 2-2.		Package
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Thread
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Thread
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2.		Module
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2.		Module
0	Thermal Status (R/O)	
1	Thermal Status Log (R/WCO)	
2	PROTCHOT # or FORCEPR# Status (R/O)	
3	PROTCHOT # or FORCEPR# Log (R/WCO)	
4	Critical Temperature Status (R/O)	
5	Critical Temperature Status Log (R/WCO)	
6	Thermal Threshold #1 Status (R/O)	
7	Thermal Threshold #1 Log (R/WCO)	
8	Thermal Threshold #2 Status (R/O)	
9	Thermal Threshold #2 Log (R/WCO)	
10	Power Limitation Status (R/O)	
11	Power Limitation Log (R/WCO)	
15:12	Reserved.	
22:16	Digital Readout (R/O)	
26:23	Reserved.	
30:27	Resolution in Degrees Celsius (R/O)	
31	Reading Valid (R/O)	
63:32	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Enable Misc. Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		Thread
0	Fast-Strings Enable	
2:1	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W)	
6:4	Reserved.	
7	Performance Monitoring Available (R)	
10:8	Reserved.	
11	Branch Trace Storage Unavailable (R/O)	
12	Processor Event Based Sampling Unavailable (R/O)	
15:13	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W)	
18	ENABLE MONITOR FSM (R/W)	
21:19	Reserved.	
22	Limit CPUID Maxval (R/W)	
23	xTPR Message Disable (R/W)	
33:24	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	
37:35	Reserved.	
38	Turbo Mode Disable (R/W)	
63:39	Reserved.	
Register Address: 1A2H, 418	MSR_TEMPERATURE_TARGET	
Temperature Target		Package
15:0	Reserved.	
23:16	Temperature Target (R)	
29:24	Target Offset (R/W)	
63:30	Reserved.	
Register Address: 1A4H, 420	MSR_MISC_FEATURE_CONTROL	
Miscellaneous Feature Control (R/W)		
0	DCU Hardware Prefetcher Disable (R/W) If 1, disables the L1 data cache prefetcher.	Core
1	L2 Hardware Prefetcher Disable (R/W) If 1, disables the L2 hardware prefetcher.	Core
63:2	Reserved.	
Register Address: 1A6H, 422	MSR_OFFCORE_RSP_0	
Offcore Response Event Select Register (R/W)		Shared

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 1A7H, 423	MSR_OFFCORE_RSP_1	
Offcore Response Event Select Register (R/W)		Shared
Register Address: 1ADH, 429	MSR_TURBO_RATIO_LIMIT	
Maximum Ratio Limit of Turbo Mode for Groups of Cores (R/W)		Package
0	Reserved.	
7:1	Maximum Number of Cores in Group 0 Number active processor cores which operates under the maximum ratio limit for group 0.	Package
15:8	Maximum Ratio Limit for Group 0 Maximum turbo ratio limit when the number of active cores are not more than the group 0 maximum core count.	Package
20:16	Number of Incremental Cores Added to Group 1 Group 1, which includes the specified number of additional cores plus the cores in group 0, operates under the group 1 turbo max ratio limit = "group 0 Max ratio limit" - "group ratio delta for group 1".	Package
23:21	Group Ratio Delta for Group 1 An unsigned integer specifying the ratio decrement relative to the Max ratio limit to Group 0.	Package
28:24	Number of Incremental Cores Added to Group 2 Group 2, which includes the specified number of additional cores plus all the cores in group 1, operates under the group 2 turbo max ratio limit = "group 1 Max ratio limit" - "group ratio delta for group 2".	Package
31:29	Group Ratio Delta for Group 2 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 1.	Package
36:32	Number of Incremental Cores Added to Group 3 Group 3, which includes the specified number of additional cores plus all the cores in group 2, operates under the group 3 turbo max ratio limit = "group 2 Max ratio limit" - "group ratio delta for group 3".	Package
39:37	Group Ratio Delta for Group 3 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 2.	Package
44:40	Number of Incremental Cores Added to Group 4 Group 4, which includes the specified number of additional cores plus all the cores in group 3, operates under the group 4 turbo max ratio limit = "group 3 Max ratio limit" - "group ratio delta for group 4".	Package
47:45	Group Ratio Delta for Group 4 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 3.	Package
52:48	Number of Incremental Cores Added to Group 5 Group 5, which includes the specified number of additional cores plus all the cores in group 4, operates under the group 5 turbo max ratio limit = "group 4 Max ratio limit" - "group ratio delta for group 5".	Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
55:53	Group Ratio Delta for Group 5 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 4.	Package
60:56	Number of Incremental Cores Added to Group 6 Group 6, which includes the specified number of additional cores plus all the cores in group 5, operates under the group 6 turbo max ratio limit = "group 5 Max ratio limit" - "group ratio delta for group 6".	Package
63:61	Group Ratio Delta for Group 6 An unsigned integer specifying the ratio decrement relative to the Max ratio limit for Group 5.	Package
Register Address: 1B0H, 432	IA32_ENERGY_PERF_BIAS	
See Table 2-2.		Thread
Register Address: 1B1H, 433	IA32_PACKAGE_THERM_STATUS	
See Table 2-2.		Package
Register Address: 1B2H, 434	IA32_PACKAGE_THERM_INTERRUPT	
See Table 2-2.		Package
Register Address: 1C8H, 456	MSR_LBR_SELECT	
Last Branch Record Filtering Select Register (R/W) See Section 19.9.2, "Filtering of Last Branch Records."		Thread
0	CPL_EQ_0	
1	CPL_NEQ_0	
2	JCC	
3	NEAR_REL_CALL	
4	NEAR_IND_CALL	
5	NEAR_RET	
6	NEAR_IND_JMP	
7	NEAR_REL_JMP	
8	FAR_BRANCH	
63:9	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP.		Thread
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W)		Thread
0	LBR Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
1	BTF Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	
5:2	Reserved.	
6	TR Setting this bit to 1 enables branch trace messages to be sent.	
7	BTS Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	
8	BTINT When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	
9	BTS_OFF_OS When set, BTS or BTM is skipped if CPL = 0.	
10	BTS_OFF_USR When set, BTS or BTM is skipped if CPL > 0.	
11	FREEZE_LBRS_ON_PMI When set, the LBR stack is frozen on a PMI request.	
12	FREEZE_PERFMON_ON_PMI When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request.	
13	Reserved.	
14	FREEZE_WHILE_SMM When set, freezes PerfMon and trace messages while in SMM.	
31:15	Reserved.	
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record from Linear IP (R)		Thread
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record to Linear IP (R)		Thread
Register Address: 1F2H, 498	IA32_SMRR_PHYSBASE	
See Table 2-2.		Core
Register Address: 1F3H, 499	IA32_SMRR_PHYSMASK	
See Table 2-2.		Core
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0	
See Table 2-2.		Core
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0	
See Table 2-2.		Core
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
See Table 2-2.		Core
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1	
See Table 2-2.		Core
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2	
See Table 2-2.		Core
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2	
See Table 2-2.		Core
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3	
See Table 2-2.		Core
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3	
See Table 2-2.		Core
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4	
See Table 2-2.		Core
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4	
See Table 2-2.		Core
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5	
See Table 2-2.		Core
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5	
See Table 2-2.		Core
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6	
See Table 2-2.		Core
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6	
See Table 2-2.		Core
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7	
See Table 2-2.		Core
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7	
See Table 2-2.		Core
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000	
See Table 2-2.		Core
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000	
See Table 2-2.		Core
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000	
See Table 2-2.		Core
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000	
See Table 2-2.		Core
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000	
See Table 2-2.		Core

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000	
See Table 2-2.		Core
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000	
See Table 2-2.		Core
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000	
See Table 2-2.		Core
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000	
See Table 2-2.		Core
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000	
See Table 2-2.		Core
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000	
See Table 2-2.		Core
Register Address: 277H, 631	IA32_PAT	
See Table 2-2.		Core
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2.		Core
Register Address: 309H, 777	IA32_FIXED_CTR0	
Fixed-Function Performance Counter Register 0 (R/W) See Table 2-2.		Thread
Register Address: 30AH, 778	IA32_FIXED_CTR1	
Fixed-Function Performance Counter Register 1 (R/W) See Table 2-2.		Thread
Register Address: 30BH, 779	IA32_FIXED_CTR2	
Fixed-Function Performance Counter Register 2 (R/W) See Table 2-2.		Thread
Register Address: 345H, 837	IA32_PERF_CAPABILITIES	
See Table 2-2. See Section 19.4.1, "IA32_DEBUGCTL MSR."		Package
Register Address: 38DH, 909	IA32_FIXED_CTR_CTRL	
Fixed-Function-Counter Control Register (R/W) See Table 2-2.		Thread
Register Address: 38EH, 910	IA32_PERF_GLOBAL_STATUS	
See Table 2-2.		Thread
Register Address: 38FH, 911	IA32_PERF_GLOBAL_CTRL	
See Table 2-2.		Thread
Register Address: 390H, 912	IA32_PERF_GLOBAL_OVF_CTRL	
See Table 2-2.		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)	
See Table 2-2.		Thread
Register Address: 3F8H, 1016	MSR_PKG_C3_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C3 Residency Counter (R/O)	
Register Address: 3F9H, 1017	MSR_PKG_C6_RESIDENCY	
63:0	Package C6 Residency Counter (R/O)	Package
Register Address: 3FAH, 1018	MSR_PKG_C7_RESIDENCY	
63:0	Package C7 Residency Counter (R/O)	Package
Register Address: 3FCH, 1020	MSR_MCO_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Module
63:0	Module C0 Residency Counter (R/O)	
Register Address: 3FDH, 1021	MSR_MC6_RESIDENCY	
63:0	Module C6 Residency Counter (R/O)	Module
Register Address: 3FFH, 1023	MSR_CORE_C6_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Core
63:0	CORE C6 Residency Counter (R/O)	
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 408H, 1032	IA32_MC2_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 40CH, 1036	IA32_MC3_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 40DH, 1037	IA32_MC3_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 40EH, 1038	IA32_MC3_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Core
Register Address: 410H, 1040	IA32_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Core
Register Address: 411H, 1041	IA32_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Core
Register Address: 412H, 1042	IA32_MC4_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Core
Register Address: 414H, 1044	IA32_MC5_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Package
Register Address: 415H, 1045	IA32_MC5_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Package
Register Address: 416H, 1046	IA32_MC5_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs."		Package
Register Address: 4C1H, 1217	IA32_A_PMC0	
See Table 2-2.		Thread
Register Address: 4C2H, 1218	IA32_A_PMC1	
See Table 2-2.		Thread
Register Address: 600H, 1536	IA32_DS_AREA	
DS Save Area (R/W) See Table 2-2.		Thread
Register Address: 606H, 1542	MSR_RAPL_POWER_UNIT	
Unit Multipliers Used in RAPL Interfaces (R/O)		Package
3:0	Power Units See Section 16.10.1, "RAPL Interfaces."	Package
7:4	Reserved.	Package
12:8	Energy Status Units Energy related information (in Joules) is based on the multiplier, $1/2^{\text{ESU}}$; where ESU is an unsigned integer represented by bits 12:8. Default value is 0EH (or 61 micro-joules).	Package
15:13	Reserved.	Package
19:16	Time Units See Section 16.10.1, "RAPL Interfaces."	Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
63:20	Reserved.	
Register Address: 60DH, 1549	MSR_PKG_C2_RESIDENCY	
Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-states.		Package
63:0	Package C2 Residency Counter (R/O)	
Register Address: 610H, 1552	MSR_PKG_POWER_LIMIT	
PKG RAPL Power Limit Control (R/W) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 611H, 1553	MSR_PKG_ENERGY_STATUS	
PKG Energy Status (R/O) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 613H, 1555	MSR_PKG_PERF_STATUS	
PKG Perf Status (R/O) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 614H, 1556	MSR_PKG_POWER_INFO	
PKG RAPL Parameters (R/W) See Section 16.10.3, "Package RAPL Domain."		Package
Register Address: 618H, 1560	MSR_DRAM_POWER_LIMIT	
DRAM RAPL Power Limit Control (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 619H, 1561	MSR_DRAM_ENERGY_STATUS	
DRAM Energy Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61BH, 1563	MSR_DRAM_PERF_STATUS	
DRAM Performance Throttling Status (R/O) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 61CH, 1564	MSR_DRAM_POWER_INFO	
DRAM RAPL Parameters (R/W) See Section 16.10.5, "DRAM RAPL Domain."		Package
Register Address: 638H, 1592	MSR_PPO_POWER_LIMIT	
PPO RAPL Power Limit Control (R/W) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 639H, 1593	MSR_PPO_ENERGY_STATUS	
PPO Energy Status (R/O) See Section 16.10.4, "PPO/PP1 RAPL Domains."		Package
Register Address: 648H, 1608	MSR_CONFIG_TDP_NOMINAL	
Base TDP Ratio (R/O) See Table 2-25.		Package

