Intel® Architecture Instruction Set Extensions and Future Features

Programming Reference

June 2026

319433-062

Revision History

Revision	Description	Date
-053	• Chapter 1: Minor updates/corrections. Updated CPUID Leaf 06H, EAX bit 18, to align with text used in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, Chapter 15. Added the field name and definition of CPUID Leaf 06H, EAX bit 22. Updated CPUID Leaf 07H, Subleaf 2, to add enumeration for MONITOR_MITG_NO. • Chapter 2: Minor updates only. • Chapter 16: Extensive updates to provide additional information and clarity on this feature.	June 2024
-054	• Removed entries from the Revision History table that were older than two years. • Chapter 1: Updated Table 1-1, "CPUID Signature Values of DisplayFamily_DisplayModel," to add values for future processors based on Diamond Rapids microarchitecture. Updated Table 1-2, "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors." Updated the CPUID instruction with corrections as necessary; added bits enumerating MOVRS, AMX-FP8, MSR_IMM, AMX-TRANSPOSE, AMX-TF32, AMX-AVX512, AMX-MOVRS, and Architectural PEBS; and added CPUID Leaf 23H, Subleaves 4 and 5. Added FP8 format information. • Chapter 2: Added the MOVRS and PREFETCHRST2instructions. Added the EVEX forms of the VSM4KEY4 and VSM4RNDS4 instructions. Updated the WRMSRNS instruction with the immediate form. Added the RDMSR instruction and updated it with the immediate form. • Chapter 3: Added the following Intel AMX instructions: T2RPNTLVW[Z0,Z1][,T1], T2RPNTLVW[Z0,Z1]RS[T1], TCONJTCMMIMFP16PS, TCONJTFP16,TCVTROWD2PS, TCVTROWPS2PBF16[H,L], TCVTROWPS2PH[H,L],TDP[B,H,BH,HB]F8PS, TILELOADDRS[T1], TILEMOVROW,TMMULTF32PS, TTCMM[IM,RL]FP16PS, TTDPBF16PS,TTDPFP16PS, TTMMULTF32PS, TTRANSPOSED. Removed the TDPFP16PS instruction; this can be found in the Intel 64 and IA-32 Architectures Software Developer's Manuals. • Added new Chapter 11, "Architectural PEBS." • Added new Chapter 12, "MOVRS Instructions." • Removal of chapters: Removed previous Chapter 4, "UC-Lock Disable." Removed previous Chapter 6, "Linear Address Masking (LAM)." Removed previous Chapter 7, "Code Prefetch Instruction Updates." Removed previous Chapter 8, "Next Generation Performance Monitoring Unit (PMU)." Removed previous Chapter 9, "Linear Address Space Separation (LASS)." Removed previous Chapter 12, "Flexible UIRET." Removed previous Chapter 15, "VMX Support for the IA32_SPEC_CTRL MSR." The information from these chapters can be found in the Intel 64 and IA-32 Architectures Software Developer's Manuals.	October 2024
-055	• Removed an incorrect line in the revision history to ensure the document contents are accurately represented. All change bars in the document remain to reflect Revision 054 edits. • Removed an inaccurate value in Table 1-1, "CPUID Signature Values of DisplayFamily_DisplayModel."	October 2024

Revision	Description	Date
-056	• Chapter 1: Added Panther Lake to Architectural PEBS in Table 1-2, "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors." Added footnote for CPUID Leaf 0AH, ECX and EDX; and CPUID Leaf 23H (ECX=1), EDX in Table 1-3, "Information Returned by CPUID Instruction" for fixed-function counters. Added #Guest Physical Address Bits to CPUID Leaf 80000008H, EAX[23:16]. • Chapter 2: Updated descriptions for AADD, AAND, AOR, and AXOR. Added reference footnote for EVEX opcode version of RDMSR and WRMSRNS. • Chapter 3: Corrected typo for CPUID Feature Flag of "TTDPFP16PS—Dot Product and Transpose of FP16 Tiles Accumulated into Packed Single Precision Tile." Added helper function to zero out of range columns in Section 3.4, "Helper Functions" and to T2RPNTLVW, TDP, and TILELOADDRS. Updated name of TCVTROWPS2PBF16 to TCVTROWPS2BF16. • Chapter 4: Added new Section 4.2, "Memory Regions," Section 4.3, "Region Aware Memory Bandwidth Monitoring (MBM)," and Section 4.4, "Region Aware Memory Bandwidth Allocation (MBA)." • Added new Chapter 13, "VMX Enhancements."	December 2024
-057	• Chapter 1: Added Asymmetric RDT bits to CPUID.(EAX=07H,ECX=01H).ECX[bits 1:0] and CPUID leaves 27H and 28H. Added 256BITSGX to CPUID.(EAX=12H,ECX=00H).EAX[bit 12]. Updated CPUID.(EAX=1EH,ECX=00H). Updated CPUID.(EAX=24H,ECX=00H).EBX[bits 18:16] for vector length support. • Chapter 2: Removed footnote references to verify vector options in the opcode tables of instructions MOVRS, VSM4KEY4, and VSM4RNDS4. Updated pseudocode for instruction PBNDKB. Clarified the CPUID Feature Flag adding AND where multiple flags are required for the opcode tables of VSM4KEY4 and VSM4RNDS4. • Chapter 3: Clarified the CPUID Feature Flag adding AND where multiple flags are required for the opcode tables of T2RPMTLVW[Z0,Z1]RS[T1], TCONJTCMMIMFP16PS, TCONJTFP16, TCVTROWD2PS, TCVTROWPS2BF16, TCVTROWPS2PH, TDP[B,H,BH,HB]F8PS, TILEMOVROW, TTCM[IM,RL]FP16PS, TTDPBF16PS, TTDPFP16PS, and TTMMULTF32PS. • Chapter 4: Added Section 4.5, "Use of RDT Tags by Other Features" and Section 4.6, "Asymmetrical Enumeration of Intel® RDT Features." • Chapter 5: Retitled Section 5.6 to "When Multiple RAO Memory Operations Are Combined" and clarified section content. Retitled Section 5.8.2 to "Event Handler." • Chapter 11: Updated Section 11.4.4, "XSAVE-Enabled Registers Group." • Added new Chapter 14, "Intel® Software Guard Extensions Feature Updates."	March 2025
-058	• Chapter 1: Updated Table 1-1 for Arrow Lake H and Arrow Lake S. Updated Table 1-2, "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors," removing Panther Lake and Diamond Rapids from NMI-Source Reporting and clarifying the introduction of Arrow Lake for various features. Updated CPUID.06H:EAX[24] to reserved. Corrected CPUID.(EAX=07H, ECX=00H):EDX[bit 04] from Fast Short REP MOV to Fast Short REP MOVSB. Added CPUID.(EAX=07H,ECX=01H):EDX.SLSM[bit 24] for Static Lockstep Mode. Updated bits 05 and 06 in CPUID.(EAX=12H,ECX=00H): EAX to reserved. Updated CPUID.(EAX=23H,ECX=04H):EBX[bits 55:49] for the XER field. • Chapter 2: Updated CPUID feature flag of EVEX forms of instructions VSM4KEY4, VSM4RNDS4, and VMOVRS.	June 2025

Revision	Description	Date
-058 (continued)	• Chapter 3: Updated CPUID feature flag of EVEX forms of instructions TCVTROWD2PS, TCVTROWPS2BF16[H,L], TCVTROWPS2PH[H,L], and TILEMOVROW. Removed zero_out_of_range_columns from Section 3.4, “Helper Functions,” and from instructions T2RPNTLVW[Z0,Z1][,T1] and TILELOADDRS. • Chapter 11. Updated Table 11-1, "Architectural PEBS Configuration Locations" in Section 11.2.2. Added bit 51 for EGPR under the XER field description and to the Bit Information/Bit Fields row for 55:49 of Table 11-5 in Section 11.3.3. Revised first paragraph of Section 11.4.4, “XSAVE-Enabled Registers Group” for clarity. Added a row for EGPR in Table 11-9 in Section 11.4.4. • Chapter 14: Removed ENCLV from Section 14.1. Updated Table 14-3 in Section 14.4, “Memory Accesses.” Updated Table 14-12 in Section 14.6.2, “Information and Error Codes.” Updated Table 14-14 and Table 14-15 in Section 14.5, “Intel® SGX Instructions Concurrency Tables.” Removed SGX_CONFLICT VM Exit Qualification column from base concurrency restrictions table, removed ETRACKC from the column heading of additional concurrency restrictions table, and updated pseudocode in Operation section for leaf functions: EADD, ECREATE, EWB, EGETKEY, EREPORT, and EREPORT2. Removed SGX_CONFLICT VM Exit Qualification column from base concurrency restrictions table and removed ETRACKC from the column heading of additional concurrency restrictions table for leaf functions: EEXTEND and EINIT. For leaf function ELDB/ELDU/ELDBC/ELDUC, removed ELDBC/ELDUC, updated tables for base concurrency restrictions and additional concurrency restrictions, and updated pseudocode in Operation section. • Added new Chapter 15, “RDPMC User Disable.” • Added new Chapter 16, “Performance Monitoring Unit Enhancements.”	June 2025
-059	• Removed references to AMX-TRANSPOSE. • Chapter 1: Updated Table 1-1, "CPUID Signature Values of DisplayFamily_DisplayModel". Updated Table 1-2, "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors" for PREFETCHIT0/1, Intel® Total Storage Encryption, URDMSR and UWRMSR, and the removal of AMX-TRANSPOSE. Removed AMX-TRANSPOSE from CPUID.1EH.01H:EAX[bit 05]. • Chapter 2: Updated description for instructions URDMSR and UWRMSR. • Chapter 3: Removed instructions T2RPNTLVW[Z0,Z1][,T1], T2RPNTLVW[Z0,Z1]RS[,T1], TCONJTCMMIMFP16PS, TCONJTFP16, TTCMM[IM,RL]FP16PS, TTDPBF16PS, TTDPFP16PS, TTMMULTF32PS, and TTRANSPOSED. • Chapter 13: Added content for support of virtualization of instructions URDMSR, UWRMSR, RDMSR, and WRMSRNS. Added Section 13.2, “Changes to VMX Non-Root Operation” and re-numbered subsequent sections. • Chapter 16: Added content for E-Core OMR performance monitoring throughout chapter. • Added new Chapter 17, "Machine Check Error Codes".	September 2025

Revision	Description	Date
-060	• Chapter 1: Updated Table 1-1, "CPUID Signature Values of DisplayFamily_DisplayModel," editing description of processor client. Updating Table 1-2, "Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors," adding new processor intercept for AVX10.1, AVX10.2, APX, MOVRS, and SM4 (EVEX). Updated Section 1.7, "FP8 Format", and its subsections for E5M2 and E4M3 FP8 formats. • Chapter 3: Updated Section 3.4, "Helper Functions," with E5M2 and E4M3 FP8 formats. Updated instruction TDP[B,H,BH,HB]F8PS—Dot Product of FP8 Tiles Accumulated into Packed Single Precision Tile in Section 3.7 with E5M2 and E4M3 FP8 formats. • Chapter 16: Updated bit 10 to reserved in Table 16-6, "PEBS Memory Aux Info Field Layout for Arctic Wolf Microarchitecture." Updated event name MEM_LOAD_L2_MISS_RETIRED.* in Table 16-8, "Event Counter Restrictions for Panther Cove and Coyote Cove Microarchitectures."	November 2025
-061	• Chapter 1: Updated Introduction for User-Timer Events and Interrupts in Table 1-2. In Section 1.4, references to the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1 and Volume 2A, were added; and the pseudocode in the Operation section was updated. In Table 1-3, field names were included and the following CPUID leaves and subleaves were updated: CPUID.07H.00H:EDX, CPUID.07H.01H:EDX, CPUID.07H.02H:EDX, CPUID.0AH.00H:EDX, CPUID.0DH.00H:EAX, CPUID.0DH.01H:ECX, CPUID.18H.00H, CPUID.1EH.01H, CPUID.23H.00H:EAX, CPUID23H.04H:EBX, CPUID.80000001H:EAX, and CPUID.80000008H:EAX. In Table 1-7, updated description for C2H. • Chapter 2: Removed instructions PBNDKB and PCONFIG. The information for these instructions can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manuals. • Chapter 4: Removed Section 4.6, "Asymmetrical Enumeration of Intel® RDT Features." The information from this section can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3. • Chapter 7: Updated Section 7.4.1 to specify that a virtual timer interrupt is pended (without arming the guest timer) if IA32_TSC_DEADLINE is written with a non-zero value that does not exceed the virtual TSC. • Chapter 12: The EUPDATESVN leaf function was updated for the move of the SGX attestation root of trust to S3M and interactions between EUPDATESVN and SEAMOPS. CR_REPORT2_KEY was replaced with CR_REPORT_MAC_KEY in Table 7-13, "List of Internal CREG." • Chapter 14: Corrected the supported UMasks for OMR events in Table 9-9. • Chapter 16: Updated error codes reported in IA32_MC4_STATUS (Table 11-1) and IA32_MC11_STATUS (Table 11-2). • Inserted new Chapter 15, "PerfMon Virtualization with PerfMon Masking," and renumbered subsequent chapters. • Removal of chapters: Removed previous Chapter 6, "Total Storage Encryption in Intel Architecture." Removed previous Chapter 11, "Architectural PEBS." The information from these chapters can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual.	March 2026