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 649H, 1609	MSR_CONFIG_TDP_LEVEL1	
ConfigTDP Level 1 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64AH, 1610	MSR_CONFIG_TDP_LEVEL2	
ConfigTDP Level 2 ratio and power level (R/O) See Table 2-25.		Package
Register Address: 64BH, 1611	MSR_CONFIG_TDP_CONTROL	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 64CH, 1612	MSR_TURBO_ACTIVATION_RATIO	
ConfigTDP Control (R/W) See Table 2-25.		Package
Register Address: 690H, 1680	MSR_CORE_PERF_LIMIT_REASONS	
Indicator of Frequency Clipping in Processor Cores (R/W) (Frequency refers to processor core frequency.)		Package
0	PROCHOT Status (R0)	
1	Thermal Status (R0)	
5:2	Reserved.	
6	VR Therm Alert Status (R0)	
7	Reserved.	
8	Electrical Design Point Status (R0)	
63:9	Reserved.	
Register Address: 6E0H, 1760	IA32_TSC_DEADLINE	
TSC Target of Local APIC's TSC Deadline Mode (R/W) See Table 2-2.		Core
Register Address: 802H, 2050	IA32_X2APIC_APICID	
x2APIC ID Register (R/O)		Thread
Register Address: 803H, 2051	IA32_X2APIC_VERSION	
x2APIC Version Register (R/O)		Thread
Register Address: 808H, 2056	IA32_X2APIC_TPR	
x2APIC Task Priority Register (R/W)		Thread
Register Address: 80AH, 2058	IA32_X2APIC_PPR	
x2APIC Processor Priority Register (R/O)		Thread
Register Address: 80BH, 2059	IA32_X2APIC_EOI	
x2APIC EOI Register (W/O)		Thread
Register Address: 80DH, 2061	IA32_X2APIC_LDR	
x2APIC Logical Destination Register (R/O)		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 80FH, 2063	IA32_X2APIC_SIVR	
x2APIC Spurious Interrupt Vector Register (R/W)		Thread
Register Address: 810H, 2064	IA32_X2APIC_ISR0	
x2APIC In-Service Register Bits [31:0] (R/O)		Thread
Register Address: 811H, 2065	IA32_X2APIC_ISR1	
x2APIC In-Service Register Bits [63:32] (R/O)		Thread
Register Address: 812H, 2066	IA32_X2APIC_ISR2	
x2APIC In-Service Register Bits [95:64] (R/O)		Thread
Register Address: 813H, 2067	IA32_X2APIC_ISR3	
x2APIC In-Service Register Bits [127:96] (R/O)		Thread
Register Address: 814H, 2068	IA32_X2APIC_ISR4	
x2APIC In-Service Register Bits [159:128] (R/O)		Thread
Register Address: 815H, 2069	IA32_X2APIC_ISR5	
x2APIC In-Service Register Bits [191:160] (R/O)		Thread
Register Address: 816H, 2070	IA32_X2APIC_ISR6	
x2APIC In-Service Register Bits [223:192] (R/O)		Thread
Register Address: 817H, 2071	IA32_X2APIC_ISR7	
x2APIC In-Service Register Bits [255:224] (R/O)		Thread
Register Address: 818H, 2072	IA32_X2APIC_TMR0	
x2APIC Trigger Mode Register Bits [31:0] (R/O)		Thread
Register Address: 819H, 2073	IA32_X2APIC_TMR1	
x2APIC Trigger Mode Register Bits [63:32] (R/O)		Thread
Register Address: 81AH, 2074	IA32_X2APIC_TMR2	
x2APIC Trigger Mode Register Bits [95:64] (R/O)		Thread
Register Address: 81BH, 2075	IA32_X2APIC_TMR3	
x2APIC Trigger Mode Register Bits [127:96] (R/O)		Thread
Register Address: 81CH, 2076	IA32_X2APIC_TMR4	
x2APIC Trigger Mode Register Bits [159:128] (R/O)		Thread
Register Address: 81DH, 2077	IA32_X2APIC_TMR5	
x2APIC Trigger Mode Register Bits [191:160] (R/O)		Thread
Register Address: 81EH, 2078	IA32_X2APIC_TMR6	
x2APIC Trigger Mode Register Bits [223:192] (R/O)		Thread
Register Address: 81FH, 2079	IA32_X2APIC_TMR7	
x2APIC Trigger Mode Register Bits [255:224] (R/O)		Thread
Register Address: 820H, 2080	IA32_X2APIC_IRR0	
x2APIC Interrupt Request Register Bits [31:0] (R/O)		Thread
Register Address: 821H, 2081	IA32_X2APIC_IRR1	

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
x2APIC Interrupt Request Register Bits [63:32] (R/O)		Thread
Register Address: 822H, 2082	IA32_X2APIC_IRR2	
x2APIC Interrupt Request Register Bits [95:64] (R/O)		Thread
Register Address: 823H, 2083	IA32_X2APIC_IRR3	
x2APIC Interrupt Request Register Bits [127:96] (R/O)		Thread
Register Address: 824H, 2084	IA32_X2APIC_IRR4	
x2APIC Interrupt Request Register Bits [159:128] (R/O)		Thread
Register Address: 825H, 2085	IA32_X2APIC_IRR5	
x2APIC Interrupt Request Register Bits [191:160] (R/O)		Thread
Register Address: 826H, 2086	IA32_X2APIC_IRR6	
x2APIC Interrupt Request Register Bits [223:192] (R/O)		Thread
Register Address: 827H, 2087	IA32_X2APIC_IRR7	
x2APIC Interrupt Request Register Bits [255:224] (R/O)		Thread
Register Address: 828H, 2088	IA32_X2APIC_ESR	
x2APIC Error Status Register (R/W)		Thread
Register Address: 82FH, 2095	IA32_X2APIC_LVT_CMCI	
x2APIC LVT Corrected Machine Check Interrupt Register (R/W)		Thread
Register Address: 830H, 2096	IA32_X2APIC_ICR	
x2APIC Interrupt Command Register (R/W)		Thread
Register Address: 832H, 2098	IA32_X2APIC_LVT_TIMER	
x2APIC LVT Timer Interrupt Register (R/W)		Thread
Register Address: 833H, 2099	IA32_X2APIC_LVT_THERMAL	
x2APIC LVT Thermal Sensor Interrupt Register (R/W)		Thread
Register Address: 834H, 2100	IA32_X2APIC_LVT_PMI	
x2APIC LVT Performance Monitor Register (R/W)		Thread
Register Address: 835H, 2101	IA32_X2APIC_LVT_LINT0	
x2APIC LVT LINT0 Register (R/W)		Thread
Register Address: 836H, 2102	IA32_X2APIC_LVT_LINT1	
x2APIC LVT LINT1 Register (R/W)		Thread
Register Address: 837H, 2103	IA32_X2APIC_LVT_ERROR	
x2APIC LVT Error Register (R/W)		Thread
Register Address: 838H, 2104	IA32_X2APIC_INIT_COUNT	
x2APIC Initial Count Register (R/W)		Thread
Register Address: 839H, 2105	IA32_X2APIC_CUR_COUNT	
x2APIC Current Count Register (R/O)		Thread
Register Address: 83EH, 2110	IA32_X2APIC_DIV_CONF	
x2APIC Divide Configuration Register (R/W)		Thread

Table 2-61. Selected MSRs Supported by Intel® Xeon Phi™ Processors with a CPUID Signature DisplayFamily_DisplayModel Value of 06_57H or 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 83FH, 2111	IA32_X2APIC_SELF_IPI	
x2APIC Self IPI Register (W/O)		Thread
Register Address: C000_0080H	IA32_EFER	
Extended Feature Enables See Table 2-2.		Thread
Register Address: C000_0081H	IA32_STAR	
System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0082H	IA32_LSTAR	
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		Thread
Register Address: C000_0084H	IA32_FMASK	
System Call Flag Mask (R/W) See Table 2-2.		Thread
Register Address: C000_0100H	IA32_FS_BASE	
Map of BASE Address of FS (R/W) See Table 2-2.		Thread
Register Address: C000_0101H	IA32_GS_BASE	
Map of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0102H	IA32_KERNEL_GS_BASE	
Swap Target of BASE Address of GS (R/W) See Table 2-2.		Thread
Register Address: C000_0103H	IA32_TSC_AUX	
AUXILIARY TSC Signature (R/W) See Table 2-2		Thread

Table 2-62 lists model-specific registers that are supported by the Intel® Xeon Phi™ processor 7215, 7285, 7295 series based on the Knights Mill microarchitecture.