Revision	Description	Date
-062	• Chapter 1: Table 1-1 was updated to reflect current status of listed processors. Table 1-2 was revised removing User-Timer Events and Interrupts and AMX-TF32; and adding RDPMC User Disable. In Table 1-3, CPUID.07H.01H:EDX[13] and CPUID.1EH.01H:EAX[6] were updated to reserved. • Chapter 2: Instructions RDMSR, WRMSRNS, URDMSR, and UWRMSR were removed. Information for these instructions can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2. • Chapter 3: In Section 3.6 and Section 3.7, TILEMOVEROW and TCVTROW* instructions were simplified by eliminating #GP conditions on immediate or register operands and dropping row chunk notion. The Description for these instructions were also updated. In Section 3.7, The TCMMIMFP16PS/TCMMRLFP16PS and TMMULTF32PS instructions were removed. Information for TCMMIMFP16PS/TCMMRLFP16PS can be found in Volume 2B of the Intel® 64 and IA-32 Architectures Software Developer's Manual. • Removal of chapters: Removed previous Chapter 6, "User-Timer Events and Interrupts." Removed previous Chapter 7, "APIC-Timer Virtualization." Removed previous Chapter 8, "Processor Trace Trigger Tracing." Removed previous Chapter 9, "Monitorless MWAIT." Removed previous Chapter 11, "VMX Enhancements." With the exception of Chapter 6, the information from these chapters can be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual.	June 2026

REVISION HISTORY

CHAPTER 1
FUTURE INTEL® ARCHITECTURE INSTRUCTION EXTENSIONS AND FEATURES

CHAPTER 2
INSTRUCTION SET REFERENCE, A-Z

CHAPTER 3
INTEL® AMX INSTRUCTION SET REFERENCE, A-Z

CHAPTER 4
INTEL® RESOURCE DIRECTOR TECHNOLOGY FEATURE UPDATES

CHAPTER 5
REMOTE ATOMIC OPERATIONS IN INTEL ARCHITECTURE

CHAPTER 6
MOVRS INSTRUCTIONS

CHAPTER 7
INTEL® SOFTWARE GUARD EXTENSIONS FEATURE UPDATES

CHAPTER 8
RDPMC USER DISABLE

CHAPTER 9
PERFORMANCE MONITORING UNIT ENHANCEMENTS

CHAPTER 10
PERFMON VIRTUALIZATION WITH PERFMON MASKING

CHAPTER 11
MACHINE CHECK ERROR CODES

TABLES

PAGE

FIGURES

PAGE

CHAPTER 1
FUTURE INTEL® ARCHITECTURE INSTRUCTION EXTENSIONS AND FEATURES

1.1 ABOUT THIS DOCUMENT

This document describes the software programming interfaces of Intel® architecture instruction extensions and features which may be included in future Intel processor generations. Intel does not guarantee the availability of these interfaces and features in any future product.

The instruction set extensions cover a diverse range of application domains and programming usages. The 512-bit SIMD vector SIMD extensions, referred to as Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instructions, deliver comprehensive set of functionality and higher performance than Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Vector Extensions 2 (Intel® AVX2) instructions. Intel AVX, Intel AVX2, and many Intel AVX-512 instructions are covered in the Intel® 64 and IA-32 Architectures Software Developer's Manual. The reader can refer to them for basic and more background information related to various features referenced in this document.

The base of the 512-bit SIMD instruction extensions are referred to as Intel AVX-512 Foundation instructions. They include extensions of the Intel AVX and Intel AVX2 family of SIMD instructions but are encoded using a new encoding scheme with support for 512-bit vector registers, up to 32 vector registers in 64-bit mode, and conditional processing using opmask registers.

Chapter 2 is an instruction set reference, providing details on new instructions.

Chapter 3 describes the Intel® Advanced Matrix Extensions (Intel® AMX).

Chapter 4 describes Intel® Resource Director Technology feature updates.

Chapter 5 describes Remote Atomic Operations (RAO) in Intel architecture.

Chapter 6 provides a brief introduction to the MOVRS instructions.

Chapter 7 describes Intel® Software Guard Extensions feature updates.

Chapter 8 describes the Read Performance-Monitoring Counter instruction.

Chapter 9 describes Performance Monitoring Unit feature enhancements.

Chapter 10 describes performance monitoring virtualization.

Chapter 11 provides machine check error codes.

1.2 DISPLAYFAMILY AND DISPLAYMODEL FOR FUTURE PROCESSORS

Table 1-1 lists the signature values of DisplayFamily and DisplayModel for future processor families discussed in this document.

Table 1-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_B6H	Processors based on Grand Ridge microarchitecture.
06_C6H	Processors supporting Arrow Lake S hybrid architecture.
06_AFH	Processors based on Sierra Forest microarchitecture.
06_ADH, 06_AEH	Processors based on Granite Rapids microarchitecture.
06_BDH	Processors supporting Lunar Lake hybrid architecture.
06_B5H	Processors supporting Arrow Lake U hybrid architecture.
06_C5H	Processors supporting Arrow Lake H hybrid architecture.
06_CCH	Processors supporting Panther Lake hybrid architecture.

Table 1-1. CPUID Signature Values of DisplayFamily_DisplayModel (Continued)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_D5H	Processors supporting Wildcat Lake hybrid architecture.
06_DDH	Processors based on Clearwater Forest microarchitecture.
12_01H	Future processors supporting Nova Lake desktop hybrid architecture.
12_03H	Future processors supporting Nova Lake mobile hybrid architecture.
13_01H	Future processors based on Diamond Rapids Server microarchitecture.

1.3 INSTRUCTION SET EXTENSIONS AND FEATURE INTRODUCTION IN INTEL® 64 AND IA-32 PROCESSORS

Recent instruction set extensions and features are listed in Table 1-2. Within these groups, most instructions and features are collected into functional subgroups.

Table 1-2. Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32 Processors[1]

Instruction Set Architecture / Feature	Introduction
AVX512_VP2INTERSECT	Tiger Lake (not currently supported in any other processors)
Intel® TSX Suspend Load Address Tracking (TSXLDTRK)	Sapphire Rapids
Intel® Advanced Matrix Extensions (Intel® AMX) Includes CPUID Leaf 1EH, “TMUL Information Main Leaf,” and CPUID bits AMX-BF16, AMX-TILE, and AMX-INT8.	Sapphire Rapids
User Interrupts (UINTR)	Sapphire Rapids, Sierra Forest, Grand Ridge, Arrow Lake H
Intel® Trust Domain Extensions (Intel® TDX)[2]	Emerald Rapids
Linear Address Masking (LAM)	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
IPI Virtualization	Sapphire Rapids, Sierra Forest, Grand Ridge, Arrow Lake H
RAO-INT	Future processors
PREFETCHIT0/1	Granite Rapids, Clearwater Forest, Nova Lake
AMX-FP16	Granite Rapids
CMPCCXADD	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
AVX-IFMA	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
AVX-NE-CONVERT	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
AVX-VNNI-INT8	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
RDMSRLIST, WRMSRLIST, and WRMSRNS	Sierra Forest, Grand Ridge, Panther Lake, Diamond Rapids
Linear Address Space Separation (LASS)	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
Virtualization of guest accesses to IA32_SPEC_CTRL	Sapphire Rapids, Sierra Forest, Grand Ridge, Panther Lake
UC Lock Disable Causes #AC	Sierra Forest, Grand Ridge
LBR Event Logging	Sierra Forest, Grand Ridge, Arrow Lake S, Diamond Rapids
AMX-COMPLEX	Granite Rapids D (06_AEH), Diamond Rapids
AVX-VNNI-INT16	Arrow Lake S, Clearwater Forest, Diamond Rapids
SHA512	Arrow Lake S, Clearwater Forest, Diamond Rapids
SM3	Arrow Lake S, Clearwater Forest, Diamond Rapids
SM4 (VEX)	Arrow Lake S, Clearwater Forest, Diamond Rapids

Table 1-2. Recent Instruction Set Extensions / Features Introduction in Intel® 64 and IA-32

Instruction Set Architecture / Feature	Introduction
SM4 (EVEX)	Diamond Rapids, Nova Lake
UIRET flexibly updates UIF	Sierra Forest, Grand Ridge, Arrow Lake H, Diamond Rapids
Intel® Total Storage Encryption and the PBNDKB instruction	Panther Lake
Intel® Advanced Vector Extensions 10 Version 1 (Intel® AVX10.1)[3]	Granite Rapids, Nova Lake
URDMSR and UWRMSR instructions	Clearwater Forest
Flexible Return and Event Delivery (FRED) and the LKGS instruction[4]	Panther Lake, Clearwater Forest, Diamond Rapids
NMI-Source Reporting[4]	Clearwater Forest
APIC-Timer Virtualization	Clearwater Forest
Management of IA32_SPEC_CTRL by VMX transitions	Clearwater Forest, Diamond Rapids
Intel Processor Trace Trigger Tracing	Clearwater Forest
Monitorless MWAIT	Clearwater Forest
Intel® Advanced Performance Extensions (Intel® APX)[5]	Diamond Rapids, Nova Lake
Intel® Advanced Vector Extensions 10 Version 2 (Intel® AVX10.2)[6]	Diamond Rapids, Nova Lake
Architectural PEBS	Panther Lake, Clearwater Forest, Diamond Rapids
RDPMC User Disable	Diamond Rapids, Nova Lake
Immediate encodings for RDMSR and WRMSRNS	Clearwater Forest
MOVRS and the PREFETCHRST2 instruction	Diamond Rapids, Nova Lake
AMX-MOVRS	Diamond Rapids
AMX-AVX512	Diamond Rapids
AMX-FP8	Diamond Rapids
Intel® RDT Region Aware Memory Bandwidth Allocation[7]	Diamond Rapids
Intel® RDT Allocation Asymmetric Enumeration	Panther Lake, Clearwater Forest, Diamond Rapids
Intel® RDT Monitoring Asymmetric Enumeration	Panther Lake, Clearwater Forest, Diamond Rapids

NOTES:
1. Visit for Intel® product specifications, features and compatibility quick reference guide, and code name decoder, visit: https://ark.intel.com/content/www/us/en/ark.html.
2. Details on Intel® Trust Domain Extensions can be found here: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html.
3. Details on Intel® Advanced Vector Extensions 10 Version 1 can be found here: https://cdrdv2.intel.com/v1/dl/getContent/784267.
4. Details on the LKGS (load into IA32_KERNEL_GS_BASE) instruction, NMI-source reporting, and Flexible Return and Event Delivery can be found here: https://cdrdv2.intel.com/v1/dl/getContent/795033.
5. Details on Intel® Advanced Performance Extensions can be found here: https://cdrdv2.intel.com/v1/dl/getContent/784266
6. Details on Intel® Advanced Vector Extensions 10 Version 2 can be found here: https://cdrdv2.intel.com/v1/dl/getContent/828965.
7. Details on Intel® RDT Region Aware Memory Bandwidth Allocation can be found here: https://cdrdv2.intel.com/v1/dl/getContent/789566.

1.4 DETECTION OF FUTURE INSTRUCTIONS AND FEATURES

Future instructions and features are enumerated by a CPUID feature flag; details can be found in Table 1-3 below. 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. Chapter 3 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A provides detail on using the CPUID—CPU Identification instruction.

CPUID—CPU Identification

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
0F A2	CPUID	Valid	Valid	Returns processor identification and feature information to the EAX, EBX, ECX, and EDX registers, as determined by input entered in EAX (in some cases, ECX as well).

Table 1-3. Information Returned by CPUID Instruction

Initial EAX Value	Information Provided about the Processor	
Basic CPUID Information		
0H	EAX	MAX_LEAF. Maximum Input Value for Basic CPUID Information.
	EBX	"Genu"
	ECX	"ntel"
	EDX	"ineI"
01H	EAX	Version Information: Type, Family, Model, and Stepping ID (see Figure 1-1).
	EBX	Bits 7-0: BRAND_INDEX. Bits 15-8: CLFLUSH_LINE_SIZE. Value ∗ 8 = cache line size in bytes. Bits 23-16: APIC_ID_SPACE. Maximum number of addressable IDs for logical processors in this physical package.* Bits 31-24: INITIAL_APIC_ID.**
	ECX	Feature Information (see Figure 1-2 and Table 1-5).
	EDX	Feature Information (see Figure 1-3 and Table 1-6).
		NOTES: * The nearest power-of-2 integer that is not smaller than EBX[23:16] is the maximum number of unique initial APIC IDs reserved for addressing different logical processors in a physical package. ** 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.
02H	EAX	Cache and TLB Information (see Table 1-7).
	EBX	Cache and TLB Information.
	ECX	Cache and TLB Information.
	EDX	Cache and TLB Information.
03H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 00-31 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
	EDX	Bits 32-63 of 96 bit processor serial number. (Available in Pentium III processor only; otherwise, the value in this register is reserved.)
		NOTES: Processor serial number (PSN) is not supported in the Pentium 4 processor or later. On all models, use the PSN flag (returned using CPUID) to check for PSN support before accessing the feature.
CPUID leaves > 3 < 80000000 are visible only when IA32_MISC_ENABLES.BOOT_NT4[bit 22] = 0 (default)		