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL	
SMM Monitor Configuration (R/W) This MSR is readable only if VMX is enabled, and writeable only if VMX is enabled and in SMM mode, and is used to configure the VMX MSEG base address. See Table 2-2.		Core
Register Address: 480H, 1152	IA32_VMX_BASIC	

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTLS	
Capability Reporting Register of Pin-based VM-execution Controls (R/O) See Table 2-2.		Core
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-based VM-execution Controls (R/O)		Core
Register Address: 483H, 1155	IA32_VMX_EXIT_CTLS	
Capability Reporting Register of VM-exit Controls (R/O) See Table 2-2.		Core
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTLS	
Capability Reporting Register of VM-entry Controls (R/O) See Table 2-2.		Core
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Table 2-2.		Core
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Table 2-2.		Core
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Table 2-2.		Core
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Table 2-2.		Core
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLS2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Table 2-2.		Core
Register Address: 48CH, 1164	IA32_VMX_EPT_VPID_ENUM	
Capability Reporting Register of EPT and VPID (R/O) See Table 2-2.		Core
Register Address: 48DH, 1165	IA32_VMX_TRUE_PINBASED_CTLS	

Table 2-62. Additional MSRs Supported by the Intel® Xeon Phi™ Processor 7215, 7285, 7295 Series with a CPUID Signature DisplayFamily_DisplayModel Value of 06_85H (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Scope
Capability Reporting Register of Pin-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48EH, 1166	IA32_VMX_TRUE_PROCBASED_CTLS	
Capability Reporting Register of Primary Processor-Based VM-Execution Flex Controls (R/O) See Table 2-2.		Core
Register Address: 48FH, 1167	IA32_VMX_TRUE_EXIT_CTLS	
Capability Reporting Register of VM-Exit Flex Controls (R/O) See Table 2-2.		Core
Register Address: 490H, 1168	IA32_VMX_TRUE_ENTRY_CTLS	
Capability Reporting Register of VM-Entry Flex Controls (R/O) See Table 2-2.		Core
Register Address: 491H, 1169	IA32_VMX_FMFUNC	
Capability Reporting Register of VM-Function Controls (R/O) See Table 2-2.		Core

2.19 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 2-63 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 2-1.

- MSRs with an "IA32_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/Unique ¹
Register Address: 0H, 0	IA32_P5_MC_ADDR		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 1H, 1	IA32_P5_MC_TYPE		
See Section 2.23, "MSRs in Pentium Processors."		0, 1, 2, 3, 4, 6	Shared
Register Address: 6H, 6	IA32_MONITOR_FILTER_LINE_SIZE		
See Section 10.10.5, "Monitor/Mwait Address Range Determination."		3, 4, 6	Shared
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER		
Time Stamp Counter See Table 2-2.		0, 1, 2, 3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.			
Register Address: 17H, 23	IA32_PLATFORM_ID		
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.		0, 1, 2, 3, 4, 6	Shared
Register Address: 1BH, 27	IA32_APIC_BASE		
APIC Location and Status (R/W) See Table 2-2. See Section 12.4.4, "Local APIC Status and Location."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2AH, 42	MSR_EBC_HARD_POWERON		
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.		0, 1, 2, 3, 4, 6	Shared
0	Output Tri-state Enabled (R) Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
1	Execute BIST (R) Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
2	In Order Queue Depth (R) Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
3	MCERR# Observation Disabled (R) Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
4	BINIT# Observation Enabled (R) Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
6:5	APIC Cluster ID (R) Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
7	Bus Park Disable (R) Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.		
11:8	Reserved.		
13:12	Agent ID (R) Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.		
63:14	Reserved.		
Register Address: 2BH, 43	MSR_EBC_SOFT_POWERON		
Processor Soft Power-On Configuration (R/W) Enables and disables processor features.		0, 1, 2, 3, 4, 6	Shared
0	RCNT/SCNT On Request Encoding Enable (R/W) Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).		
1	Data Error Checking Disable (R/W) Set to disable system data bus parity checking; clear to enable parity checking.		
2	Response Error Checking Disable (R/W) Set to disable (default); clear to enable.		
3	Address/Request Error Checking Disable (R/W) Set to disable (default); clear to enable.		
4	Initiator MCERR# Disable (R/W) Set to disable MCERR# driving for initiator bus requests (default); clear to enable.		
5	Internal MCERR# Disable (R/W) Set to disable MCERR# driving for initiator internal errors (default); clear to enable.		
6	BINIT# Driver Disable (R/W) Set to disable BINIT# driver (default); clear to enable driver.		
63:7	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
Processor Frequency Configuration The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.		2,3, 4, 6	Shared
15:0	Reserved.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
18:16	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6)		
	133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.		
	266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.		
23:19	Reserved.		
31:24	Core Clock Frequency to System Bus Frequency Ratio (R) The processor core clock frequency to system bus frequency ratio observed at the deassertion of the reset pin.		
63:32	Reserved.		
Register Address: 2CH, 44	MSR_EBC_FREQUENCY_ID		
Processor Frequency Configuration (R) The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.		0, 1	Shared
20:0	Reserved.		
23:21	Scalable Bus Speed (R/W) Indicates the intended scalable bus speed: <u>Encoding Scalable Bus Speed</u> 000B 100 MHz All others values reserved.		
63:24	Reserved.		
Register Address: 3AH, 58	IA32_FEATURE_CONTROL		
Control Features in IA-32 Processor (R/W) See Table 2-2. (If CPUID.01H:ECX[5])		3, 4, 6	Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
BIOS Update Trigger Register (W) See Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID		
BIOS Update Signature ID (R/W) See Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 9BH, 155	IA32_SMM_MONITOR_CTL		
SMM Monitor Configuration (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: FEH, 254	IA32_MTRRCAP		
MTRR Information See Section 13.11.1, "MTRR Feature Identification."		0, 1, 2, 3, 4, 6	Unique
Register Address: 174H, 372	IA32_SYSENTER_CS		
CS Register Target for CPL 0 Code (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP		
Stack Pointer for CPL 0 Stack (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP		
CPL 0 Code Entry Point (R/W) See Table 2-2 and Section 6.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."		0, 1, 2, 3, 4, 6	Unique
Register Address: 179H, 377	IA32_MCG_CAP		
Machine Check Capabilities (R) See Table 2-2 and Section 17.3.1.1, "IA32_MCG_CAP MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17AH, 378	IA32_MCG_STATUS		
Machine Check Status (R) See Table 2-2 and Section 17.3.1.2, "IA32_MCG_STATUS MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 17BH, 379	IA32_MCG_CTL		
Machine Check Feature Enable (R/W) See Table 2-2 and Section 17.3.1.3, "IA32_MCG_CTL MSR."			
Register Address: 180H, 384	MSR_MCG_RAX		
Machine Check EAX/RAX Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 181H, 385	MSR_MCG_RBX		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Machine Check EBX/RBX Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 182H, 386	MSR_MCG_RCX		
Machine Check ECX/RCX Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 183H, 387	MSR_MCG_RDX		
Machine Check EDX/RDX Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 184H, 388	MSR_MCG_RSI		
Machine Check ESI/RSI Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 185H, 389	MSR_MCG_RDI		
Machine Check EDI/RDI Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 186H, 390	MSR_MCG_RBP		
Machine Check EBP/RBP Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 187H, 391	MSR_MCG_RSP		
Machine Check ESP/RSP Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 188H, 392	MSR_MCG_RFLAGS		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Machine Check EFLAGS/RFLAG Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 189H, 393	MSR_MCG_RIP		
Machine Check EIP/RIP Save State See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.		
Register Address: 18AH, 394	MSR_MCG_MISC		
Machine Check Miscellaneous See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
0	DS When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.		
63:1	Reserved.		
Register Address: 18BH–18FH, 395–399	MSR_MCG_RESERVED1–MSR_MCG_RESERVED5		
Reserved.			
Register Address: 190H, 400	MSR_MCG_R8		
Machine Check R8 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 191H, 401	MSR_MCG_R9		
Machine Check R9D/R9 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 192H, 402	MSR_MCG_R10		
Machine Check R10 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 193H, 403	MSR_MCG_R11		
Machine Check R11 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 194H, 404	MSR_MCG_R12		
Machine Check R12 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 195H, 405	MSR_MCG_R13		
Machine Check R13 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 196H, 406	MSR_MCG_R14		
Machine Check R14 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 197H, 407	MSR_MCG_R15		
Machine Check R15 See Section 17.3.2.6, "IA32_MCG Extended Machine Check State MSRs."		0, 1, 2, 3, 4, 6	Unique
63:0	Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.		
Register Address: 198H, 408	IA32_PERF_STATUS		
See Table 2-2. See Section 16.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique
Register Address: 199H, 409	IA32_PERF_CTL		
See Table 2-2. See Section 16.1, "Enhanced Intel Speedstep® Technology."		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 19AH, 410	IA32_CLOCK_MODULATION		
Thermal Monitor Control (R/W) See Table 2-2 and Section 16.8.3, "Software Controlled Clock Modulation."		0, 1, 2, 3, 4, 6	Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT		
Thermal Interrupt Control (R/W) See Section 16.8.2, "Thermal Monitor," and Table 2-2.		0, 1, 2, 3, 4, 6	Unique
Register Address: 19CH, 412	IA32_THERM_STATUS		
Thermal Monitor Status (R/W) See Section 16.8.2, "Thermal Monitor," and Table 2-2.		0, 1, 2, 3, 4, 6	Shared
Register Address: 19DH, 413	MSR_THERM2_CTL		
Thermal Monitor 2 Control			
For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.		3	Shared
For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.		4, 6	Shared
Register Address: 1A0H, 416	IA32_MISC_ENABLE		
Enable Miscellaneous Processor Features (R/W)		0, 1, 2, 3, 4, 6	Shared
0	Fast-Strings Enable. See Table 2-2.		
1	Reserved.		
2	x87 FPU Fopcode Compatibility Mode Enable		
3	Thermal Monitor 1 Enable See Section 16.8.2, "Thermal Monitor," and Table 2-2.		
4	Split-Lock Disable When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (default), normal split-locks are issued to the bus. This debug feature is specific to the Pentium 4 processor.		
5	Reserved.		
6	Third-Level Cache Disable (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache. Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs. See Section 13.5.4, "Disabling and Enabling the L3 Cache."		
7	Performance Monitoring Available (R) See Table 2-2.		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
8	Suppress Lock Enable When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.		
9	Prefetch Queue Disable When set, disables the prefetch queue. When clear (default), enables the prefetch queue.		
10	FERR# Interrupt Reporting Enable (R/W) When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled. When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt. This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.		
11	Branch Trace Storage Unavailable (BTS_UNAVAILABLE) (R) See Table 2-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.		
12	PEBS_UNAVAILABLE: Processor Event Based Sampling Unavailable (R) See Table 2-2. When set, the processor does not support processor event-based sampling (PEBS); when clear, PEBS is supported.		
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.	3	
17:14	Reserved.		
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	3, 4, 6	