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
Deterministic Cache Parameters Leaf (Initial EAX Value = 04H)		
04H		**NOTES:** Leaf 04H output depends on the initial value in ECX. See also: "INPUT EAX = 4: Returns Deterministic Cache Parameters for each level" on page 1-46.
	EAX	Bits 4-0: CACHE_TYPE. 0 = Null - No more caches. 1 = Data Cache. 2 = Instruction Cache. 3 = Unified Cache. 4-31 = Reserved. Bits 7-5: CACHE_LEVEL (starts at 1). Bits 8: SELF_INITIALIZING_CACHE (does not need SW initialization). Bits 9: FULLY_ASSOC (fully associative cache).
		Bits 13-10: Reserved. Bits 25-14: MAX_LP_ADDRESSABLE_IDS. Maximum number of addressable IDs for logical processors sharing this cache.*, ** Bits 31-26: MAX_CORES_ADDRESSABLE_IDS_PKG. Maximum number of addressable IDs for processor cores in the physical package.*, ***, ****
	EBX	Bits 11-00: LINE_SIZE. L = System Coherency Line Size.* Bits 21-12: PHYS_LINE_PARTITIONS. P = Physical Line partitions.* Bits 31-22: NUM_WAYS. W = Ways of associativity.*
	ECX	Bits 31-00: S = NUM_SETS. Number of Sets.*
	EDX	Bit 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. Bit 1: INCLUSIVE_CACHE. 0 = Cache is not inclusive of lower cache levels. 1 = Cache is inclusive of lower cache levels. Bit 2: COMPLEX_CACHE_INDEXING. 0 = Direct mapped cache. 1 = A complex function is used to index the cache, potentially using all address bits. Bits 31-03: Reserved. **NOTES:** * 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. *** 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.
MONITOR/MWAIT Leaf (Initial EAX Value = 05H)		
05H	EAX	Bits 15-00: SMALLEST_MONITOR_LINE_SIZE (in bytes; default is processor's monitor granularity). Bits 31-16: Reserved.
	EBX	Bits 15-00: LARGEST_MONITOR_LINE_SIZE (in bytes; default is processor's monitor granularity). Bits 31-16: Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: MONITOR_MWAIT_EXTENSIONS. Enumeration of Monitor-Mwait extensions (beyond EAX and EBX registers) supported. Bit 01: INTERRUPT_AS_BREAK_EVENT. Supports treating interrupts as break-event for MWAIT, even when interrupts are disabled. Bit 02: Reserved. Bit 03: MONITORLESS_MWAIT. If 1, monitorless MWAIT is supported. Bits 31-04: Reserved.
	EDX	Bits 03-00: C0_SUB_STATES. Number of C0* sub C-states supported using MWAIT. Bits 07-04: C1_SUB_STATES. Number of C1* sub C-states supported using MWAIT. Bits 11-08: C2_SUB_STATES. Number of C2* sub C-states supported using MWAIT. Bits 15-12: C3_SUB_STATES. Number of C3* sub C-states supported using MWAIT. Bits 19-16: C4_SUB_STATES. Number of C4* sub C-states supported using MWAIT. Bits 23-20: C5_SUB_STATES. Number of C5* sub C-states supported using MWAIT. Bits 27-24: C6_SUB_STATES. Number of C6* sub C-states supported using MWAIT. Bits 31-28: C7_SUB_STATES. Number of C7* sub C-states supported using MWAIT. **NOTE:** * The definition of C0 through C7 states for MWAIT extension are processor-specific C-states, not ACPI C-states.
Thermal and Power Management Leaf (Initial EAX Value = 06H)		
06H	EAX	Bit 00: DIGITAL_TEMP_SENSOR. Digital temperature sensor is supported if set. Bit 01: TURBO_BOOST. Intel® Turbo Boost Technology Available (see description of IA32_MISC_ENABLE[38]). Bit 02: ALWAYS_RUNNING_APIC_TIMER. APIC-Timer-always-running feature is supported if set. Bit 03: Reserved. Bit 04: POWER_LIMIT_NOTIFY. Power limit notification controls are supported if set. Bit 05: EXT_CLOCK_MOD. Clock modulation duty cycle extension is supported if set. Bit 06: PKG_THERM_MGMT. Package thermal management is supported if set. Bit 07: HWP. HWP base registers (IA32_PM_ENABLE[bit 0], IA32_HWP_CAPABILITIES, IA32_HWP_REQUEST, IA32_HWP_STATUS) are supported if set. Bit 08: HWP_INTERRUPT. IA32_HWP_INTERRUPT MSR is supported if set. Bit 09: HWP_ACTIVITY_WINDOW. IA32_HWP_REQUEST[bits 41:32] is supported if set. Bit 10: HWP_EPP. IA32_HWP_REQUEST[bits 31:24] is supported if set. Bit 11: HWP_REQUEST_PKG. IA32_HWP_REQUEST_PKG MSR is supported if set. Bit 12: Reserved. Bit 13: HDC. HDC base registers IA32_PKG_HDC_CTL, IA32_PM_CTL1, IA32_THREAD_STALL MSRs are supported if set. Bit 14: TURBO_BOOST_MAX. Intel® Turbo Boost Max Technology 3.0 available. Bit 15: HWP_CAP. Highest Performance change is supported if set. Bit 16: HWP_PECI_OVERRIDE. HWP PECI override is supported if set. Bit 17: FLEXIBLE_HWP. Flexible HWP is supported if set. Bit 18: HWP_REQUEST_FAST_ACCESS. Fast access mode, low latency, and posted IA32_HWP_REQUEST MSR are supported if set. Bit 19: HW_FEEDBACK. IA32_HW_FEEDBACK_PTR, IA32_HW_FEEDBACK_CONFIG, IA32_PACKAGE_THERM_STATUS bit 26 and IA32_PACKAGE_THERM_INTERRUPT bit 25 are supported if set. Bit 20: HWP_REQUEST_IGNORE_IDLE. Ignoring Idle Logical Processor HWP request is supported if set. Bit 21: Reserved. Bit 22: HWP_CTL. The IA32_HWP_CTL MSR is supported if set. Bit 23: THREAD_DIRECTOR. Intel® Thread Director supported if set. IA32_HW_FEEDBACK_CHAR and IA32_HW_FEEDBACK_THREAD_CONFIG MSRs are supported if set. Bits 31-24: Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EBX	Bits 03-00: DTS_NUM_INT_THRESHOLDS. Number of Interrupt Thresholds in Digital Thermal Sensor. Bits 31-04: Reserved.
	ECX	Bit 00: HW_FEEDBACK_CAP. Hardware Coordination Feedback Capability (Presence of IA32_MPERF and IA32_APERF). The capability to 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. Bits 02-01: Reserved. Bit 03: ENERGY_PERF_BIAS. The processor supports performance-energy bias preference if CPUID.06H:ECX.SETBH[bit 3] is set, and it also implies the presence of the IA32_ENERGY_PERF_BIAS MSR (MSR address 1B0H). Bits 07-04: Reserved. Bits 15-08: 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. Bits 31-16: Reserved.
	EDX	Bits 7-0: HW_FEEDBACK_CAPS. Bitmap of supported hardware feedback interface capabilities. 0 = When set to 1, indicates support for performance capability reporting. 1 = When set to 1, indicates support for energy efficiency capability reporting. 2-7 = Reserved. Bits 11-08: 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. Bits 15-12: Reserved. Bits 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. **NOTE:** Bits 0 and 1 will always be set together.
Structured Extended Feature Flags Enumeration Main Leaf (Initial EAX Value = 07H, ECX = 0)		
07H		**NOTE:** 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.
	EAX	Bits 31-00: MAX_SUBLEAF. Reports the maximum number sub-leaves that are supported in leaf 07H.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor
	EBX Bit 00: FSGSBASE. Supports RDFSBASE/RDGSBASE/WRFSBASE/WRGSBASE if 1. Bit 01: TSC_ADJUST. IA32_TSC_ADJUST MSR is supported if 1. Bit 02: SGX. Bit 03: BMI1. Bit 04: HLE. Bit 05: AVX2. Supports Intel® Advanced Vector Extensions 2 (Intel® AVX2) if 1. Bit 06: FDP_EXCPTN_ONLY. x87 FPU Data Pointer updated only on x87 exceptions if 1. Bit 07: SMEP. Supports Supervisor Mode Execution Protection if 1. Bit 08: BMI2. Bit 09: ENH_REP_MOVSB_STOSB. Supports Enhanced REP MOVSB/STOSB if 1. Bit 10: INVPCID. Bit 11: RTM. Bit 12: RDT_M. Supports Intel® Resource Director Technology (Intel® RDT) Monitoring capability if 1. Bit 13: FCS_FDS_DEPRECATION. Deprecates FPU CS and FPU DS values if 1. Bit 14: MPX. Intel® Memory Protection Extensions. Bit 15: RDT_A. Supports Intel® Resource Director Technology (Intel® RDT) Allocation capability if 1. Bit 16: AVX512F. Bit 17: AVX512DQ. Bit 18: RDSEED. Bit 19: ADX. Bit 20: SMAP. Bit 21: AVX512_IFMA. Bit 22: Reserved. Bit 23: CLFLUSHOPT.
	Bit 24: CLWB. Bit 25: INTEL_PROC_TRACE (Intel Processor Trace). Bit 26: AVX512PF. (Intel® Xeon Phi™ only.) Bit 27: AVX512ER. (Intel® Xeon Phi™ only.) Bit 28: AVX512CD. Bit 29: SHA. Bit 30: AVX512BW. Bit 31: AVX512VL.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bit 00: PREFETCHWT1. (Intel® Xeon Phi™ only.) Bit 01: AVX512_VBMI. Bit 02: UMIP. Supports user-mode instruction prevention if 1. Bit 03: PKU. Supports protection keys for user-mode pages if 1. Bit 04: OSPKE. If 1, OS has set CR4.PKE to enable protection keys (and the RDPKRU/WRPKRU instructions). Bit 05: WAITPKG. Bit 06: AVX512_VBMI2. Bit 07: CET_SS. Supports CET shadow stack features if 1. 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_PL0_SSP. Bit 08: GFNI. Bit 09: VAES. Bit 10: VPCLMULQDQ. Bit 11: AVX512_VNNI. Bit 12: AVX512_BITALG. Bit 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. Bit 14: AVX512_VPOPCNTDQ. Bit 15: Reserved. Bit 16: LA57. Supports 57-bit linear addresses and five-level paging if 1. Bits 21-17: MPX_MAWAU. The value of MAWAU used by the BNDLDX and BNDSTX instructions in 64-bit mode. Bit 22: RDPID. RDPID and IA32_TSC_AUX are available if 1. Bit 23: KEY_LOCKER. Supports Key Locker if 1. Bit 24: BUS_LOCK_DETECT. If 1, indicates support for bus lock debug exceptions. Bit 25: CLDEMOTE. Supports cache line demote if 1. Bit 26: Reserved. Bit 27: MOVDIRI. Supports MOVDIRI if 1. Bit 28: MOVDIR64B. Supports MOVDIR64B if 1. Bit 29: ENQCMD: Supports Enqueue Stores if 1. Bit 30: SGX_LC. Supports SGX Launch Configuration if 1. Bit 31: PKS. Supports protection keys for supervisor-mode pages if 1.
	EDX	Bit 00: Reserved. Bit 01: SGX_KEYS. If 1, Attestation Services for Intel® SGX is supported. Bit 02: AVX512_4VNNIW. (Intel® Xeon Phi™ only.) Bit 03: AVX512_4FMAPS. (Intel® Xeon Phi™ only.) Bit 04: FAST_SHORT_REP_MOVSB (Fast Short REP MOVSB). Bit 05: UINTR. If 1, the processor supports user interrupts. Bits 07-06: Reserved. Bit 08: AVX512_VP2INTERSECT. Bit 09: MCU_OPT_CTRL. If 1, enumerates support for the IA32_MCU_OPT_CTRL MSR and indicates that its bit 0 (RNGDS_MITG_DIS) is also supported. Bit 10: MD_CLEAR supported. Bit 11: RTM_ALWAYS_ABORT. If set, any execution of XBEGIN immediately aborts and transitions to the specified fallback address. Bit 12: Reserved. Bit 13: RTM_FORCE_ABORT. If 1, RTM_FORCE_ABORT supported. Processors that set this bit support the TSX_FORCE_ABORT MSR. They allow software to set TSX_FORCE_ABORT[0] (RTM_-FORCE_ABORT). Bit 14: SERIALIZE. Bit 15: HYBRID. If 1, the processor is identified as a hybrid part. If CPUID.0.MAXLEAF ≥ 1AH and CPUID.1A.EAX ≠ 0, then the Native Model ID Enumeration Leaf 1AH exists.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
		Bit 16: TSXLDTRK. If 1, the processor supports Intel TSX suspend/resume of load address tracking. Bit 17: Reserved. Bit 18: PCONFIG. Bit 19: ARCH_LBRS. If 1, indicates support for architectural LBRs. Bit 20: CET_IBT. Supports CET indirect branch tracking features if 1. Processors that set this bit define bits 5:2 and bits 63:10 of the IA32_U_CET and IA32_S_CET MSRs. Bit 21: Reserved. Bit 22: AMX_BF16. If 1, the processor supports tile computational operations on bfloat16 numbers. Bit 23: AVX512_FP16. Bit 24: AMX_TILE. If 1, the processor supports tile architecture. Bit 25: AMX_INT8. If 1, the processor supports tile computational operations on 8-bit integers. Bit 26: IBRS_IBPB. Enumerates support for 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). Bit 27: SPEC_CTRL_ST_PREDICTORS. Enumerates support for 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). Bit 28: L1D_FLUSH_INTERFACE. Enumerates support for L1D_FLUSH. Processors that set this bit support the IA32_FLUSH_CMD MSR. They allow software to set IA32_FLUSH_CMD[0] (L1D_FLUSH). Bit 29: ARCH_CAPABILITIES. Enumerates support for the IA32_ARCH_CAPABILITIES MSR. Bit 30: CORE_CAPABILITIES. Enumerates support for the IA32_CORE_CAPABILITIES MSR. IA32_CORE_CAPABILITIES is an architectural MSR that enumerates model-specific features. In general, a bit being set in this MSR indicates that a model-specific feature is supported; software should consult CPUID family/model/stepping to determine the behavior of these enumerated features, as that behavior may differ on different processor models. Some bits in the MSR enumerate features with behavior that is consistent across processor models (and for which consultation of CPUID family/model/stepping is not necessary); such bits are identified explicitly in the documentation of the IA32_CORE_CAPABILITIES MSR. Bit 31: SPEC_CTRL_SSBD. Enumerates support for 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).
Structured Extended Feature Enumeration Sub-leaf (Initial EAX Value = 07H, ECX = 1)		
07H		**NOTES:** Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.
	EAX	This field reports 0 if the sub-leaf index, 1, is invalid. Bit 00: SHA512. If 1, supports the SHA512 instructions. Bit 01: SM3. If 1, supports the SM3 instructions. Bit 02: SM4. If 1, supports the SM4 instructions. Bit 03: RAO_INT. If 1, supports the RAO-INT instructions. Bit 04: AVX_VNNI. AVX (VEX-encoded) versions of the Vector Neural Network Instructions. Bit 05: AVX512_BF16. Vector Neural Network Instructions supporting bfloat16 inputs and conversion instructions from IEEE single precision. Bit 06: LASS. If 1, supports Linear Address Space Separation. Bit 07: CMPCCXADD. If 1, supports the CMPccXADD instruction. Bit 08: ARCH_PERFMON_EXT. If 1, supports ArchPerfMonExt. When set, indicates that the Architectural Performance Monitoring Extended Leaf (EAX = 23H) is valid. Bit 09: Reserved. Bit 10: FAST_REP_MOVSB. If 1, supports fast zero-length MOVSB. Bit 11: FAST_REP_STOSB. If 1, supports fast short STOSB. Bit 12: FAST_REP_CMPSB_SCASB. If 1, supports fast short CMPSB, SCASB.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
		Bits 16-13: Reserved. Bit 17: FRED. If 1, supports Flexible Return and Event Delivery and the architectural state (MSRs) defined by FRED. Any Intel processor that enumerates support for FRED transitions will also enumerate support for LKGS. Bit 18: LKGS. If 1, supports the LKGS (load into IA32_KERNEL_GS_BASE) instruction. Bit 19: WRMSRNS. If 1, supports the WRMSRNS instruction. Bit 20: NMI_SRC. If 1, supports NMI-source reporting. Bit 21: AMX_FP16. If 1, the processor supports tile computational operations on FP16 numbers. Bit 22: HRESET. If 1, supports history reset and the IA32_HRESET_ENABLE MSR. When set, indicates that the Processor History Reset Leaf (EAX = 20H) is valid. Bit 23: AVX_IFMA. If 1, supports the AVX-IFMA instructions. Bits 25-24: Reserved. Bit 26: LAM. If 1, supports Linear Address Masking. Bit 27: MSRLIST. If 1, supports the RDMSRLIST and WRMSRLIST instructions and the IA32_BARRIER MSR. Bits 29-28: Reserved. Bit 30: INVD_DISABLE_POST_BIOS_DONE. If 1, supports INVD execution prevention after BIOS Done. Bit 31: MOVRS.
	EBX	This field reports 0 if the sub-leaf index, 1, is invalid; otherwise it is reserved. Bit 00: PPIN. Enumerates the presence of the IA32_PPIN and IA32_PPIN_CTL MSRs. If 1, these MSRs are supported. Bit 01: PBNDKB. If 1, supports the PBNDKB instruction and enumerates the existence of the IA32_TSE_CAPABILITY MSR. Bit 02: Reserved. Bit 03: 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]. Bits 31-04: Reserved.
	ECX	This field reports 0 if the sub-leaf index, 1, is invalid; otherwise it is reserved. Bit 00: RDT_M_ASYM. If 1, at least one logical processor on this platform supports Asymmetrical Intel® RDT Monitoring capability.[1] Bit 01: RDT_A_ASYM. If 1, at least one logical processor on this platform supports Asymmetrical Intel® RDT Allocation capability.[1] Bits 04-02: Reserved. Bit 05: MSR_IMM. If 1, the immediate forms of the RDMSR and WRMSRNS instructions are supported. Bits 31-06: Reserved.
	EDX	This field reports 0 if the sub-leaf index, 1, is invalid. Bit 03-00: Reserved. Bit 04: AVX_VNNI_INT8. If 1, supports the AVX-VNNI-INT8 instructions. Bit 05: AVX_NE_CONVERT. If 1, supports the AVX-NE-CONVERT instructions. Bits 07-06: Reserved. Bit 08: AMX_COMPLEX. If 1, supports the AMX-COMPLEX instructions. Bit 09: Reserved. Bit 10: AVX_VNNI_INT16. If 1, supports the AVX-VNNI-INT16 instructions. Bits 13-11: Reserved. Bit 14: PREFETCHI. If 1, supports the PREFETCHIT0/1 instructions. Bit 15: USER_MSR. If 1, supports the URDMSR and UWRMSR instructions.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
		Bits 16: Reserved. Bit 17: UIRET_UIF. If 1, UIRET sets UIF to the value of bit 1 of the RFLAGS image loaded from the stack. Bit 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. Bit 19: AVX10. If 1, supports the Intel® AVX10 instructions and indicates the presence of CPUID Leaf 24H, which enumerates version number. Bit 20: Reserved. Bit 21: APX_F. If 1, the processor provides foundational support for Intel® Advanced Performance Extensions. Bit 22: SEC_TEE_ATTESTATION. If 1, Trusted Execution Environment technologies (Trust Domain Extensions and Software Guard Extensions) support attestation rooted in the Security Engine. Bit 23: MWAIT. If 1, MWAIT is supported (even if CPUID.01H:ECX.MONITOR[bit 3] is enumerated as 0). Bit 24: SLSM. If 1, indicates bit 0 of the IA32_INTEGRITY_STATUS MSR is supported. Bit 0 of this MSR indicates whether static lockstep is active on this logical processor. Bits 31-25: Reserved.
Structured Extended Feature Enumeration Sub-leaf (Initial EAX Value = 07H, ECX = 2)		
07H		**NOTES:** Leaf 07H output depends on the initial value in ECX. If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.
	EAX	This field reports 0 if the sub-leaf index, *2*, is invalid; otherwise it is reserved.
	EBX	This field reports 0 if the sub-leaf index, *2*, is invalid; otherwise it is reserved.
	ECX	This field reports 0 if the sub-leaf index, *2*, is invalid; otherwise it is reserved.
	EDX	This field reports 0 if the sub-leaf index, 2, is invalid. Bit 00: PSFD. If 1, indicates bit 7 of the IA32_SPEC_CTRL MSR is supported. Bit 7 of this MSR disables Fast Store Forwarding Predictor without disabling Speculative Store Bypass. Bit 01: IPRED_CTRL. If 1, indicates bits 3 and 4 of the IA32_SPEC_CTRL MSR are supported. Bit 3 of this MSR enables IPRED_DIS control for CPL3. Bit 4 of this MSR enables IPRED_DIS control for CPL0/1/2. Bit 02: RRSBA_CTRL. If 1, indicates bits 5 and 6 of the IA32_SPEC_CTRL MSR are supported. Bit 5 of this MSR disables RRSBA behavior for CPL3. Bit 6 of this MSR disables RRSBA behavior for CPL0/1/2. Bit 03: DDPD_U. If 1, indicates bit 8 of the IA32_SPEC_CTRL MSR is supported. Bit 8 of this MSR disables Data Dependent Prefetcher. Bit 04: BHI_CTRL. If 1, indicates bit 10 of the IA32_SPEC_CTRL MSR is supported. Bit 10 of this MSR enables BHI_DIS_S behavior. Bit 05: MCDT_NO. Processors that enumerate this bit as 1 do not exhibit MXCSR Configuration Dependent Timing (MCDT) behavior and do not need to be mitigated to avoid data-dependent behavior for certain instructions. Bit 06: UC_LOCK_DISABLE. If 1, supports the UC-lock disable feature and it causes #AC. Bit 07: MONITOR_MITG_NO. If 1, indicates that 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. Bits 31-08: Reserved.
Structured Extended Feature Enumeration Sub-leaves (Initial EAX Value = 07H, ECX = n, n > 2)		

Table 1-3. Information Returned by CPUID Instruction (Continued)

<table>
<tr><th>Initial EAX Value</th><th colspan="2">Information Provided about the Processor</th></tr>
<tr><td>07H</td><td></td><td>NOTES:
Leaf 07H output depends on the initial value in ECX.
If ECX contains an invalid sub leaf index, EAX/EBX/ECX/EDX return 0.</td></tr>
<tr><td></td><td>EAX</td><td>This field reports 0 if the sub-leaf index, n, is invalid; otherwise it is reserved.</td></tr>
<tr><td></td><td>EBX</td><td>This field reports 0 if the sub-leaf index, n, is invalid; otherwise it is reserved.</td></tr>
<tr><td></td><td>ECX</td><td>This field reports 0 if the sub-leaf index, n, is invalid; otherwise it is reserved.</td></tr>
<tr><td></td><td>EDX</td><td>This field reports 0 if the sub-leaf index, n, is invalid; otherwise it is reserved.</td></tr>
<tr><td colspan="3">Direct Cache Access Information Leaf (Initial EAX Value = 09H)</td></tr>
<tr><td>09H</td><td>EAX</td><td>PLATFORM_DCA_CAP. Value of bits [31:0] of IA32_PLATFORM_DCA_CAP MSR (address 1F8H).</td></tr>
<tr><td></td><td>EBX</td><td>Reserved.</td></tr>
<tr><td></td><td>ECX</td><td>Reserved.</td></tr>
<tr><td></td><td>EDX</td><td>Reserved.</td></tr>
<tr><td colspan="3">Architectural Performance Monitoring Leaf (Initial EAX Value = 0AH)</td></tr>
<tr><td>0AH</td><td>EAX</td><td>Bits 07-00: VERSION. Version ID of architectural performance monitoring.
Bits 15- 08: NUM_GP_CTRS. Number of general-purpose performance monitoring counter per logical processor.
Bits 23-16: GP_CTR_WIDTH. Bit width of general-purpose, performance monitoring counter.
Bits 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.</td></tr>
<tr><td></td><td>EBX</td><td>Bit 00: CORE_CYC_NA. Core cycle event not available if 1 or if EAX[31:24] < 1.
Bit 01: INTR_RET_NA. Instruction retired event not available if 1 or if EAX[31:24] < 2.
Bit 02: REF_CYC_NA. Reference cycles event not available if 1 or if EAX[31:24] < 3.
Bit 03: LLC_CYC_NA. Last-level cache reference event not available if 1 or if EAX[31:24] < 4.
Bit 04: LLC_MISSES_NA. Last-level cache misses event not available if 1 or if EAX[31:24] < 5.
Bit 05: BR_INSTR_RET_NA. Branch instruction retired event not available if 1 or if EAX[31:24] < 6.
Bit 06: BR_MISPRED_RET_NA. Branch mispredict retired event not available if 1 or if EAX[31:24] < 7.
Bit 07: SLOTS_NA. Topdown slots event not available if 1 or if EAX[31:24] < 8.
Bit 08: BACKEND_NA. Topdown backend bound not available if 1 or if EAX[31:24] < 9.
Bit 09: BADSPEC_NA. Topdown bad speculation not available if 1 or if EAX[31:24] < 10.
Bit 10: FRONTEND_NA. Topdown frontend bound not available if 1 or if EAX[31:24] < 11.
Bit 11: RETIRING_NA. Topdown retiring not available if 1 or if EAX[31:24] < 12.
Bit 12: LBR_INSERTS_NA. LBR inserts not available if 1 or if EAX[31:24] < 13.
Bits 31-13: Reserved.</td></tr>
<tr><td></td><td>ECX</td><td>Bits 31-00: FIXED_CTR_MASK. Supported fixed-function counters. If bit 'i' is set, it implies that Fixed-Function Counter 'i' is supported.
Software is recommended to use the following logic to check if a Fixed-Function Counter is supported on a given processor: FxCtr[i]_is_supported := ECX[i] || (EDX[4:0] > i);[2]</td></tr>
<tr><td></td><td>EDX</td><td>Bits 04-00: NUM_FIXED_CTR. Number of contiguous fixed-function performance counters starting from 0 (if Version ID > 1).[2]
Bits 12-05: FIXED_CTR_WIDTH. Bit width of fixed-function performance counters (if Version ID > 1).
Bits 14-13: Reserved.
Bit 15: ANYTHREAD_DEPRECATION (AnyThread deprecation).
Bits 19-16: SLOTS_PER_CYC.
Bits 31-20: Reserved.</td></tr>
<tr><td colspan="3">Extended Topology Enumeration Leaf (Initial EAX Value = 0BH, ECX ≥ 0)</td></tr>
</table>

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
0BH		**NOTES:** CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH. The sub-leaves of CPUID leaf 0BH 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. 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 and 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.
	EAX	Bits 04-00: 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. Bits 31-05: Reserved.
	EBX	Bits 15-00: 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" 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.) This number reflects configuration as shipped by Intel. Note, software must not use this field to enumerate processor topology*. Bits 31-16: Reserved.
	ECX	Bits 07-00: LEVEL_NUM. The input ECX sub-leaf index. Bits 15-08: DOMAIN_TYPE. This field provides an identification value which indicates the domain as shown below. Although domains are ordered, their assigned identification values are not and software should not depend on it. <u>Hierarchy</u> / <u>Domain</u> / <u>Domain Type Identification Value</u> Lowest / Logical Processor / 1 Highest / Core / 2 (Note that enumeration values of 0 and 3-255 are reserved.) Bits 31-16: Reserved.
	EDX	Bits 31-00: X2APIC_ID. x2APIC ID of the current logical processor. **NOTE:** * 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.
Processor Extended State Enumeration Main Leaf (Initial EAX Value = 0DH, ECX = 0)		
0DH		**NOTE:** Leaf 0DH main leaf (ECX = 0).