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
19	Adjacent Cache Line Prefetch Disable (R/W) When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector. Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.		
21:20	Reserved.		
22	Limit CPUID MAXVAL (R/W) See Table 2-2. Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.	3, 4, 6	
23	xTPR Message Disable (R/W) See Table 2-2.		Shared
24	L1 Data Cache Context Mode (R/W) When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 13.5.6, "L1 Data Cache Context Mode." When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].		
33:25	Reserved.		
34	XD Bit Disable (R/W) See Table 2-3.		Unique
63:35	Reserved.		
Register Address: 1A1H, 417	MSR_PLATFORM_BRV		
Platform Feature Requirements (R)		3, 4, 6	Shared
17:0	Reserved.		
18	PLATFORM Requirements When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.		
63:19	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 19.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D7H, 471	MSR_LER_FROM_LIP		
63:0	From Linear IP Linear address of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 19.13.3, "Last Exception Records."		0, 1, 2, 3, 4, 6	Unique
31:0	From Linear IP Linear address of the target of the last branch instruction.		
63:32	Reserved.		
Register Address: 1D8H, 472	MSR_LER_TO_LIP		
63:0	From Linear IP Linear address of the target of the last branch instruction (If IA-32e mode is active).		Unique
Register Address: 1D9H, 473	MSR_DEBUGCTLA		
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 19.13.1, "MSR_DEBUGCTLA MSR."		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DAH, 474	MSR_LASTBRANCH_TOS		
Last Branch Record Stack TOS (R/O) Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 19.13.2, "LBR Stack for Processors Based on Intel NetBurst® Microarchitecture," and addresses 1DBH-1DEH and 680H-68FH.		0, 1, 2, 3, 4, 6	Unique
Register Address: 1DBH, 475	MSR_LASTBRANCH_0		
Last Branch Record 0 (R/O) One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH. See Section 19.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		0, 1, 2	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 1DCH, 476	MSR_LASTBRANCH_1		
Last Branch Record 1 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DDH, 477	MSR_LASTBRANCH_2		
Last Branch Record 2 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 1DEH, 478	MSR_LASTBRANCH_3		
Last Branch Record 3 See description of the MSR_LASTBRANCH_0 MSR at 1DBH.		0, 1, 2	Unique
Register Address: 200H, 512	IA32_MTRR_PHYSBASE0		
Variable Range Base MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 201H, 513	IA32_MTRR_PHYSMASK0		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 202H, 514	IA32_MTRR_PHYSBASE1		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 203H, 515	IA32_MTRR_PHYSMASK1		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 204H, 516	IA32_MTRR_PHYSBASE2		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 205H, 517	IA32_MTRR_PHYSMASK2		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 206H, 518	IA32_MTRR_PHYSBASE3		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 207H, 519	IA32_MTRR_PHYSMASK3		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 208H, 520	IA32_MTRR_PHYSBASE4		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 209H, 521	IA32_MTRR_PHYSMASK4		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20AH, 522	IA32_MTRR_PHYSBASE5		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20BH, 523	IA32_MTRR_PHYSMASK5		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20CH, 524	IA32_MTRR_PHYSBASE6		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20DH, 525	IA32_MTRR_PHYSMASK6		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20EH, 526	IA32_MTRR_PHYSBASE7		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 20FH, 527	IA32_MTRR_PHYSMASK7		
Variable Range Mask MTRR See Section 13.11.2.3, "Variable Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 250H, 592	IA32_MTRR_FIX64K_00000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 258H, 600	IA32_MTRR_FIX16K_80000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 259H, 601	IA32_MTRR_FIX16K_A0000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 268H, 616	IA32_MTRR_FIX4K_C0000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 269H, 617	IA32_MTRR_FIX4K_C8000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26AH, 618	IA32_MTRR_FIX4K_D0000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 26BH, 619	IA32_MTRR_FIX4K_D8000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26CH, 620	IA32_MTRR_FIX4K_E0000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26DH, 621	IA32_MTRR_FIX4K_E8000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26EH, 622	IA32_MTRR_FIX4K_F0000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 26FH, 623	IA32_MTRR_FIX4K_F8000		
Fixed Range MTRR See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 277H, 631	IA32_PAT		
Page Attribute Table See Section 13.11.2.2, "Fixed Range MTRRs."		0, 1, 2, 3, 4, 6	Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE		
Default Memory Types (R/W) See Table 2-2 and Section 13.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		0, 1, 2, 3, 4, 6	Shared
Register Address: 300H, 768	MSR_BPU_COUNTER0		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 301H, 769	MSR_BPU_COUNTER1		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 302H, 770	MSR_BPU_COUNTER2		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 303H, 771	MSR_BPU_COUNTER3		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 304H, 772	MSR_MS_COUNTER0		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 305H, 773	MSR_MS_COUNTER1		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 306H, 774	MSR_MS_COUNTER2		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 307H, 775	MSR_MS_COUNTER3		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 308H, 776	MSR_FLAME_COUNTER0		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 309H, 777	MSR_FLAME_COUNTER1		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30AH, 778	MSR_FLAME_COUNTER2		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30BH, 779	MSR_FLAME_COUNTER3		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30CH, 780	MSR_IQ_COUNTER0		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30DH, 781	MSR_IQ_COUNTER1		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30EH, 782	MSR_IQ_COUNTER2		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 30FH, 783	MSR_IQ_COUNTER3		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 310H, 784	MSR_IQ_COUNTER4		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 311H, 785	MSR_IQ_COUNTER5		
See Section 21.6.3.2, "Performance Counters."		0, 1, 2, 3, 4, 6	Shared
Register Address: 360H, 864	MSR_BPU_CCCR0		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 361H, 865	MSR_BPU_CCCR1		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 362H, 866	MSR_BPU_CCCR2		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 363H, 867	MSR_BPU_CCCR3		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 364H, 868	MSR_MS_CCCR0		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 365H, 869	MSR_MS_CCCR1		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 366H, 870	MSR_MS_CCCR2		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 367H, 871	MSR_MS_CCCR3		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 368H, 872	MSR_FLAME_CCCR0		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 369H, 873	MSR_FLAME_CCCR1		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36AH, 874	MSR_FLAME_CCCR2		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36BH, 875	MSR_FLAME_CCCR3		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36CH, 876	MSR_IQ_CCCR0		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36DH, 877	MSR_IQ_CCCR1		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36EH, 878	MSR_IQ_CCCR2		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 36FH, 879	MSR_IQ_CCCR3		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 370H, 880	MSR_IQ_CCCR4		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 371H, 881	MSR_IQ_CCCR5		
See Section 21.6.3.3, "CCCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A0H, 928	MSR_BSU_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A1H, 929	MSR_BSU_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A2H, 930	MSR_FSB_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A3H, 931	MSR_FSB_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A4H, 932	MSR_FIRM_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A5H, 933	MSR_FIRM_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A6H, 934	MSR_FLAME_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A7H, 935	MSR_FLAME_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A8H, 936	MSR_DAC_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3A9H, 937	MSR_DAC_ESCR1		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AAH, 938	MSR_MOB_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ABH, 939	MSR_MOB_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ACH, 940	MSR_PMH_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3ADH, 941	MSR_PMH_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AEH, 942	MSR_SAAT_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3AFH, 943	MSR_SAAT_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BOH, 944	MSR_U2L_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B1H, 945	MSR_U2L_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B2H, 946	MSR_BPU_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B3H, 947	MSR_BPU_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B4H, 948	MSR_IS_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B5H, 949	MSR_IS_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B6H, 950	MSR_ITLB_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B7H, 951	MSR_ITLB_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B8H, 952	MSR_CRU_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3B9H, 953	MSR_CRU_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BAH, 954	MSR_IQ_ESCR0		
See Section 21.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.		0, 1, 2	Shared

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 3BBH, 955	MSR_IQ_ESCR1		
See Section 21.6.3.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.		0, 1, 2	Shared
Register Address: 3BCH, 956	MSR_RAT_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BDH, 957	MSR_RAT_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3BEH, 958	MSR_SSU_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3COH, 960	MSR_MS_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C1H, 961	MSR_MS_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C2H, 962	MSR_TBPU_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C3H, 963	MSR_TBPU_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C4H, 964	MSR_TC_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C5H, 965	MSR_TC_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C8H, 968	MSR_IX_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3C9H, 969	MSR_IX_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CAH, 970	MSR_ALF_ESCR0		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CBH, 971	MSR_ALF_ESCR1		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CCH, 972	MSR_CRU_ESCR2		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3CDH, 973	MSR_CRU_ESCR3		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E0H, 992	MSR_CRU_ESCR4		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3E1H, 993	MSR_CRU_ESCR5		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F0H, 1008	MSR_TC_PRECISE_EVENT		
See Section 21.6.3.1, "ESCR MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 3F1H, 1009	IA32_PEBS_ENABLE (MSR_PEBS_ENABLE)		
Processor Event Based Sampling (PEBS) (R/W) Controls the enabling of processor event sampling and replay tagging.		0, 1, 2, 3, 4, 6	Shared
12:0	See https://perfmon-events.intel.com/ .		
23:13	Reserved.		
24	UOP Tag Enables replay tagging when set.		
25	ENABLE_PEBS_MY_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 21.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Intel Hyper-Threading Technology.		
26	ENABLE_PEBS_OTH_THR (R/W) Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 21.6.4.3, "IA32_PEBS_ENABLE MSR," for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.		
63:27	Reserved.		
Register Address: 3F2H, 1010	MSR_PEBS_MATRIX_VERT		
See https://perfmon-events.intel.com/ .		0, 1, 2, 3, 4, 6	Shared
Register Address: 400H, 1024	IA32_MCO_CTL		
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 401H, 1025	IA32_MCO_STATUS		
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 402H, 1026	IA32_MCO_ADDR		
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 403H, 1027	IA32_MCO_MISC		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 17.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC0_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 404H, 1028	IA32_MC1_CTL		
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 405H, 1029	IA32_MC1_STATUS		
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 406H, 1030	IA32_MC1_ADDR		
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 407H, 1031	IA32_MC1_MISC		
See Section 17.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			Shared
Register Address: 408H, 1032	IA32_MC2_CTL		
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 409H, 1033	IA32_MC2_STATUS		
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40AH, 1034	IA32_MC2_ADDR		
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 40BH, 1035	IA32_MC2_MISC		
See Section 17.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 40CH, 1036	IA32_MC3_CTL		
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40DH, 1037	IA32_MC3_STATUS		
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 40EH, 1038	IA32_MC3_ADDR		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 40FH, 1039	IA32_MC3_MISC		
See Section 17.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		0, 1, 2, 3, 4, 6	Shared
Register Address: 410H, 1040	IA32_MC4_CTL		
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 411H, 1041	IA32_MC4_STATUS		
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		0, 1, 2, 3, 4, 6	Shared
Register Address: 412H, 1042	IA32_MC4_ADDR		
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 413H, 1043	IA32_MC4_MISC		
See Section 17.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.			
Register Address: 480H, 1152	IA32_VMX_BASIC		
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information."		3, 4, 6	Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL		
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Table 2-2 and Appendix A.3, "VM-Execution Controls."		3, 4, 6	Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL		
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL		
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 485H, 1157	IA32_VMX_MISC		
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data," and Table 2-2.		3, 4, 6	Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0		
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1		
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0," and Table 2-2.		3, 4, 6	Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0		
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.		3, 4, 6	Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1		
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4," and Table 2-2.		3, 4, 6	Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM		
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration," and Table 2-2.		3, 4, 6	Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTLSS2		
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls," and Table 2-2.		3, 4, 6	Unique
Register Address: 600H, 1536	IA32_DS_AREA		
DS Save Area (R/W) See Table 2-2 and Section 21.6.3.4, "Debug Store (DS) Mechanism."		0, 1, 2, 3, 4, 6	Unique
Register Address: 680H, 1664	MSR_LASTBRANCH_0_FROM_IP		
Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor. The MSRs at 680H-68FH, 6C0H-6CFH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH, which performed the same function for early releases. See Section 19.12, "Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture."		3, 4, 6	Unique
Register Address: 681H, 1665	MSR_LASTBRANCH_1_FROM_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 682H, 1666	MSR_LASTBRANCH_2_FROM_IP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 683H, 1667	MSR_LASTBRANCH_3_FROM_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 684H, 1668	MSR_LASTBRANCH_4_FROM_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 685H, 1669	MSR_LASTBRANCH_5_FROM_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 686H, 1670	MSR_LASTBRANCH_6_FROM_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 687H, 1671	MSR_LASTBRANCH_7_FROM_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 688H, 1672	MSR_LASTBRANCH_8_FROM_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 689H, 1673	MSR_LASTBRANCH_9_FROM_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68AH, 1674	MSR_LASTBRANCH_10_FROM_IP		
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68BH, 1675	MSR_LASTBRANCH_11_FROM_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68CH, 1676	MSR_LASTBRANCH_12_FROM_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68DH, 1677	MSR_LASTBRANCH_13_FROM_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 68EH, 1678	MSR_LASTBRANCH_14_FROM_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Register Address: 68FH, 1679	MSR_LASTBRANCH_15_FROM_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 680H.		3, 4, 6	Unique
Register Address: 6C0H, 1728	MSR_LASTBRANCH_0_TO_IP		
Last Branch Record 0 (R/W) One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 19.12, “Last Branch, Call Stack, Interrupt, and Exception Recording for Processors based on Skylake Microarchitecture.”		3, 4, 6	Unique
Register Address: 6C1H, 1729	MSR_LASTBRANCH_1_TO_IP		
Last Branch Record 1 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C2H, 1730	MSR_LASTBRANCH_2_TO_IP		
Last Branch Record 2 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C3H, 1731	MSR_LASTBRANCH_3_TO_IP		
Last Branch Record 3 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C4H, 1732	MSR_LASTBRANCH_4_TO_IP		
Last Branch Record 4 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C5H, 1733	MSR_LASTBRANCH_5_TO_IP		
Last Branch Record 5 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C6H, 1734	MSR_LASTBRANCH_6_TO_IP		
Last Branch Record 6 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C7H, 1735	MSR_LASTBRANCH_7_TO_IP		
Last Branch Record 7 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C8H, 1736	MSR_LASTBRANCH_8_TO_IP		
Last Branch Record 8 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6C9H, 1737	MSR_LASTBRANCH_9_TO_IP		
Last Branch Record 9 See description of MSR_LASTBRANCH_0 at 6C0H.		3, 4, 6	Unique
Register Address: 6CAH, 1738	MSR_LASTBRANCH_10_TO_IP		