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	EAX	Bits 31-00: Reports the valid bit fields of the lower 32 bits of the XFEATURE_ENABLED_MASK register. If a bit is 0, the corresponding bit field in XCR0 is reserved. Bit 00: X87. x87 state. Bit 01: SSE. SSE state. Bit 02: AVX. AVX state. Bit 03: MPX_BNDREGS. MPX state. Bit 04: MPX_BNDCSR. MPX state. Bit 05: AVX512_OPMASK. AVX-512 opmask state. Bit 06: AVX512_ZMM_HI256. AVX-512 ZMM upper 256 data state. Bit 07: AVX512_HI16_ZMM. AVX-512 upper 16 ZMM registers state. Bit 08: N/A. Always returns 0 (allocated for IA32_XSS). Bit 09: PKRU. PKRU state. Bits 16-10: N/A. Always returns 0 (allocated for IA32_XSS). Bit 17: AMX_TILECFG. TILECFG state. Bit 18: AMX_TILEDATA. TILEDATA state. Bits 31-19: Reserved.
	EBX	Bits 31-00: 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.
	ECX	Bit 31-00: 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 processor, i.e all the valid bit fields in XCR0.
	EDX	Bit 31-00: VALID_XCR0_UPPER_32. Reports the valid bit fields of the upper 32 bits of the XCR0 register. If a bit is 0, the corresponding bit field in XCR0 is reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
Processor Extended State Enumeration Sub-leaf (Initial EAX Value = 0DH, ECX = 1)		
0DH	EAX	Bit 00: XSAVEOPT. XSAVEOPT is available. Bit 01: XSAVEC. Supports XSAVEC and the compacted form of XRSTOR if set. Bit 02: XGETBV1. Supports XGETBV with ECX = 1 if set. Bit 03: XSAVES. Supports XSAVES/XRSTORS and IA32_XSS if set. Bit 04: XFD. Supports Extended Feature Disable (XFD) if set. Bits 31-05: Reserved.
	EBX	Bits 31-00: XSAVES_BYTES_ENABLED_FEATURES. The size in bytes of the XSAVE area containing all states enabled by XCR0 | IA32_XSS. **NOTE:** 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 XCR0. If EAX[1] and EAX[3] are both enumerated as 0, EBX enumerates zero.
	ECX	Register 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. Bits 07-00: N/A. Always returns 0 (allocated for XCR0). Bit 08: PT. PT state. Bit 09: N/A. Always returns 0 (allocated for XCR0). Bit 10: PASID. PASID state. Bit 11: CET_U. CET user state. Bit 12: CET_S. CET supervisor state. Bit 13: HDC. HDC state. Bit 14: UINTR. UINTR state. Bits 15: LBR. LBR state (only for the architectural LBR feature). Bit 16: HWP. HWP state. Bits 18-17: N/A. Always returns 0 (allocated for XCR0). Bits 31-19: Reserved.
	EDX	Bits 31-00: Reserved.
Processor Extended State Enumeration Sub-leaves (Initial EAX Value = 0DH, ECX = n, n > 1)		
0DH		**NOTES:** Leaf 0DH output depends on the initial value in ECX. Each sub-leaf index (starting at position 2) is supported if it corresponds to a supported bit in either the XCR0 register or the IA32_XSS MSR. * If ECX contains an invalid sub-leaf index, EAX/EBX/ECX/EDX return 0. Sub-leaf n (0 ≤ n ≤ 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 ≤ n ≤ 63) is invalid if sub-leaf 0 returns 0 in EDX[n-32] and sub-leaf 1 returns 0 in EDX[n-32].
	EAX	Bits 31-00: 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*. This field reports 0 if the sub-leaf index, *n*, is invalid.*
	EBX	Bits 31-00: 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 XCR0 register.*
	ECX	Bit 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 XCR0. Bit 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).

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
		Bit 2: COMP_XFD. This bit is set to indicate support for XFD faulting. Bits 31-03: Reserved. This field reports 0 if the sub-leaf index, n, is invalid.*
	EDX	This field reports 0 if the sub-leaf index, *n*, is invalid;* otherwise it is reserved.
Intel® Resource Director Technology Monitoring Enumeration Sub-leaf (Initial EAX Value = 0FH, ECX = 0)		
0FH		**NOTES:** Leaf 0FH output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX.
	EAX	Reserved.
	EBX	Bits 31-0: MAX_RMID. Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: L3_MON. Supports L3 Cache Intel RDT Monitoring if 1. Bits 31-02: Reserved.
L3 Cache Intel® RDT Monitoring Capability Enumeration Sub-leaf (Initial EAX Value = 0FH, ECX = 1)		
0FH		**NOTE:** Leaf 0FH output depends on the initial value in ECX.
	EAX	No bits set: 24-bit counters. Bits 07-00: CTR_WIDTH. Encode counter width offset from 24b: 0x0 = 24-bit counters. 0x1 = 25-bit counters. ... 0x25 = 61-bit counters. Bit 08: RDT_M_OVF. If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR (bit 61). Bit 09: IO_RDT_CMT. If 1, indicates the presence of non-CPU agent Intel RDT CMT support. Bit 10: IO_RDT_MBM. If 1, indicates the presence of non-CPU agent Intel RDT MBM support. Bits 31-11: Reserved.
	EBX	Bits 31-00: CONV_FACTOR. Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.
	ECX	Bits 31-00: MAX_RMID_L3. Maximum range (zero-based) of RMID of this resource type.
	EDX	Bit 00: CMT_L3_OCCUP. Supports L3 occupancy monitoring if 1. Bit 01: MBM_L3_TOTAL. Supports L3 Total Bandwidth monitoring if 1. Bit 02: MBM_L3_LOCAL. Supports L3 Local Bandwidth monitoring if 1. Bits 31-03: Reserved.
Intel® Resource Director Technology Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = 0)		
10H		**NOTES:** Leaf 10H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX.
	EAX	Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	EBX	Bit 00: Reserved. Bit 01: CAT_L3. Supports L3 Cache Allocation Technology if 1. Bit 02: CAT_L2. Supports L2 Cache Allocation Technology if 1. Bit 03: MBA. Supports Memory Bandwidth Allocation if 1. Bit 04: Reserved. Bit 05: Supports Cache Bandwidth Allocation if 1. Bits 31-06: Reserved.
	ECX	Reserved.
	EDX	Reserved.
L3 Cache Intel® RDT Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID = 1)		
10H		**NOTE:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 04-00: CAT_L3_BITMASK_LENGTH. Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31-05: Reserved.
	EBX	Bits 31-00: CAT_L3_CONTENTION. Bit-granular map of isolation/contention of allocation units.
	ECX	Bit 00: Reserved. Bit 01: CAT_L3_NONCPU. If 1, indicates L3 CAT for non-CPU agents is supported. Bit 02: CAT_L3_CDP. If 1, indicates L3 Code and Data Prioritization Technology is supported. Bit 03: CAT_L3_NONCONTIG. 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 31-04: Reserved.
	EDX	Bits 15-00: CAT_L3_MAX_CLOS. Highest COS number supported for this ResID. Bits 31-16: Reserved.
L2 Cache Intel® RDT Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID = 2)		
10H		**NOTE:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 04-00: CAT_L2_BITMASK_LENGTH. Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31-05: Reserved.
	EBX	Bits 31-00: CAT_L2_CONTENTION. Bit-granular map of isolation/contention of allocation units.
	ECX	Bits 01-00: Reserved. Bit 02: CAT_L2_CDP. If 1, indicates L2 Code and Data Prioritization Technology is supported. Bit 03: CAT_L2_NONCONTIG. If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 31-04: Reserved.
	EDX	Bits 15-00: CAT_L2_MAX_CLOS. Highest COS number supported for this ResID. Bits 31-16: Reserved.
Memory Bandwidth Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID = 3)		
10H		**NOTE:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 11-00: MBA_MAX. Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result. Bits 31-12: Reserved.
	EBX	Bits 31-00: Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	ECX	Bit 00: PER_THREAD_MBA. Per-thread MBA controls are supported. Bit 01: Reserved. Bit 02: MBA_LINEAR. Reports whether the response of the delay values is linear. Bits 31-03: Reserved.
	EDX	Bits 15-00: MBA_MAX_CLOS. Highest COS number supported for this ResID. Bits 31-16: Reserved.
Cache Bandwidth Allocation Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID = 5)		
10H		**NOTE:** Leaf 10H output depends on the initial value in ECX.
	EAX	Bits 07-00: CBA_MAX_LEVELS. Reports the maximum core throttling level supported for the corresponding ResID. Add one to the return value to get the number of throttling levels supported. Bits 11-08: BW_SCOPE. If 1, indicates the logical processor scope of the IA32_QoS_Core_BW_Thrtl_n MSRs. Other values are reserved. Bits 31-12: Reserved.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 02-00: Reserved. Bit 03: CBA_LINEAR. If 1, the response of the bandwidth control is approximately linear. If 0, the response of the bandwidth control is non-linear. Bits 31-04: Reserved.
	EDX	Bits 15-00: CBA_MAX_CLOS. Highest Class of Service (COS) number supported for this ResID. Bits 31-16: Reserved.
Intel® Software Guard Extensions Capability Enumeration Leaf, Sub-leaf 0 (Initial EAX Value = 12H, ECX = 0)		
12H		**NOTE:** Leaf 12H sub-leaf 0 (ECX = 0) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 00: SGX1. If 1, indicates Intel SGX supports the collection of SGX1 leaf functions. Bit 01: SGX2. If 1, indicates Intel SGX supports the collection of SGX2 leaf functions. Bits 06-02: Reserved. Bit 07: EVERIFYREPORT2. If 1, indicates Intel SGX supports ENCLU instruction leaf EVERIFYREPORT2. Bits 09-08: Reserved. Bit 10: EUPDATESVN. If 1, indicates Intel SGX supports ENCLS instruction leaf EUPDATESVN. Bit 11: EDECCSSA. If 1, indicates Intel SGX supports ENCLU instruction leaf EDECCSSA. Bit 12: 256BITSGX. If 1, indicates Intel SGX supports leaf functions EGETKEY256 and EREPORT2, MSR IA32_SGXLEPUBKEYHASH[4,5] and MSR IA32_SGXLECONFIG. Bits 31-13: Reserved.
	EBX	Bits 31-00: MISCSELECT. Bit vector of supported extended Intel SGX features.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 07-00: MAX_ENCLAVE_SIZE_NOT_64. The maximum supported enclave size in non-64-bit mode is 2^(EDX[7:0]). Bits 15-08: MAX_ENCLAVE_SIZE_64. The maximum supported enclave size in 64-bit mode is 2^(EDX[15:8]). Bits 31-16: Reserved.
Intel® SGX Attributes Enumeration Leaf, Sub-leaf 1 (Initial EAX Value = 12H, ECX = 1)		
12H		**NOTE:** Leaf 12H sub-leaf 1 (ECX = 1) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1.
	EAX	Bit 31-00: ECREATE_SECS_ATTRIBUTES_31_0. Reports the valid bits of SECS.ATTRIBUTES[31:0] that software can set with ECREATE.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EBX	Bit 31-00: ECREATE_SECS_ATTRIBUTES_63_32. Reports the valid bits of SECS.ATTRIBUTES[63:32] that software can set with ECREATE.
	ECX	Bit 31-00: ECREATE_SECS_ATTRIBUTES_95_64. Reports the valid bits of SECS.ATTRIBUTES[95:64] that software can set with ECREATE.
	EDX	Bit 31-00: ECREATE_SECS_ATTRIBUTES_127_96. Reports the valid bits of SECS.ATTRIBUTES[127:96] that software can set with ECREATE.
Intel® SGX EPC Enumeration Leaf, Sub-leaves (Initial EAX Value = 12H, ECX = 2 or higher)		
12H		**NOTES:** Leaf 12H sub-leaf 2 or higher (ECX >= 2) is supported if CPUID.(EAX=07H, ECX=0H):EBX[SGX] = 1. For sub-leaves (ECX = 2 or higher), definition of EDX,ECX,EBX,EAX[31:4] depends on the sub-leaf type listed below.
	EAX	Bit 03-00: SUB_LEAF_TYPE. 0000b: Indicates this sub-leaf is invalid. 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.
	Type	0000b. This sub-leaf is invalid. EDX:ECX:EBX:EAX return 0.
	Type	0001b. This sub-leaf enumerates an EPC sections with EDX:ECX, EBX:EAX defined as follows: EAX[11:04]: Reserved (enumerate 0). EAX[31:12]: EPC_SECTION_ADDR_31_12. Bits 31:12 of the physical address of the base of the EPC section. EBX[19:00]: EPC_SECTION_ADDR_51_32. Bits 51:32 of the physical address of the base of the EPC section. EBX[31:20]: Reserved. ECX[03:00]: EPC_SECTION_PROPERTY. EPC section property encoding defined as follows: If ECX[3:0] = 0000b, then all bits of the EDX:ECX pair are enumerated as 0. If ECX[3:0] = 0001b, then this section has confidentiality, integrity, and replay protection. If ECX[3:0] = 0010b, then this section has confidentiality protection only. If ECX[3:0] = 0011b, then this section has confidentiality and integrity protection. All other encodings are reserved. ECX[11:04]: Reserved (enumerate 0). ECX[31:12]: EPC_SECTION_SIZE_31_12. Bits 31:12 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[19:00]: EPC_SECTION_SIZE_51_32. Bits 51:32 of the size of the corresponding EPC section within the Processor Reserved Memory. EDX[31:20]: Reserved.
Intel® Processor Trace Enumeration Main Leaf (Initial EAX Value = 14H, ECX = 0)		
14H		**NOTE:** Leaf 14H main leaf (ECX = 0).
	EAX	Bits 31-00: MAX_SUBLEAF. Reports the maximum sub-leaf supported in leaf 14H.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EBX	Bit 00: CR3_FILTER. If 1, indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed. Bits 01: CYC_ACC. If 1, indicates support of Configurable PSB and Cycle-Accurate Mode. Bits 02: IP_FILTER. If 1, indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset. Bits 03: MTC. If 1, indicates support of MTC timing packet and suppression of COFI-based packets. Bit 04: PTWRITE. If 1, indicates support of PTWRITE. Writes can set IA32_RTIT_CTL[12] (PTWEn) and IA32_RTIT_CTL[5] (FUPonPTW), and PTWRITE can generate packets. Bit 05: PWR_EVT_TRACE. If 1, indicates support of Power Event Trace. Writes can set IA32_RTIT_CTL[4] (PwrEvtEn), enabling Power Event Trace packet generation. Bit 06: PMI_PRESERVE. If 1, indicates support for PSB and PMI preservation. 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. Bit 07: EVENT_TRACE. If 1, writes can set IA32_RTIT_CTL[31] (EventEn), enabling Event Trace packet generation. Bit 08: TNT_DIS. If 1, writes can set IA32_RTIT_CTL[55] (DisTNT), disabling TNT packet generation. Bit 09: PTTT. If 1, Processor Trace Trigger Tracing (PTTT) is supported. Bits 31-10: Reserved.
	ECX	Bit 00: TOPAOUT. If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed. Bit 01: MENTRY. If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS. Bit 02: SNGL_RNG_OUT. If 1, indicates support of Single-Range Output scheme. Bit 03: TRACE_TRANSPORT_SUBSYSTEM. If 1, indicates support of output to Trace Transport subsystem. Bits 30-04: Reserved. Bit 31: LIP. If 1, generated packets which contain IP payloads have LIP values, which include the CS base component.
	EDX	Bits 31-00: Reserved.
Intel® Processor Trace Enumeration Sub-leaf (Initial EAX Value = 14H, ECX = 1)		
14H	EAX	Bits 02-00: RANGECNT. Number of configurable Address Ranges for filtering. Bits 07-03: Reserved. Bits 10-8: TRIGGER_CFG_CNT. Number of IA32_RTIT_TRIGGERx_CFG MSRs. The number of triggers supported is 4x this value. Bits 15-11: Reserved. Bits 31-16: MTC_RATE. Bitmap of supported MTC period encodings.
	EBX	Bits 15-00: CYC_THRESHOLDS. Bitmap of supported Cycle Threshold value encodings. Bits 31-16: PSB_RATE. Bitmap of supported Configurable PSB frequency encodings.
	ECX	Bit 00: ICNT. If 1, Trigger Action Attribution is supported. Bit 01: TRIGGER_PAUSE. If 1, the trigger actions TRACE_PAUSE and TRACE_RESUME are supported. Bits 14-02: Reserved. Bit 15: TRIGGER_DR_MATCH. If 1, trigger input DR match is supported. Bits 31-16: Reserved.
	EDX	Bits 31-00: Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
Time Stamp Counter and Core Crystal Clock Information Leaf (Initial EAX Value = 15H)		
15H		**NOTES:** If EBX[31:0] is 0, the TSC and "core crystal clock" ratio is not enumerated. EBX[31:0]/EAX[31:0] indicates the ratio of the TSC frequency and the core crystal clock frequency. If ECX is 0, the core crystal clock frequency is not enumerated. "TSC frequency" = "core crystal clock frequency" * EBX/EAX. The core crystal clock may differ from the reference clock, bus clock, or core clock frequencies.
	EAX	Bits 31-00: DENOMINATOR. An unsigned integer which is the denominator of the TSC/"core crystal clock" ratio.
	EBX	Bits 31-00: MUERATOR. An unsigned integer which is the numerator of the TSC/"core crystal clock" ratio.
	ECX	Bits 31-00: NOMINAL_ART_FREQUENCY. An unsigned integer which is the nominal frequency of the core crystal clock in Hz.
	EDX	Bits 31-00: Reserved = 0.
Processor Frequency Information Leaf (Initial EAX Value = 16H)		
16H	EAX	Bits 15-00: PROCESSOR_BASE_FREQUENCY. Processor Base Frequency (in MHz). Bits 31-16: Reserved =0.
	EBX	Bits 15-00: MAXIMUM_FREQUENCY. Maximum Frequency (in MHz). Bits 31-16: Reserved = 0.
	ECX	Bits 15-00: BUS_FREQUENCY. Bus (Reference) Frequency (in MHz). Bits 31-16: Reserved = 0.
	EDX	Reserved.
		NOTES: 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.
System-On-Chip Vendor Attribute Enumeration Main Leaf (Initial EAX Value = 17H, ECX = 0)		
17H		**NOTES:** Leaf 17H main leaf (ECX = 0). Leaf 17H output depends on the initial value in ECX. Leaf 17H sub-leaves 1 through 3 reports SOC Vendor Brand String. Leaf 17H is valid if MaxSOCID_Index >= 3. Leaf 17H sub-leaves 4 and above are reserved.
	EAX	Bits 31-00: MAX_SOCID_INDEX. Reports the maximum input value of supported sub-leaf in leaf 17H.
	EBX	Bits 15-00: SOC_VENDOR_ID. Bit 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. Bits 31-17: Reserved = 0.
	ECX	Bits 31-00: PROJECT_ID. A unique number an SOC vendor assigns to its SOC projects.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	EDX	Bits 31-00: STEPPING_ID. A unique number within an SOC project that an SOC vendor assigns.
System-On-Chip Vendor Attribute Enumeration Sub-leaf (Initial EAX Value = 17H, ECX = 1..3)		
17H	EAX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	EBX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	ECX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
	EDX	Bit 31-00: SOC Vendor Brand String. UTF-8 encoded string.
		NOTES: Leaf 17H output depends on the initial value in ECX. 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 of EAX:EBX:ECX:EDX and from the sub-leaf 1 fragment towards sub-leaf 3.
System-On-Chip Vendor Attribute Enumeration Sub-leaves (Initial EAX Value = 17H, ECX > MaxSOCID_Index)		
17H		**NOTE:** Leaf 17H output depends on the initial value in ECX.
	EAX	Bits 31-00: Reserved.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Deterministic Address Translation Parameters Main Leaf (Initial EAX Value = 18H, ECX = 0)		
18H		**NOTES:** Each sub-leaf enumerates a different address translations structure. 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. * 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. ** Add one to the return value to get the result.
	EAX	Bits 31-00: MAX_SUBLEAF. Reports the maximum input value of supported sub-leaf in leaf 18H.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Deterministic Address Translation Parameters Sub-leaf (Initial EAX Value = 18H, ECX ≥ 1)		