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
Last Branch Record 10 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CBH, 1739	MSR_LASTBRANCH_11_TO_IP		
Last Branch Record 11 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CCH, 1740	MSR_LASTBRANCH_12_TO_IP		
Last Branch Record 12 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CDH, 1741	MSR_LASTBRANCH_13_TO_IP		
Last Branch Record 13 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CEH, 1742	MSR_LASTBRANCH_14_TO_IP		
Last Branch Record 14 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: 6CFH, 1743	MSR_LASTBRANCH_15_TO_IP		
Last Branch Record 15 See description of MSR_LASTBRANCH_0 at 6COH.		3, 4, 6	Unique
Register Address: C000_0080H	IA32_EFER		
Extended Feature Enables See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0081H	IA32_STAR		
System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0082H	IA32_LSTAR		
IA-32e Mode System Call Target Address (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0084H	IA32_FMASK		
System Call Flag Mask (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0100H	IA32_FS_BASE		
Map of BASE Address of FS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0101H	IA32_GS_BASE		
Map of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique
Register Address: C000_0102H	IA32_KERNEL_GS_BASE		
Swap Target of BASE Address of GS (R/W) See Table 2-2.		3, 4, 6	Unique

Table 2-63. MSRs in the Pentium® 4 and Intel® Xeon® Processors (Contd.)

Register Address: Hex, Decimal	Register Name		
Register Information / Bit Fields	Bit Description	Model Availability	Shared/ Unique ¹
NOTES 1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.			

2.19.1 MSRs Unique to Intel® Xeon® Processor MP with L3 Cache

The MSRs listed in Table 2-64 apply to Intel® Xeon® Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf, CPUID.04H, to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

Table 2-64. MSRs Unique to 64-bit Intel® Xeon® Processor MP with Up to an 8 MB L3 Cache

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/ Unique
Register Address: 107CCH	MSR_IFSB_BUSQ0		
IFSB BUSQ Event Control and Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107CDH	MSR_IFSB_BUSQ1		
IFSB BUSQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107CEH	MSR_IFSB_SNPQ0		
IFSB SNPQ Event Control and Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107CFH	MSR_IFSB_SNPQ1		
IFSB SNPQ Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D0H	MSR_EFSB_DRDY0		
EFSB DRDY Event Control and Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107D1H	MSR_EFSB_DRDY1		
EFSB DRDY Event Control and Counter Register (R/W)		3, 4	Shared
Register Address: 107D2H	MSR_IFSB_CTL6		
IFSB Latency Event Control Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared
Register Address: 107D3H	MSR_IFSB_CNTR7		
IFSB Latency Event Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		3, 4	Shared

The MSRs listed in Table 2-65 apply to Intel® Xeon® Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter, CPUID.04H, to detect the presence of the third level cache, and

with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 2-65 are shared between logical processors in the same core, but are replicated for each core.

Table 2-65. MSRs Unique to Intel® Xeon® Processor 7100 Series

Register Address: Hex	Register Name		
Register Information		Model Availability	Shared/Unique
Register Address: 107CCH	MSR_EMON_L3_CTR_CTL0		
GBUSQ Event Control and Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107CDH	MSR_EMON_L3_CTR_CTL1		
GBUSQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107CEH	MSR_EMON_L3_CTR_CTL2		
GSPNQ Event Control and Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107CFH	MSR_EMON_L3_CTR_CTL3		
GSPNQ Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D0H	MSR_EMON_L3_CTR_CTL4		
FSB Event Control and Counter Register (R/W) See Section 21.6.6, "Performance Monitoring on 64-bit Intel® Xeon® Processor MP with Up to 8-MByte L3 Cache."		6	Shared
Register Address: 107D1H	MSR_EMON_L3_CTR_CTL5		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D2H	MSR_EMON_L3_CTR_CTL6		
FSB Event Control and Counter Register (R/W)		6	Shared
Register Address: 107D3H	MSR_EMON_L3_CTR_CTL7		
FSB Event Control and Counter Register (R/W)		6	Shared

2.20 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 2-66. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/Unique
Register Address: 0H, 0	P5_MC_ADDR	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique
Register Address: 1H, 1	P5_MC_TYPE	
See Section 2.23, "MSRs in Pentium Processors," and Table 2-2.		Unique

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 6H, 6	IA32_MONITOR_FILTER_SIZE	
See Section 10.10.5, "Monitor/Mwait Address Range Determination," and Table 2-2.		Unique
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER	
See Section 19.17, "Time-Stamp Counter," and Table 2-2.		Unique
Register Address: 17H, 23	IA32_PLATFORM_ID	
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.		Shared
Register Address: 1BH, 27	IA32_APIC_BASE	
See Section 12.4.4, "Local APIC Status and Location," and Table 2-2.		Unique
Register Address: 2AH, 42	MSR_EBL_CR_POWERON	
Processor Hard Power-On Configuration (R/W) Enables and disables processor features; (R) indicates current processor configuration.		Shared
0	Reserved.	
1	Data Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
2	Response Error Checking Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
3	MCERR# Drive Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
4	Address Parity Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
6: 5	Reserved.	
7	BINIT# Driver Enable (R/W) 1 = Enabled; 0 = Disabled. Note: Not all processor implements R/W.	
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.	
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.	
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	
11	Reserved.	
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled.	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
13	Reserved	
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes	
15	Reserved.	
17:16	APIC Cluster ID (R/O)	
18	System Bus Frequency (R/O) 0 = 100 MHz. 1 = Reserved.	
19	Reserved.	
21: 20	Symmetric Arbitration ID (R/O)	
26:22	Clock Frequency Ratio (R/O)	
Register Address: 3AH, 58	IA32_FEATURE_CONTROL	
Control Features in IA-32 Processor (R/W) See Table 2-2.		Unique
Register Address: 40H, 64	MSR_LASTBRANCH_0	
Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: - Last Branch Record Stack TOS at 1C9H. - Section 19.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."		Unique
Register Address: 41H, 65	MSR_LASTBRANCH_1	
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 42H, 66	MSR_LASTBRANCH_2	
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 43H, 67	MSR_LASTBRANCH_3	
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 44H, 68	MSR_LASTBRANCH_4	
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 45H, 69	MSR_LASTBRANCH_5	
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 46H, 70	MSR_LASTBRANCH_6	
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.		Unique
Register Address: 47H, 71	MSR_LASTBRANCH_7	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal		Register Name	
Register Information / Bit Fields		Bit Description	Shared/ Unique
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.			Unique
Register Address: 79H, 121	IA32_BIOS_UPDT_TRIG		
BIOS Update Trigger Register (W) See Table 2-2.			Unique
Register Address: 8BH, 139	IA32_BIOS_SIGN_ID		
BIOS Update Signature ID (R/W) See Table 2-2.			Unique
Register Address: C1H, 193	IA32_PMC0		
Performance Counter Register See Table 2-2.			Unique
Register Address: C2H, 194	IA32_PMC1		
Performance Counter Register See Table 2-2.			Unique
Register Address: CDH, 205	MSR_FSB_FREQ		
Scaleable Bus Speed (R/O) This field indicates the scalable bus clock speed.			Shared
2:0	101B: 100 MHz (FSB 400) 001B: 133 MHz (FSB 533) 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.		
63:3	Reserved.		
Register Address: E7H, 231	IA32_MPERF		
Maximum Performance Frequency Clock Count (R/W) See Table 2-2.			Unique
Register Address: E8H, 232	IA32_APERF		
Actual Performance Frequency Clock Count (R/W) See Table 2-2.			Unique
Register Address: FEH, 254	IA32_MTRRCAP		
See Table 2-2.			Unique
Register Address: 11EH, 281	MSR_BBL_CR_CTL3		
Control Register 3 Used to configure the L2 Cache.			Shared
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.		

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
7:1	Reserved.	
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.	
22:9	Reserved.	
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.	
63:24	Reserved.	
Register Address: 174H, 372	IA32_SYSENTER_CS	
See Table 2-2.		Unique
Register Address: 175H, 373	IA32_SYSENTER_ESP	
See Table 2-2.		Unique
Register Address: 176H, 374	IA32_SYSENTER_EIP	
See Table 2-2.		Unique
Register Address: 179H, 377	IA32_MCG_CAP	
See Table 2-2.		Unique
Register Address: 17AH, 378	IA32_MCG_STATUS	
Global Machine Check Status		Unique
0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.	
1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.	
2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.	
63:3	Reserved	
Register Address: 186H, 390	IA32_PERFVTSELO	
See Table 2-2.		Unique
Register Address: 187H, 391	IA32_PERFVTSEL1	
See Table 2-2.		Unique
Register Address: 198H, 408	IA32_PERF_STATUS	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Table 2-2.		Shared
Register Address: 199H, 409	IA32_PERF_CTL	
See Table 2-2.		Unique
Register Address: 19AH, 410	IA32_CLOCK_MODULATION	
Clock Modulation (R/W) See Table 2-2.		Unique
Register Address: 19BH, 411	IA32_THERM_INTERRUPT	
Thermal Interrupt Control (R/W) See Table 2-2 and Section 16.8.2, "Thermal Monitor."		Unique
Register Address: 19CH, 412	IA32_THERM_STATUS	
Thermal Monitor Status (R/W) See Table 2-2 and Section 16.8.2, "Thermal Monitor".		Unique
Register Address: 19DH, 413	MSR_THERM2_CTL	
Thermal Monitor 2 Control		Unique
15:0	Reserved.	
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.	
63:16	Reserved.	
Register Address: 1A0H, 416	IA32_MISC_ENABLE	
Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.		
2:0	Reserved.	
3	Automatic Thermal Control Circuit Enable (R/W) See Table 2-2.	Unique
6:4	Reserved.	
7	Performance Monitoring Available (R) See Table 2-2.	Shared
9:8	Reserved.	
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.	Shared
11	Branch Trace Storage Unavailable (R/O) See Table 2-2.	Shared

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
12	Reserved.	
13	TM2 Enable (R/W) When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0. When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.	Shared
15:14	Reserved.	
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled	Shared
18	ENABLE MONITOR FSM (R/W) See Table 2-2.	Shared
19	Reserved.	
22	Limit CPUID Maxval (R/W) See Table 2-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 2.	Shared
33:23	Reserved.	
34	XD Bit Disable (R/W) See Table 2-3.	Shared
63:35	Reserved.	
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS	
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).		Unique
Register Address: 1D9H, 473	IA32_DEBUGCTL	
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in Table 2-2.		Unique
Register Address: 1DDH, 477	MSR_LER_FROM_LIP	
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 1DEH, 478	MSR_LER_TO_LIP	
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.		Unique
Register Address: 200H, 512	MTRRphysBase0	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Memory Type Range Registers		Unique
Register Address: 201H, 513	MTRRphysMask0	
Memory Type Range Registers		Unique
Register Address: 202H, 514	MTRRphysBase1	
Memory Type Range Registers		Unique
Register Address: 203H, 515	MTRRphysMask1	
Memory Type Range Registers		Unique
Register Address: 204H, 516	MTRRphysBase2	
Memory Type Range Registers		Unique
Register Address: 205H, 517	MTRRphysMask2	
Memory Type Range Registers		Unique
Register Address: 206H, 518	MTRRphysBase3	
Memory Type Range Registers		Unique
Register Address: 207H, 519	MTRRphysMask3	
Memory Type Range Registers		Unique
Register Address: 208H, 520	MTRRphysBase4	
Memory Type Range Registers		Unique
Register Address: 209H, 521	MTRRphysMask4	
Memory Type Range Registers		Unique
Register Address: 20AH, 522	MTRRphysBase5	
Memory Type Range Registers		Unique
Register Address: 20BH, 523	MTRRphysMask5	
Memory Type Range Registers		Unique
Register Address: 20CH, 524	MTRRphysBase6	
Memory Type Range Registers		Unique
Register Address: 20DH, 525	MTRRphysMask6	
Memory Type Range Registers		Unique
Register Address: 20EH, 526	MTRRphysBase7	
Memory Type Range Registers		Unique
Register Address: 20FH, 527	MTRRphysMask7	
Memory Type Range Registers		Unique
Register Address: 250H, 592	MTRRfix64K_00000	
Memory Type Range Registers		Unique
Register Address: 258H, 600	MTRRfix16K_80000	
Memory Type Range Registers		Unique
Register Address: 259H, 601	MTRRfix16K_A0000	
Memory Type Range Registers		Unique

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Register Address: 268H, 616	MTRRfix4K_C0000	
Memory Type Range Registers		Unique
Register Address: 269H, 617	MTRRfix4K_C8000	
Memory Type Range Registers		Unique
Register Address: 26AH, 618	MTRRfix4K_D0000	
Memory Type Range Registers		Unique
Register Address: 26BH, 619	MTRRfix4K_D8000	
Memory Type Range Registers		Unique
Register Address: 26CH, 620	MTRRfix4K_E0000	
Memory Type Range Registers		Unique
Register Address: 26DH, 621	MTRRfix4K_E8000	
Memory Type Range Registers		Unique
Register Address: 26EH, 622	MTRRfix4K_F0000	
Memory Type Range Registers		Unique
Register Address: 26FH, 623	MTRRfix4K_F8000	
Memory Type Range Registers		Unique
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE	
Default Memory Types (R/W) See Table 2-2 and Section 13.11.2.1, "IA32_MTRR_DEF_TYPE MSR."		Unique
Register Address: 400H, 1024	IA32_MCO_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 401H, 1025	IA32_MCO_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 402H, 1026	IA32_MCO_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 404H, 1028	IA32_MC1_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 405H, 1029	IA32_MC1_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."		Unique
Register Address: 406H, 1030	IA32_MC1_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 408H, 1032	IA32_MC2_CTL	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 409H, 1033	IA32_MC2_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 40AH, 1034	IA32_MC2_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 40CH, 1036	MSR_MC4_CTL	
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."		Unique
Register Address: 40DH, 1037	MSR_MC4_STATUS	
See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."		Unique
Register Address: 40EH, 1038	MSR_MC4_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 410H, 1040	IA32_MC3_CTL	
IA32_MC3_CTL	See Section 17.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 411H, 1041	IA32_MC3_STATUS	
IA32_MC3_STATUS	See Section 17.3.2.2, "IA32_MCi_STATUS MSRS."	
Register Address: 412H, 1042	MSR_MC3_ADDR	
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.		Unique
Register Address: 413H, 1043	MSR_MC3_MISC	
Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISC_V flag in the IA32_MCi_STATUS register is set.		Unique
Register Address: 414H, 1044	MSR_MC5_CTL	
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).		Unique
Register Address: 415H, 1045	MSR_MC5_STATUS	
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCi_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.		Unique
Register Address: 416H, 1046	MSR_MC5_ADDR	
Machine Check Error Reporting Register - contains the address of the code or data memory location that produced the machine-check error if the ADDR_V flag in the IA32_MCi_STATUS register is set.		Unique
Register Address: 417H, 1047	MSR_MC5_MISC	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
Machine Check Error Reporting Register - contains additional information describing the machine-check error if the MISCV flag in the IA32_MCI_STATUS register is set.		Unique
Register Address: 480H, 1152	IA32_VMX_BASIC	
Reporting Register of Basic VMX Capabilities (R/O) See Table 2-2 and Appendix A.1, "Basic VMX Information." (If CPUID.01H:ECX[5])		Unique
Register Address: 481H, 1153	IA32_VMX_PINBASED_CTL	
Capability Reporting Register of Pin-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 482H, 1154	IA32_VMX_PROCBASED_CTL	
Capability Reporting Register of Primary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 483H, 1155	IA32_VMX_EXIT_CTL	
Capability Reporting Register of VM-Exit Controls (R/O) See Appendix A.4, "VM-Exit Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 484H, 1156	IA32_VMX_ENTRY_CTL	
Capability Reporting Register of VM-Entry Controls (R/O) See Appendix A.5, "VM-Entry Controls." (If CPUID.01H:ECX[5])		Unique
Register Address: 485H, 1157	IA32_VMX_MISC	
Reporting Register of Miscellaneous VMX Capabilities (R/O) See Appendix A.6, "Miscellaneous Data." (If CPUID.01H:ECX[5])		Unique
Register Address: 486H, 1158	IA32_VMX_CR0_FIXED0	
Capability Reporting Register of CR0 Bits Fixed to 0 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX[5])		Unique
Register Address: 487H, 1159	IA32_VMX_CR0_FIXED1	
Capability Reporting Register of CR0 Bits Fixed to 1 (R/O) See Appendix A.7, "VMX-Fixed Bits in CR0." (If CPUID.01H:ECX[5])		Unique
Register Address: 488H, 1160	IA32_VMX_CR4_FIXED0	
Capability Reporting Register of CR4 Bits Fixed to 0 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX[5])		Unique
Register Address: 489H, 1161	IA32_VMX_CR4_FIXED1	
Capability Reporting Register of CR4 Bits Fixed to 1 (R/O) See Appendix A.8, "VMX-Fixed Bits in CR4." (If CPUID.01H:ECX[5])		Unique
Register Address: 48AH, 1162	IA32_VMX_VMCS_ENUM	
Capability Reporting Register of VMCS Field Enumeration (R/O) See Appendix A.9, "VMCS Enumeration." (If CPUID.01H:ECX[5])		Unique
Register Address: 48BH, 1163	IA32_VMX_PROCBASED_CTL2	
Capability Reporting Register of Secondary Processor-Based VM-Execution Controls (R/O) See Appendix A.3, "VM-Execution Controls." (If CPUID.01H:ECX[5] and IA32_VMX_PROCBASED_CTL2[bit 63])		Unique
Register Address: 600H, 1536	IA32_DS_AREA	

Table 2-66. MSRs in Intel® Core™ Solo, Intel® Core™ Duo Processors, and Dual-Core Intel® Xeon® Processor LV (Contd.)

Register Address: Hex, Decimal	Register Name	
Register Information / Bit Fields	Bit Description	Shared/ Unique
DS Save Area (R/W) See Table 2-2 and Section 21.6.3.4, "Debug Store (DS) Mechanism."		Unique
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.	
63:32	Reserved.	
Register Address: C000_0080H	IA32_EFER	
See Table 2-2.		Unique
10:0	Reserved.	
11	Execute Disable Bit Enable	
63:12	Reserved.	