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
18H		**NOTES:** 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. * 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. ** Add one to the return value to get the result.
	EAX	Bits 31-00: Reserved.
	EBX	Bit 00: 4KB_ENTRIES. 4K page size entries supported by this structure. Bit 01: 2MB_ENTRIES. 2MB page size entries supported by this structure. Bit 02: 4MB_ENTRIES. 4MB page size entries supported by this structure. Bit 03: 1GB_ENTRIES. 1 GB page size entries supported by this structure. Bits 07-04: Reserved. Bits 10-08: PARTITIONING. (0: Soft partitioning between the logical processors sharing this structure.) Bits 15-11: Reserved. Bits 31-16: NUM_WAYS. W = Ways of associativity.
	ECX	Bits 31-00: NUM_SETS. S = Number of Sets.
	EDX	Bits 04-00: TYPE. Translation cache type field. 0000b: Null (indicates this sub-leaf is not valid). 0001b: Data TLB. 0010b: Instruction TLB. 0011b: Unified TLB. All other encodings are reserved. Bits 07-05: LEVEL_NUM. Translation cache level (starts at 1). Bit 08: FULLY_ASSOC. Fully associative structure. Bits 13-09: Reserved. Bits 25-14: MAX_LP_ADDRESSABLE_IDS. Maximum number of addressable IDs for logical processors sharing this translation cache.** Bits 31-26: Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
Key Locker Leaf (Initial EAX Value = 19H)		
19H	EAX	Bit 00: CPL0_RESTRICT. Key Locker restriction of CPL0-only supported. Bit 01: NO_ENCRYPT_RESTRICT. Key Locker restriction of no-encrypt supported. Bit 02: NO_DECRYPT_RESTRICT. Key Locker restriction of no-decrypt supported. Bits 31-03: Reserved.
	EBX	Bit 00: AESKLE. If 1, the AES Key Locker instructions are fully enabled. Bit 01: Reserved. Bit 02: AES_WIDE. If 1, the AES wide Key Locker instructions are supported. Bit 03: Reserved. Bit 04: IWKEYBACKUP. If 1, the platform supports the Key Locker MSRs and backing up the internal wrapping key. Bits 31-05: Reserved.
	ECX	Bit 00: NOBACKUP. If 1, the NoBackup parameter to LOADIWKEY is supported. Bit 01: RAND_IWKEY. If 1, KeySource encoding of 1 (randomization of the internal wrapping key) is supported. Bits 31- 02: Reserved.
	EDX	Reserved.
Native Model ID Enumeration Leaf (Initial EAX Value = 1AH, ECX = 0)		
1AH		**NOTE:** This leaf exists on all hybrid parts, however this leaf is not only available on hybrid parts. The following algorithm is used for detection of this leaf: If CPUID.0.MAXLEAF $\geq$ 1AH and CPUID.1A.EAX $\neq$ 0, then the leaf exists.
	EAX	Enumerates the native model ID and core type.* Bits 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 leaf 01H, and does not identify the SOC. Bits 31-24: CORE_TYPE. 10H: Reserved. 20H: Intel Atom®. 30H: Reserved. 40H: Intel® Core™. **NOTE:** * 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.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
PCONFIG Information Sub-leaf (Initial EAX Value = 1BH, ECX $\geq$ 0)		
1BH		For details on this sub-leaf, see “INPUT EAX = 1BH: Returns PCONFIG Information” on page 1-48. **NOTE:** Leaf 1BH is supported if CPUID.07H.00H:EDX[18] = 1.
Last Branch Records Information Leaf (Initial EAX Value = 1CH)		
1CH		**NOTES:** This leaf pertains to the architectural feature. For leaf 01CH, CPUID will ignore the ECX value.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	EAX	Bits 07 - 00: LBR_DEPTH_VALUES. For each bit n set in this field, the IA32_LBR_DEPTH.DEPTH value 8*(n+1) is supported. Bits 29 - 08: Reserved. Bit 30: DEEP_C_STATE_RESET. If set, indicates that LBRs may be cleared on an MWAIT that requests a C-state numerically greater than C1. Bit 31: IP_VALUES_CONTAIN_LIP. If set, LBR IP values contain LIP. If clear, IP values contain Effective IP.
	EBX	Bit 00: CPL_FILTERING. If set, the processor supports setting IA32_LBR_CTL[2:1] to a non-zero value. Bit 01: BRANCH_FILTERING. If set, the processor supports setting IA32_LBR_CTL[22:16] to a non-zero value. Bit 02: CALL_STACK_MODE. If set, the processor supports setting IA32_LBR_CTL[3] to 1. Bits 31-03: Reserved.
	ECX	Bit 00: MISPREDICT_BIT. IA32_LBR_x_INFO[63] holds indication of branch misprediction (MISPRED). Bit 01: TIMED_LBRS. 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). Bit 02: BRANCH_TYPE_FIELD_SUPPORTED. IA32_LBR_INFO_x[59:56] holds indication of the recorded operation's branch type (BR_TYPE). Bits 15-03: Reserved. Bits 19-16: EVENT_LOGGING_BITMAP. Bits 31-20: Reserved.
	EDX	Bits 31 - 00: Reserved.
Tile Information Main Leaf (Initial EAX Value = 1DH, ECX = 0)		
1DH		**NOTES:** For sub-leaves of 1DH, they are indexed by the palette id. Leaf 1DH sub-leaves 2 and above are reserved.
	EAX	Bits 31-00: MAX_PALETTE. Highest numbered palette sub-leaf. Value = 1.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Tile Palette 1 Sub-leaf (Initial EAX Value = 1DH, ECX = 1)		
1DH	EAX	Bits 15-00: TOTAL_TILE_BYTES. Palette 1 total_tile_bytes. Value = 8192. Bits 31-16: BYTES_PER_TILE. Palette 1 bytes_per_tile. Value = 1024.
	EBX	Bits 15-00: BYTES_PER_ROW. Palette 1 bytes_per_row. Value = 64. Bits 31-16: MAX_NAMES. Palette 1 max_names (number of tile registers). Value = 8.
	ECX	Bits 15-00: MAX_ROWS. Palette 1 max_rows. Value = 16. Bits 31-16: Reserved.
	EDX	Bits 31-00: Reserved.
TMUL Information Main Leaf (Initial EAX Value = 1EH, ECX = 0)		
1EH		**NOTES:** Leaf 1EH sub-leaf 2 and above are reserved.
	EAX	Bits 31-00: Reports the maximum number of sub-leaves that are supported in leaf 1EH.
	EBX	Bits 07-00: TMUL_MAXK. tmul_maxk (rows or columns). Value = 16. Bits 23-08: TMUL_MAXN. tmul_maxn (column bytes). Value = 64. Bits 31-24: Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value	Information Provided about the Processor	
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
TMUL Information Sub-leaf (Initial EAX Value = 1EH, ECX =1)		
1EH		**NOTE:** Leaf 1EH sub-leaf 2 and above are reserved. AMX feature enumerations are moving to a single location under CPUID.1EH.01H for easier enumeration by software and support of VMM Passthrough as described in the Intel® CPUID Passthrough Virtualization Considerations Technical Paper. Existing feature enumerations prior to this architectural change (i.e., CPUID.07H.01:EAX.AMX_FP16[21]) will be enumerated in both CPUID.1EH.01H:EAX[3] and CPUID.07H.01:EAX[21]. Implementations using a new location in CPUID.1EH.01H:EAX will need to use its corresponding enumeration bit in CPUID.07H.01 when CPUID.1EH.00H:EAX[31:0] = 0.
	EAX	Bit 00: AMX_INT8. If 1, the processor supports tile computational operations on 8-bit integers. Bit 01: AMX_BF16. If 1, the processor supports tile computational operations on bfloat16 numbers. Bit 02: AMX_COMPLEX. If 1, supports the AMX-COMPLEX instructions. Bit 03: AMX_FP16. If 1, the processor supports tile computational operations on FP16 numbers. Bit 04: AMX_FP8. If 1, supports Intel AMX computations for the FP8 data type. Bit 06-05: Reserved. Bit 07: AMX_AVX512. If 1, supports the AMX-AVX512 instructions. Bit 08: AMX_MOVRS. If 1, supports the AMX-MOVRS instructions. Bits 31-09: Reserved.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
V2 Extended Topology Enumeration Leaf (Initial EAX Value = 1FH, *ECX* ≥ *0*)		
1FH		**NOTES:** CPUID leaf 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. The sub-leaves of CPUID leaf 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 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EAX	Bits 04-00: 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. Bits 31-05: Reserved.
	EBX	Bits 15-00: 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 symmetric topology 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, software must not use this field to enumerate processor topology*. Bits 31-16: Reserved.
	ECX	Bits 07-00: LEVEL_NUM. The input ECX sub-leaf index. Bits 15-08: DOMAIN_TYPE. This field provides an identification value which indicates the domain as shown below. Although domains are ordered, as also shown below, 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.) Hierarchy — Domain — Domain Type Identification Value Lowest — Logical Processor — 1 ... — Core — 2 ... — Module — 3 ... — Tile — 4 ... — Die — 5 ... — DieGrp — 6 Highest — Package/Socket — (implied) (Note that enumeration values of 0 and 7-255 are reserved.) Bits 31-16: Reserved.
	EDX	Bits 31-00: X2APIC_ID. x2APIC ID of the current logical processor. It is always valid and does not vary with the sub-leaf index in ECX. **NOTES:** * 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.
Processor History Reset Sub-leaf (Initial EAX Value = 20H, ECX = 0)		
20H	EAX	MAX_SUBLEAF. Reports the maximum number of sub-leaves that are supported in leaf 20H.
	EBX	Indicates which bits may be set in the IA32_HRESET_ENABLE MSR to enable enhanced hardware feedback interface history. Bit 00: 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 EHFI history. Bits 31-01: Reserved.
	ECX	Reserved.
	EDX	Reserved.
Architectural Performance Monitoring Extended Main Leaf (Initial EAX Value = 23H, ECX = 0)		