2.21 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 2.22 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 2-67. MSRs in Pentium M Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	IA32_TIME_STAMP_COUNTER
See Section 19.17, "Time-Stamp Counter," and see Table 2-2.	
Register Address: 17H, 23	IA32_PLATFORM_ID
Platform ID (R) See Table 2-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	
Register Address: 2AH, 42	MSR_EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features. (R) Indicates current processor configuration.	
0	Reserved.
1	Data Error Checking Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
2	Response Error Checking Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
3	MCERR# Drive Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
4	Address Parity Enable (R) 0 = Disabled. Always 0 on the Pentium M processor.
6:5	Reserved.
7	BINIT# Driver Enable (R) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
8	Output Tri-state Enabled (R/O) 1 = Enabled; 0 = Disabled.
9	Execute BIST (R/O) 1 = Enabled; 0 = Disabled.
10	MCERR# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
11	Reserved.
12	BINIT# Observation Enabled (R/O) 1 = Enabled; 0 = Disabled. Always 0 on the Pentium M processor.
13	Reserved.
14	1 MByte Power on Reset Vector (R/O) 1 = 1 MByte; 0 = 4 GBytes. Always 0 on the Pentium M processor.
15	Reserved.
17:16	APIC Cluster ID (R/O) Always 00B on the Pentium M processor.
18	System Bus Frequency (R/O) 0 = 100 MHz. 1 = Reserved. Always 0 on the Pentium M processor.
19	Reserved.
21:20	Symmetric Arbitration ID (R/O) Always 00B on the Pentium M processor.
26:22	Clock Frequency Ratio (R/O)
Register Address: 40H, 64	MSR_LASTBRANCH_0

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Last Branch Record 0 (R/W) One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: ▪ Last Branch Record Stack TOS at 1C9H. ▪ Section 19.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 41H, 65	MSR_LASTBRANCH_1
Last Branch Record 1 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 42H, 66	MSR_LASTBRANCH_2
Last Branch Record 2 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 43H, 67	MSR_LASTBRANCH_3
Last Branch Record 3 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 44H, 68	MSR_LASTBRANCH_4
Last Branch Record 4 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 45H, 69	MSR_LASTBRANCH_5
Last Branch Record 5 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 46H, 70	MSR_LASTBRANCH_6
Last Branch Record 6 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 47H, 71	MSR_LASTBRANCH_7
Last Branch Record 7 (R/W) See description of MSR_LASTBRANCH_0.	
Register Address: 119H, 281	MSR_BBL_CR_CTL
Control Register Used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.	
63:0	Reserved.
Register Address: 11EH, 281	MSR_BBL_CR_CTL3
Control Register 3 Used to configure the L2 Cache.	
0	L2 Hardware Enabled (R/O) 1 = If the L2 is hardware-enabled. 0 = Indicates if the L2 is hardware-disabled.
4:1	Reserved.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
5	ECC Check Enable (R/O) This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default). 1 = Enabled. For the Pentium M processor, ECC checking on the cache data bus is always enabled.
7:6	Reserved.
8	L2 Enabled (R/W) 1 = L2 cache has been initialized. 0 = Disabled (default). Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
22:9	Reserved.
23	L2 Not Present (R/O) 0 = L2 Present. 1 = L2 Not Present.
63:24	Reserved.
Register Address: 179H, 377	IA32_MCG_CAP
Read-only register that provides information about the machine-check architecture of the processor.	
7:0	Count (R/O) Indicates the number of hardware unit error reporting banks available in the processor.
8	IA32_MCG_CTL Present (R/O) 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
63:9	Reserved.
Register Address: 17AH, 378	IA32_MCG_STATUS
Global Machine Check Status	
0	RIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
1	EIPV When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
2	MCIP When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
63:3	Reserved.
Register Address: 198H, 408	IA32_PERF_STATUS

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
See Table 2-2.	
Register Address: 199H, 409	IA32_PERF_CTL
See Table 2-2.	
Register Address: 19AH, 410	IA32_CLOCK_MODULATION
Clock Modulation (R/W). See Table 2-2 and Section 16.8.3, "Software Controlled Clock Modulation."	
Register Address: 19BH, 411	IA32_THERM_INTERRUPT
Thermal Interrupt Control (R/W) See Table 2-2 and Section 16.8.2, "Thermal Monitor."	
Register Address: 19CH, 412	IA32_THERM_STATUS
Thermal Monitor Status (R/W) See Table 2-2 and Section 16.8.2, "Thermal Monitor."	
Register Address: 19DH, 413	MSR_THERM2_CTL
Thermal Monitor 2 Control	
15:0	Reserved.
16	TM_SELECT (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
63:16	Reserved.
Register Address: 1A0H, 416	IA32_MISC_ENABLE
Enable Miscellaneous Processor Features (R/W) Allows a variety of processor functions to be enabled and disabled.	
2:0	Reserved.
3	Automatic Thermal Control Circuit Enable (R/W) 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
6:4	Reserved.
7	Performance Monitoring Available (R) 1 = Performance monitoring enabled. 0 = Performance monitoring disabled.

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
9:8	Reserved.
10	FERR# Multiplexing Enable (R/W) 1 = FERR# asserted by the processor to indicate a pending break event within the processor. 0 = Indicates compatible FERR# signaling behavior. This bit must be set to 1 to support XAPIC interrupt model usage.
	Branch Trace Storage Unavailable (R/O) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported
12	Processor Event Based Sampling Unavailable (R/O) 1 = Processor does not support processor event based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
15:13	Reserved.
16	Enhanced Intel SpeedStep Technology Enable (R/W) 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
22:17	Reserved.
23	xTPR Message Disable (R/W) When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
63:24	Reserved.
Register Address: 1C9H, 457	MSR_LASTBRANCH_TOS
Last Branch Record Stack TOS (R/W) Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: - MSR_LASTBRANCH_0_FROM_IP (at 40H). - Section 19.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 1D9H, 473	MSR_DEBUGCTLB
Debug Control (R/W) Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 19.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."	
Register Address: 1DDH, 477	MSR_LER_TO_LIP
Last Exception Record To Linear IP (R) This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 19.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 19.16.2, "Last Branch and Last Exception MSRs."	
Register Address: 1DEH, 478	MSR_LER_FROM_LIP

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Last Exception Record From Linear IP (R) Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 19.15, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)," and Section 19.16.2, "Last Branch and Last Exception MSRs."	
Register Address: 2FFH, 767	IA32_MTRR_DEF_TYPE
Default Memory Types (R/W) Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 13.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	
Register Address: 400H, 1024	IA32_MCO_CTL
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 401H, 1025	IA32_MCO_STATUS
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 402H, 1026	IA32_MCO_ADDR
See Section 14.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MCO_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MCO_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 404H, 1028	IA32_MC1_CTL
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 405H, 1029	IA32_MC1_STATUS
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 406H, 1030	IA32_MC1_ADDR
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 408H, 1032	IA32_MC2_CTL
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 409H, 1033	IA32_MC2_STATUS
See Chapter 17.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 40AH, 1034	IA32_MC2_ADDR
See Section 17.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDR_V flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 40CH, 1036	MSR_MC4_CTL
See Section 17.3.2.1, "IA32_MCi_CTL MSRs."	
Register Address: 40DH, 1037	MSR_MC4_STATUS
See Section 17.3.2.2, "IA32_MCi_STATUS MSRs."	
Register Address: 40EH, 1038	MSR_MC4_ADDR

Table 2-67. MSRs in Pentium M Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 410H, 1040	MSR_MC3_CTL
See Section 17.3.2.1, "IA32_MCI_CTL MSRs."	
Register Address: 411H, 1041	MSR_MC3_STATUS
See Section 17.3.2.2, "IA32_MCI_STATUS MSRs."	
Register Address: 412H, 1042	MSR_MC3_ADDR
See Section 17.3.2.3, "IA32_MCI_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDR_V flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.	
Register Address: 600H, 1536	IA32_DS_AREA
DS Save Area (R/W) See Table 2-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 21.6.3.4, "Debug Store (DS) Mechanism."	
31:0	DS Buffer Management Area Linear address of the first byte of the DS buffer management area.
63:32	Reserved.

2.22 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 2-2 for a list of the architectural MSRs.

Table 2-68. MSRs in the P6 Family Processors

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 0H, 0	P5_MC_ADDR
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 2.23, "MSRs in Pentium Processors."	
Register Address: 10H, 16	TSC
See Section 19.17, "Time-Stamp Counter."	
Register Address: 17H, 23	IA32_PLATFORM_ID
Platform ID (R) The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	
49:0	Reserved.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
52:50	Platform Id (R) Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
56:53	L2 Cache Latency Read.
59:57	Reserved.
60	Clock Frequency Ratio Read.
63:61	Reserved.
Register Address: 1BH, 27	APIC_BASE
Section 12.4.4, "Local APIC Status and Location."	
7:0	Reserved.
8	Boot Strap Processor Indicator Bit 1 = BSP
10:9	Reserved.
11	APIC Global Enable Bit - Permanent till reset 1 = Enabled. 0 = Disabled.
31:12	APIC Base Address.
63:32	Reserved.
Register Address: 2AH, 42	EBL_CR_POWERON
Processor Hard Power-On Configuration (R/W) Enables and disables processor features, and (R) indicates current processor configuration.	
0	Reserved ¹
1	Data Error Checking Enable (R/W) 1 = Enabled. 0 = Disabled.
2	Response Error Checking Enable FRCERR Observation Enable (R/W) 1 = Enabled. 0 = Disabled.
3	AERR# Drive Enable (R/W) 1 = Enabled. 0 = Disabled.
4	BERR# Enable for Initiator Bus Requests (R/W) 1 = Enabled. 0 = Disabled.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
5	Reserved.
6	BERR# Driver Enable for Initiator Internal Errors (R/W) 1 = Enabled. 0 = Disabled.
7	BINIT# Driver Enable (R/W) 1 = Enabled. 0 = Disabled.
8	Output Tri-state Enabled (R) 1 = Enabled. 0 = Disabled.
9	Execute BIST (R) 1 = Enabled. 0 = Disabled.
10	AERR# Observation Enabled (R) 1 = Enabled. 0 = Disabled.
11	Reserved.
12	BINIT# Observation Enabled (R) 1 = Enabled. 0 = Disabled.
13	In Order Queue Depth (R) 1 = 1. 0 = 8.
14	1-MByte Power on Reset Vector (R) 1 = 1MByte. 0 = 4GBytes.
15	FRC Mode Enable (R) 1 = Enabled. 0 = Disabled.
17:16	APIC Cluster ID (R)
19:18	System Bus Frequency (R) 00 = 66MHz. 10 = 100Mhz. 01 = 133MHz. 11 = Reserved.
21:20	Symmetric Arbitration ID (R)
25:22	Clock Frequency Ratio (R)
26	Low Power Mode Enable (R/W)
27	Clock Frequency Ratio
63:28	Reserved. ¹
Register Address: 33H, 51	MSR_TEST_CTRL

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Test Control Register	
29:0	Reserved.
30	Streaming Buffer Disable
31	Disable LOCK# Assertion for split locked access.
Register Address: 79H, 121	BIOS_UPDT_TRIG
BIOS Update Trigger Register.	
Register Address: 88H, 136	BBL_CR_D0[63:0]
Chunk 0 data register D[63:0]: used to write to and read from the L2.	
Register Address: 89H, 137	BBL_CR_D1
Chunk 1 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8AH, 138	BBL_CR_D2
Chunk 2 data register D[63:0]: used to write to and read from the L2.	
Register Address: 8BH, 139	BIOS_SIGN/BBL_CR_D3
BIOS Update Signature Register or Chunk 3 data register D[63:0]. Used to write to and read from the L2 depending on the usage model.	
Register Address: C1H, 193	PerfCtr0 (PERFCTR0)
Performance Counter Register See Table 2-2.	
Register Address: C2H, 194	PerfCtr1 (PERFCTR1)
Performance Counter Register See Table 2-2.	
Register Address: FEH, 254	MTRRcap
Memory Type Range Registers	
Register Address: 116H, 278	BBL_CR_ADDR
Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.	
2:0	Reserved; set to 0.
31:3	Address bits [35:3].
63:32	Reserved.
Register Address: 118H, 280	BBL_CR_DECC
Data ECC register D[7:0]: used to write ECC and read ECC to/from L2.	
Register Address: 119H, 281	BBL_CR_CTL
Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response.	