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
23H		**NOTE:** Output depends on ECX input value.
	EAX	Bits 31-00: 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.)
	EBX	Bit 00: UNITMASK2. If set, the processor supports the UnitMask2 field in the IA32_PERFEVTSELx MSRs. Bit 01: EQ. If set, the processor supports the equal flag in the IA32_PERFEVTSELx MSRs. Bit 02: RDPMC_USR_DISABLE. Bits 31-03: Reserved.
	ECX	Bits 07-00: SLOTS_PER_CYC. 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. Bits 31-08: Reserved.
	EDX	Bits 31-00: Reserved.
Architectural Performance Monitoring Extended Sub-Leaf (Initial EAX Value = 23H, ECX = 1)		
23H	EAX	Bits 31-00: GP_COUNTERS. General-purpose counters bitmap. For each bit *n* set in this field, the processor supports general-purpose performance monitoring counter *n*.
	EBX	Bits 31-00: FIXED_COUNTERS. Fixed-function counters bitmap. For each bit *m* set in this field, the processor supports fixed-function performance monitoring counter *m*.[2]
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Architectural Performance Monitoring Extended Sub-Leaf (Initial EAX Value = 23H, ECX = 2)		
23H	EAX	Bits 31-00: ACR_GP_RELOAD. Bitmap of Auto Counter Reload (ACR) general-purpose counters that can be reloaded. For each bit *n* set in this field, the processor supports ACR for general-purpose performance monitoring counter *n*.
	EBX	Bits 31-00: ACR_FIXED_RELOAD. Bitmap of Auto Counter Reload (ACR) fixed-function counters that can be reloaded. For each bit *m* set in this field, the processor supports ACR for fixed-function performance monitoring counter *m*.
	ECX	Bits 31-00: ACR_GP_TRIGGER. Bitmap of Auto Counter Reload (ACR) general-purpose 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.
	EDX	Bits 31-00: ACR_FIXED_TRIGGER. Bitmap of Auto Counter Reload (ACR) fixed-function 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.
Architectural Performance Monitoring Extended Sub-Leaf (Initial EAX Value = 23H, ECX = 3)		
23H		**NOTE:** Architectural Performance Monitoring Events Bitmap. For each bit *n* set in this field, the processor supports Architectural Performance Monitoring Event of index *n*.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EAX	Bit 00: CORE_CYC. Core cycles. Bit 01: INSTR_RET. Instructions retired. Bit 02: REF_CYC. Reference cycles. Bit 03: LLC_REF. Last level cache references. Bit 04: LLLC_MISSES. ast level cache misses. Bit 05: BR_INSTR_RET. Branch instructions retired. Bit 06: BR_MISPRED_RET. Branch mispredicts retired. Bit 07: SLOTS. Topdown slots. Bit 08: BACKEND. Topdown backend bound. Bit 09: BADSPEC. Topdown bad speculation. Bit 10: FRONTEND. Topdown frontend bound. Bit 11: RETIRING. Topdown retiring. Bit 12: LBR_INSERTS. LBR inserts. Bits 31-13: Reserved.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Architectural Performance Monitoring Extended Sub-Leaf (Initial EAX Value = 23H, ECX = 4)		
23H	EAX	Bits 31-00: Reserved.
	EBX	Bits 02-00: Reserved. Bit 03: ALLOW_IN_RECORD. If 1, indicates that the ALLOW_IN_RECORD bit is available in the IA32_PMC_GPn_CFG_C and IA32_PMC_FXm_CFG_C MSRs. Bit 04: CNTR_GP. If 1, indicates that counters group sub-group general-purpose counters is available. Bit 05: CNTR_FIXED. If 1, indicates that counters group sub-group fixed-function counters is available. Bit 06: CNTR_METRICS. If 1, indicates that counters group sub-group performance metrics is available. Bit 07: Reserved. Bits 09-08: LBR. LBR group and both bits [41:40] are available. Bits 15-10: Reserved. Bits 23-16: XER. XER group bits [50:49] and bits [55:53] are available. Bits 28-24: Reserved. Bit 29: GPR. If 1, the GPR group is available. Bit 30: AUX. If 1, the AUX group is available. Bit 31: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Architectural Performance Monitoring Extended Sub-Leaf (Initial EAX Value = 23H, ECX = 5)		
23H	EAX	Bits 31-00: GP_PEBS. General-purpose counters support Architectural PEBS. Bit vector of general-purpose counters for which the Architectural PEBS mechanism is available (bit n == GP counter #n). If EAX[n] == 1, then the IA32_PMC_GPn_CFG_C MSR is available, and PEBS is supported on that counter; the PEBS_EN[63] field can be set; and the RELOAD[31:0] field can be set. Note that CPUID.(EAX=23H, ECX=04H):EBX governs which adaptive group bits can be set.
	EBX	Bits 31-00: GP_PDIST. General-purpose counters for which PEBS supports PDIST.
	ECX	Bits 31-00: FIXED_PEBS. Fixed-function counters support Architectural PEBS. Bit vector of fixed-function counters for which the Architectural PEBS mechanism is available. If ECX[x] == 1, then the IA32_PMC_FXm_CFG_C MSR is available, and PEBS is supported; the PEBS_EN[63] field can be set; and the RELOAD[31:0] field can be set. Note that CPUID.(EAX=23H, ECX=04H):EBX governs which adaptive group bits can be set.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EDX	Bits 31-00: FIXED_PDIST. Fixed-function counters for which PEBS supports PDIST.
Converged Vector ISA Main Leaf (Initial EAX Value = 24H, ECX = 0)		
24H		**NOTE:** Output depends on ECX input value.
	EAX	Bits 31-00: MAX_SUBLEAF. Reports the maximum number sub-leaves that are supported in leaf 24H.
	EBX	Bits 07-00: VECTOR_ISA_VERSION. Reports the Intel AVX10 Converged Vector ISA version. Bits 15-08: Reserved. Bit 18-16: Reserved at 111.[3] Bits 31-19: Reserved.
	ECX	Bits 31-00: Reserved.
	EDX	Bits 31-00: Reserved.
Intel® Resource Director Technology Monitoring Asymmetric Enumeration Sub-leaf (Initial EAX Value = 27H, ECX = 0)		
27H		**NOTES:** Leaf 27H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource type starting at bit position 1 of EDX. Leaf 27H can return different information on each logical processor and software must account for this.
	EAX	Reserved.
	EBX	Bits 31-0: MAX_RMID. Maximum range (zero-based) of RMID within this physical processor of all types.
	ECX	Reserved.
	EDX	Bit 00: Reserved. Bit 01: L3_MON. Supports L3 Cache Intel RDT Monitoring if 1. Bits 31-02: Reserved.
L3 Cache Intel® RDT Monitoring Capability Asymmetric Enumeration Sub-leaf (Initial EAX Value = 27H, ECX = 1)		
27H		**NOTE:** Leaf 27H output depends on the initial value in ECX. Leaf 27H can return different information on each logical processor and software must account for this.
	EAX	No bits set: 24-bit counters. Bits 07-00: CTR_WIDTH. Encode counter width offset from 24b: 0x0 = 24-bit counters. 0x1 = 25-bit counters. ... 0x25 = 61-bit counters. Bit 08: RDT_M_OVF. If 1, indicates the presence of an overflow bit in the IA32_QM_CTR MSR (bit 61). Bit 09: IO_RDT_CMT. If 1, indicates the presence of non-CPU agent Intel RDT CMT support. Bit 10: IO_RDT_MBM. If 1, indicates the presence of non-CPU agent Intel RDT MBM support. Bits 31-11: Reserved.
	EBX	Bits 31-00: CONV_FACTOR. Conversion factor from reported IA32_QM_CTR value to occupancy metric (bytes) and Memory Bandwidth Monitoring (MBM) metrics.
	ECX	Bits 31-00: MAX_RMID_L3. Maximum range (zero-based) of RMID of this resource type.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	EDX	Bit 00: CMT_L3_OCCUP. Supports L3 occupancy monitoring if 1. Bit 01: MBM_L3_TOTAL. Supports L3 Total Bandwidth monitoring if 1. Bit 02: MBM_L3_LOCAL. Supports L3 Local Bandwidth monitoring if 1. Bits 31-03: Reserved.
Intel® Resource Director Technology Allocation Asymmetric Enumeration Sub-leaf (Initial EAX Value = 28H, ECX = 0)		
28H		**NOTES:** Leaf 28H output depends on the initial value in ECX. Sub-leaf index 0 reports valid resource identification (ResID) starting at bit position 1 of EBX. Leaf 28H can return different information on each logical processor and software must account for this.
	EAX	Reserved.
	EBX	Bit 00: Reserved. Bit 01: CAT_L3. Supports L3 Cache Allocation Technology if 1. Bit 02: CAT_L2. Supports L2 Cache Allocation Technology if 1. Bit 03: MBA. Supports Memory Bandwidth Allocation if 1. Bit 04: Reserved. Bit 05: Supports Cache Bandwidth Allocation if 1. Bits 31-06: Reserved.
	ECX	Reserved.
	EDX	Reserved.
L3 Cache Intel® RDT Allocation Asymmetric Enumeration Sub-leaf (Initial EAX Value = 28H, ECX = ResID = 1)		
28H		**NOTE:** Leaf 28H output depends on the initial value in ECX. Leaf 28H can return different information on each logical processor and software must account for this.
	EAX	Bits 04-00: CAT_L3_BITMASK_LENGTH. Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31-05: Reserved.
	EBX	Bits 31-00: CAT_L3_CONTENTION. Bit-granular map of isolation/contention of allocation units.
	ECX	Bit 00: Reserved. Bit 01: CAT_L3_NONCPU If 1, indicates L3 CAT for non-CPU agents is supported. Bit 02: CAT_L3_CDP. If 1, indicates L3 Code and Data Prioritization Technology is supported. Bit 03: CAT_L3_NONCONTIG. 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 31-04: Reserved.
	EDX	Bits 15-00: CAT_L3_MAX_CLOS. Highest COS number supported for this ResID. Bits 31-16: Reserved.
L2 Cache Intel® RDT Allocation Asymmetric Enumeration Sub-leaf (Initial EAX Value = 28H, ECX = ResID = 2)		
28H		**NOTE:** Leaf 28H output depends on the initial value in ECX. Leaf 28H can return different information on each logical processor and software must account for this.
	EAX	Bits 04-00: CAT_L2_BITMASK_LENGTH. Length of the capacity bit mask for the corresponding ResID. Add one to the return value to get the result. Bits 31-05: Reserved.
	EBX	Bits 31-00: CAT_L2_CONTENTION. Bit-granular map of isolation/contention of allocation units.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
	ECX	Bits 01-00: Reserved. Bit 02: CAT_L2_CDP. If 1, indicates L2 Code and Data Prioritization Technology is supported. Bit 03: CAT_L2_NONCONTIG. If 1, indicates non-contiguous capacity bitmask is supported. The bits that are set in the various IA32_L2_MASK_n registers do not have to be contiguous. Bits 31-04: Reserved.
	EDX	Bits 15-00: CAT_L2_MAX_CLOS. Highest COS number supported for this ResID. Bits 31-16: Reserved.
Memory Bandwidth Allocation Asymmetric Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID = 3)		
28H		**NOTE:** Leaf 28H output depends on the initial value in ECX. Leaf 28H can return different information on each logical processor and software must account for this.
	EAX	Bits 11-00: MBA_MAX. Reports the maximum MBA throttling value supported for the corresponding ResID. Add one to the return value to get the result. Bits 31-12: Reserved.
	EBX	Bits 31-00: Reserved.
	ECX	Bit 00: PER_THREAD_MBA. Per-thread MBA controls are supported. Bit 01: Reserved. Bit 02: MBA_LINEAR. Reports whether the response of the delay values is linear. Bits 31-03: Reserved.
	EDX	Bits 15-00: MBA_MAX_CLOS. Highest COS number supported for this ResID. Bits 31-16: Reserved.
Cache Bandwidth Allocation Asymmetric Enumeration Sub-leaf (Initial EAX Value = 10H, ECX = ResID = 5)		
28H		**NOTE:** Leaf 28H output depends on the initial value in ECX. Leaf 28H can return different information on each logical processor and software must account for this.
	EAX	Bits 07-00: CBA_MAX_LEVELS. Reports the maximum core throttling level supported for the corresponding ResID. Add one to the return value to get the number of throttling levels supported. Bits 11-08: BW_SCOPE. If 1, indicates the logical processor scope of the IA32_QoS_Core_BW_Thrtl_n MSRs. Other values are reserved. Bits 31-12: Reserved.
	EBX	Bits 31-00: Reserved.
	ECX	Bits 02-00: Reserved. Bit 03: CBA_LINEAR. If 1, the response of the bandwidth control is approximately linear. If 0, the response of the bandwidth control is non-linear. Bits 31-04: Reserved.
	EDX	Bits 15-00: CBA_MAX_CLOS. Highest Class of Service (COS) number supported for this ResID. Bits 31-16: Reserved.
Unimplemented CPUID Leaf Functions		
21H		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is 21H. If the value returned by CPUID.0:EAX (the maximum input value for basic CPUID information) is at least 21H, 0 is returned in the registers EAX, EBX, ECX, and EDX. Otherwise, the data for the highest basic information leaf is returned.
40000000H — 4FFFFFFFH		Invalid. No existing or future CPU will return processor identification or feature information if the initial EAX value is in the range 40000000H to 4FFFFFFFH.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
Extended Function CPUID Information		
80000000H	EAX	MAX_EXTENDED_LEAF. Maximum Input Value for Extended Function CPUID Information.
	EBX	Reserved.
	ECX	Reserved.
	EDX	Reserved.
80000001H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bit 00: LAHF_SAHF_64. LAHF/SAHF available in 64-bit mode. Bits 04-01: Reserved. Bit 05: LZCNT. LZCNT available. Bits 07-06: Reserved. Bit 08: PREFETCHW. Bits 31-09: Reserved.
	EDX	Bits 10-00: Reserved. Bit 11: SYSCALL_SYSRET_64. SYSCALL/SYSRET available (when in 64-bit mode). Bits 19-12: Reserved. Bit 20: EXECUTE_DIS. Execute Disable Bit available. Bits 25-21: Reserved. Bit 26: PAGE_1GB. 1-GByte pages are available if 1. Bit 27: RDTSCP. RDTSCP and IA32_TSC_AUX are available if 1. Bit 28: Reserved. Bit 29: INTEL64. Intel® 64 Architecture available if 1. Bits 31-30: Reserved.
80000002H	EAX EBX ECX EDX	Processor Brand String. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000003H	EAX EBX ECX EDX	Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000004H	EAX EBX ECX EDX	Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued. Processor Brand String Continued.
80000005H	EAX EBX ECX EDX	Reserved. Reserved. Reserved. Reserved.
80000006H	EAX	Reserved.
	EBX	Reserved.
	ECX	Bits 07-00: L2_LINE_SIZE. Cache Line size in bytes. Bits 11-08: Reserved. Bits 15-12: L2_ASSOC. L2 Associativity field.* Bits 31-16: L2_SIZE. Cache size in 1K units.
	EDX	Reserved.