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
4:0	L2 Command: 01100 = Data Read w/ LRU update (RLU). 01110 = Tag Read w/ Data Read (TRR). 01111 = Tag Inquire (TI). 00010 = L2 Control Register Read (CR). 00011 = L2 Control Register Write (CW). 010 + MESI encode = Tag Write w/ Data Read (TWR). 111 + MESI encode = Tag Write w/ Data Write (TWW). 100 + MESI encode = Tag Write (TW).
6:5	
7	State to L2
9:8	Reserved.
11:10	Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11 Way to L2
13:12	Modified - 11, Exclusive - 10, Shared - 01, Invalid - 00 Way from L2
15:14	State from L2.
16	Reserved.
17	L2 Hit.
18	Reserved.
20:19	User supplied ECC.
21	Processor number: ² Disable = 1. Enable = 0. Reserved.
63:22	Reserved.
Register Address: 11AH, 282	BBL_CR_TRIG
Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.	
Register Address: 11BH, 283	BBL_CR_BUSY
Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY.	
Register Address: 11EH, 286	BBL_CR_CTL3
Control register 3: used to configure the L2 Cache.	
0	L2 Configured (read/write).
4:1	L2 Cache Latency (read/write).
5	ECC Check Enable (read/write).
6	Address Parity Check Enable (read/write).
7	CRTN Parity Check Enable (read/write).
8	L2 Enabled (read/write).

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
10:9	L2 Associativity (read only): 00 = Direct Mapped. 01 = 2 Way. 10 = 4 Way. 11 = Reserved.
12:11	Number of L2 banks (read only).
17:13	Cache size per bank (read/write): 00001 = 256 KBytes. 00010 = 512 KBytes. 00100 = 1 MByte. 01000 = 2 MBytes. 10000 = 4 MBytes.
18	Cache State error checking enable (read/write).
19	Reserved.
22:20	L2 Physical Address Range support: 111 = 64 GBytes. 110 = 32 GBytes. 101 = 16 GBytes. 100 = 8 GBytes. 011 = 4 GBytes. 010 = 2 GBytes. 001 = 1 GByte. 000 = 512 MBytes.
23	L2 Hardware Disable (read only).
24	Reserved.
25	Cache bus fraction (read only).
63:26	Reserved.
Register Address: 174H, 372	SYSENTER_CS_MSR
CS register target for CPL 0 code	
Register Address: 175H, 373	SYSENTER_ESP_MSR
Stack pointer for CPL 0 stack	
Register Address: 176H, 374	SYSENTER_EIP_MSR
CPL 0 code entry point	
Register Address: 179H, 377	MCG_CAP
Machine Check Global Control Register	
Register Address: 17AH, 378	MCG_STATUS
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.	
Register Address: 17BH, 379	MCG_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 186H, 390	PerfEvtSel0 (EVNTSEL0)

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Performance Event Select Register 0 (R/W)	
7:0	Event Select Refer to Performance Counter section for a list of event encodings.
15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
17	OS Controls the counting of events at Privilege level of 0.
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BPO pin.
20	INT Enables the signaling of counter overflow via input to APIC: 1 = Enable. 0 = Disable.
22	ENABLE Enables the counting of performance events in both counters: 1 = Enable. 0 = Disable.
23	INV Inverts the result of the CMASK condition: 1 = Inverted. 0 = Non-Inverted.
31:24	CMASK (Counter Mask)
Register Address: 187H, 391	PerfEvtSel1 (EVNTSEL1)
Performance Event Select for Counter 1 (R/W)	
7:0	Event Select Refer to Performance Counter section for a list of event encodings.
15:8	UMASK (Unit Mask) Unit mask register set to 0 to enable all count options.
16	USER Controls the counting of events at Privilege levels of 1, 2, and 3.
17	OS Controls the counting of events at Privilege level of 0.

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
18	E Occurrence/Duration Mode Select: 1 = Occurrence. 0 = Duration.
19	PC Enabled the signaling of performance counter overflow via BP0 pin.
20	INT Enables the signaling of counter overflow via input to APIC. 1 = Enable. 0 = Disable.
23	INV Inverts the result of the CMASK condition. 1 = Inverted. 0 = Non-Inverted.
31:24	CMASK (Counter Mask)
Register Address: 1D9H, 473	DEBUGCTLMR
Enables last branch, interrupt, and exception recording; taken branch breakpoints; the breakpoint reporting pins; and trace messages. This register can be written to using the WRMSR instruction, when operating at privilege level 0 or when in real-address mode.	
0	Enable/Disable Last Branch Records
1	Branch Trap Flag
2	Performance Monitoring/Break Point Pins
3	Performance Monitoring/Break Point Pins
4	Performance Monitoring/Break Point Pins
5	Performance Monitoring/Break Point Pins
6	Enable/Disable Execution Trace Messages
31:7	Reserved.
Register Address: 1DBH, 475	LASTBRANCHFROMIP
32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.	
Register Address: 1DCH, 476	LASTBRANCHTOIP
32-bit register for recording the instruction pointers for the last branch, interrupt, or exception that the processor took prior to a debug exception being generated.	
Register Address: 1DDH, 477	LASTINTFROMIP
Last INT from IP	
Register Address: 1DEH, 478	LASTINTTOIP
Last INT to IP	
Register Address: 200H, 512	MTRRphysBase0
Memory Type Range Registers	
Register Address: 201H, 513	MTRRphysMask0
Memory Type Range Registers	

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 202H, 514	MTRRphysBase1
Memory Type Range Registers	
Register Address: 203H, 515	MTRRphysMask1
Memory Type Range Registers	
Register Address: 204H, 516	MTRRphysBase2
Memory Type Range Registers	
Register Address: 205H, 517	MTRRphysMask2
Memory Type Range Registers	
Register Address: 206H, 518	MTRRphysBase3
Memory Type Range Registers	
Register Address: 207H, 519	MTRRphysMask3
Memory Type Range Registers	
Register Address: 208H, 520	MTRRphysBase4
Memory Type Range Registers	
Register Address: 209H, 521	MTRRphysMask4
Memory Type Range Registers	
Register Address: 20AH, 522	MTRRphysBase5
Memory Type Range Registers	
Register Address: 20BH, 523	MTRRphysMask5
Memory Type Range Registers	
Register Address: 20CH, 524	MTRRphysBase6
Memory Type Range Registers	
Register Address: 20DH, 525	MTRRphysMask6
Memory Type Range Registers	
Register Address: 20EH, 526	MTRRphysBase7
Memory Type Range Registers	
Register Address: 20FH, 527	MTRRphysMask7
Memory Type Range Registers	
Register Address: 250H, 592	MTRRfix64K_00000
Memory Type Range Registers	
Register Address: 258H, 600	MTRRfix16K_80000
Memory Type Range Registers	
Register Address: 259H, 601	MTRRfix16K_A0000
Memory Type Range Registers	
Register Address: 268H, 616	MTRRfix4K_C0000
Memory Type Range Registers	
Register Address: 269H, 617	MTRRfix4K_C8000

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Memory Type Range Registers	
Register Address: 26AH, 618	MTRRfix4K_D0000
Memory Type Range Registers	
Register Address: 26BH, 619	MTRRfix4K_D8000
Memory Type Range Registers	
Register Address: 26CH, 620	MTRRfix4K_E0000
Memory Type Range Registers	
Register Address: 26DH, 621	MTRRfix4K_E8000
Memory Type Range Registers	
Register Address: 26EH, 622	MTRRfix4K_F0000
Memory Type Range Registers	
Register Address: 26FH, 623	MTRRfix4K_F8000
Memory Type Range Registers	
Register Address: 2FFH, 767	MTRRdefType
Memory Type Range Registers	
2:0	Default memory type
10	Fixed MTRR enable
11	MTRR Enable
Register Address: 400H, 1024	MCO_CTL
Machine Check Error Reporting Register - controls signaling of #MC for errors produced by a particular hardware unit (or group of hardware units).	
Register Address: 401H, 1025	MCO_STATUS
Machine Check Error Reporting Register - contains information related to a machine-check error if its VAL (valid) flag is set. Software is responsible for clearing IA32_MCI_STATUS MSRs by explicitly writing 0s to them; writing 1s to them causes a general-protection exception.	
15:0	MC_STATUS_MCACOD
31:16	MC_STATUS_MSCOD
57	MC_STATUS_DAM
58	MC_STATUS_ADDRV
59	MC_STATUS_MISCV
60	MC_STATUS_EN. (Note: For MCO_STATUS only, this bit is hardcoded to 1.)
61	MC_STATUS_UC
62	MC_STATUS_O
63	MC_STATUS_V
Register Address: 402H, 1026	MCO_ADDR
Register Address: 403H, 1027	MCO_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 404H, 1028	MC1_CTL

Table 2-68. MSRs in the P6 Family Processors (Contd.)

Register Address: Hex, Decimal	Register Name
Register Information / Bit Fields	Bit Description
Register Address: 405H, 1029	MC1_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 406H, 1030	MC1_ADDR
Register Address: 407H, 1031	MC1_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 408H, 1032	MC2_CTL
Register Address: 409H, 1033	MC2_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 40AH, 1034	MC2_ADDR
Register Address: 40BH, 1035	MC2_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 40CH, 1036	MC4_CTL
Register Address: 40DH, 1037	MC4_STATUS
Bit definitions same as MC0_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.	
Register Address: 40EH, 1038	MC4_ADDR
Defined in MCA architecture but not implemented in P6 Family processors.	
Register Address: 40FH, 1039	MC4_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
Register Address: 410H, 1040	MC3_CTL
Register Address: 411H, 1041	MC3_STATUS
Bit definitions same as MC0_STATUS.	
Register Address: 412H, 1042	MC3_ADDR
Register Address: 413H, 1043	MC3_MISC
Defined in MCA architecture but not implemented in the P6 family processors.	
NOTES 1. Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors. 2. The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to "1". Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset. 3. The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.	

2.23 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 2.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 2-69. MSRs in the Pentium Processor

Register Address: Hex, Decimal	Register Name
Register Information	
Register Address: 0H, 0	P5_MC_ADDR
See Section 17.11.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 1H, 1	P5_MC_TYPE
See Section 17.11.2, "Pentium Processor Machine-Check Exception Handling."	
Register Address: 10H, 16	TSC
See Section 19.17, "Time-Stamp Counter."	
Register Address: 11H, 17	CESR
See Section 21.6.9.1, "Control and Event Select Register (CESR)."	
Register Address: 12H, 18	CTR0
Section 21.6.9.3, "Events Counted."	
Register Address: 13H, 19	CTR1
Section 21.6.9.3, "Events Counted."	