Table 1-3. Information Returned by CPUID Instruction (Continued)

Initial EAX Value		Information Provided about the Processor
		NOTES: * L2 associativity field encodings: 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 04H, sub-leaf 2**0FH - Fully associative ** CPUID leaf 04H provides details of deterministic cache parameters, including the L2 cache in sub-leaf 2.
80000007H	EAX	Reserved = 0.
	EBX	Reserved = 0.
	ECX	Reserved = 0.
	EDX	Bits 07-00: Reserved. Bit 08: TSC_INVARIANT. Invariant TSC available if 1. Bits 31-09: Reserved.
80000008H	EAX	Virtual/Physical Address Size Bits 07-00: PHYS_ADDR_SIZE.* Bits 15-08: LIN_ADDR_SIZE. Number of linear-address bits. Bits 23-16: GUEST_PHYS_ADDR_SIZE. Number of guest-physical-address bits (for software operating in a virtual machine). If this field is zero, PHYS_ADDR_SIZE should be used. Intel processors return zero for this field. Software emulating CPUID may return a different value. Bits 31-24: Reserved.
	EBX	Bits 08-00: Reserved. Bit 09: WBNOINVD. WBNOINVD is available if 1. Bits 31-10: Reserved.
	ECX	Reserved.
	EDX	Reserved.
		NOTES: * 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].

NOTES:

1. The logical processor might not be available to report information due to user settings in the operating system or processor based licensing by software.
2. The valid range of fixed-function counters is 0 through 15.
3. Earlier versions of this specification documented these bits as enumerating support for different vector lengths. All processors supporting Intel® AVX10 will include support for all vector lengths.

INPUT EAX = 0H: Returns CPUID's Highest Value for Basic Processor Information and the Vendor Identification String

When CPUID executes with EAX set to 0H, the processor returns the highest value the CPUID recognizes for returning basic processor information. The value is returned in the EAX register and is processor specific.

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 4 bits of BL *)
EDX := 49656e69h (* "inel", with i in the low 4 bits of DL *)
ECX := 6c65746eh (* "ntel", with n in the low 4 bits of CL *)

INPUT EAX = 80000000H: Returns CPUID's Highest Value for Extended Processor Information

When CPUID executes with EAX set to 0H, the processor returns the highest value the processor recognizes for returning extended processor information. The value is returned in the EAX register and is processor specific.

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 12, "Processor Management and Initialization," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.

INPUT EAX = 01H: Returns Model, Family, Stepping Information

When CPUID executes with EAX set to 01H, version information is returned in EAX (see Figure 1-1). 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 1-4 for available processor type values. Stepping IDs are provided as needed.

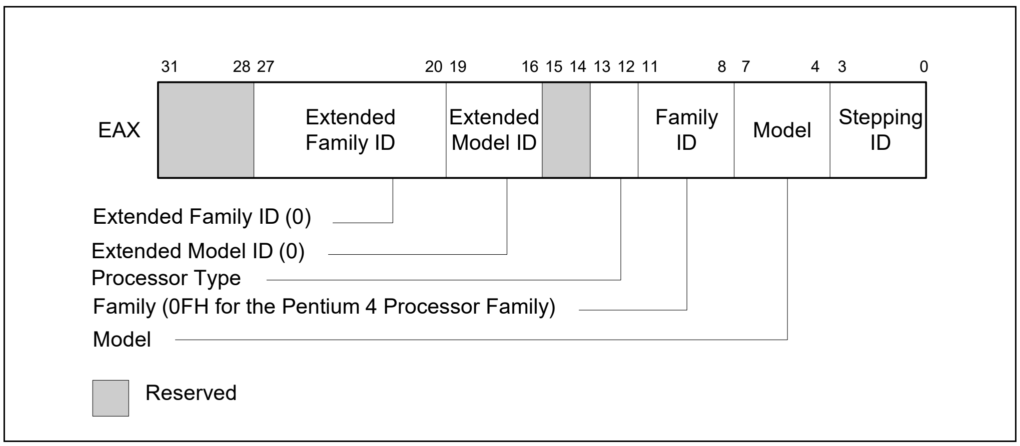


Figure 1-1. Version Information Returned by CPUID in EAX

Table 1-4. 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 "Caching Translation Information" in Chapter 5, “Paging,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A, and Chapter 20, “Input/Output,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1, 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 Displayed_Family = Family_ID;
    ELSE Displayed_Family = Extended_Family_ID + Family_ID;
FI;
(* Show Display_Family as HEX field. *)
```

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 Displayed_Model = (Extended_Model_ID << 4) + Model_ID;
    (* Right justify and zero-extend 4-bit field; display Model_ID as HEX field.*)
    ELSE Displayed_Model = Model_ID;
FI;
(* Show Display_Model as HEX field. *)
```

INPUT EAX = 01H: Returns Additional Information in EBX

When CPUID executes with EAX set to 01H, additional information is returned to the EBX register:

- Brand index (low byte of EBX) — 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 later in this section.
- CLFLUSH instruction cache line size (second byte of EBX) — this number indicates the size of the cache line flushed with CLFLUSH instruction in 8-byte increments. This field was introduced in the Pentium 4 processor.
- Local APIC ID (high byte of EBX) — 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.

INPUT EAX = 01H: Returns Feature Information in ECX and EDX

When CPUID executes with EAX set to 01H, feature information is returned in ECX and EDX.

- Figure 1-2 and Table 1-5 show encodings for ECX.
- Figure 1-3 and Table 1-6 show encodings for EDX.

For all feature flags, a 1 indicates that the feature is supported. Use Intel to properly interpret feature flags.

NOTE

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.

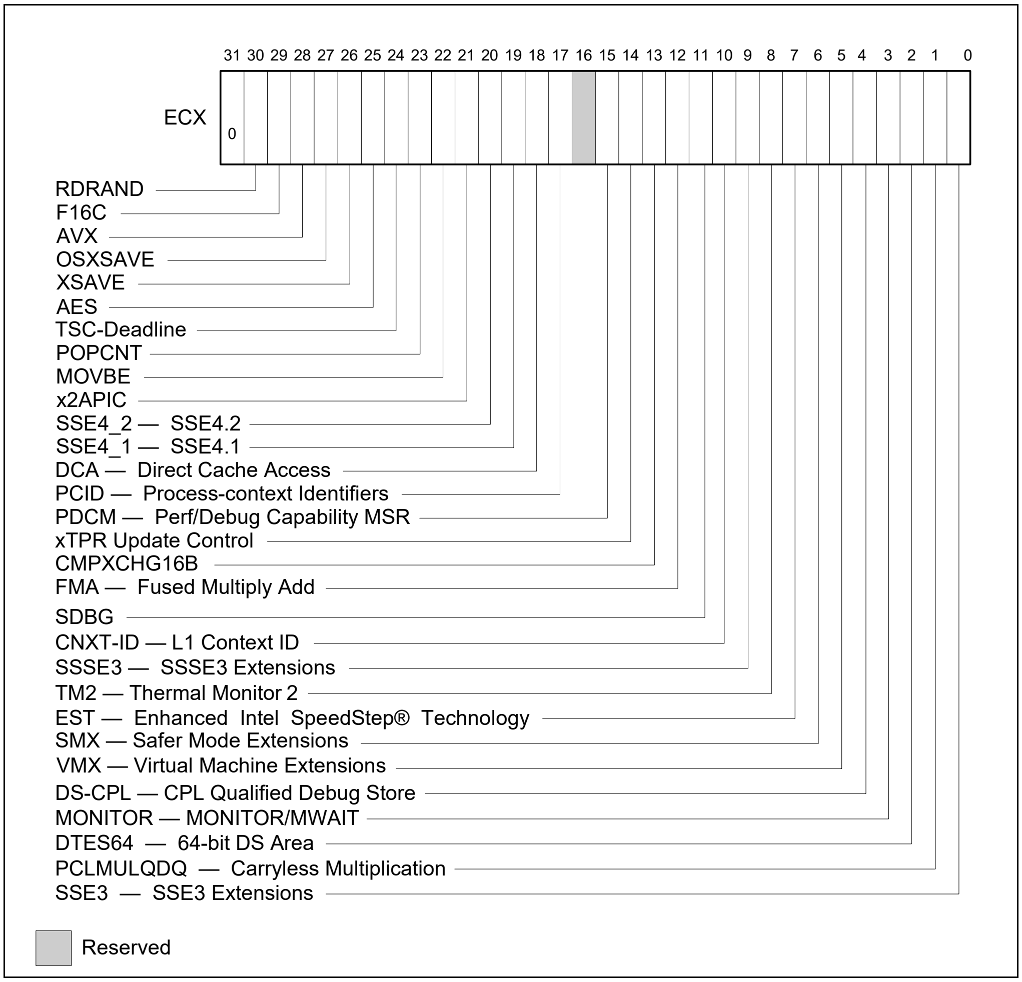


Figure 1-2. Feature Information Returned in the ECX Register

Table 1-5. Feature Information Returned in the ECX Register

Bit #	Mnemonic	Description
0	SSE3	**Intel® Streaming SIMD Extensions 3 (Intel® SSE3).** A value of 1 indicates the processor supports this technology.
1	PCLMULQDQ	A value of 1 indicates the processor supports the PCLMULQDQ instruction.
2	DTES64	**64-bit DS Area.** A value of 1 indicates the processor supports DS area using 64-bit layout.
3	MONITOR	**MONITOR/MWAIT.** A value of 1 indicates the processor supports this feature.
4	DS-CPL	**CPL Qualified Debug Store.** A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	**Virtual Machine Extensions.** A value of 1 indicates that the processor supports this technology.
6	SMX	**Safer Mode Extensions.** A value of 1 indicates that the processor supports this technology. See Chapter 7, "Safer Mode Extensions Reference," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2D.
7	EST	**Enhanced Intel SpeedStep® Technology.** A value of 1 indicates that the processor supports this technology.
8	TM2	**Thermal Monitor 2.** A value of 1 indicates whether the processor supports this technology.

Table 1-5. Feature Information Returned in the ECX Register (Continued)

Bit #	Mnemonic	Description
9	SSSE3	A value of 1 indicates the presence of the Supplemental Streaming SIMD Extensions 3 (SSSE3). A value of 0 indicates the instruction extensions are not present in the processor.
10	CNXT-ID	**L1 Context ID.** A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLE MSR Bit 24 (L1 Data Cache Context Mode) for details.
11	SDBG	A value of 1 indicates the processor supports IA32_DEBUG_INTERFACE MSR for silicon debug.
12	FMA	A value of 1 indicates the processor supports FMA extensions using YMM state.
13	CMPXCHG16B	**CMPXCHG16B Available.** A value of 1 indicates that the feature is available.
14	xTPR Update Control	**xTPR Update Control.** A value of 1 indicates that the processor supports changing IA32_MISC_ENABLES[bit 23].
15	PDCM	**PerfMon and Debug Capability.** A value of 1 indicates the processor supports the performance and debug feature indication MSR IA32_PERF_CAPABILITIES.
16	Reserved	Reserved.
17	PCID	**Process-context identifiers.** A value of 1 indicates that the processor supports PCIDs and that software may set CR4.PCIDE to 1.
18	DCA	A value of 1 indicates the processor supports the ability to prefetch data from a memory mapped device.
19	SSE4.1	A value of 1 indicates that the processor supports SSE4.1.
20	SSE4.2	A value of 1 indicates that the processor supports SSE4.2.
21	x2APIC	A value of 1 indicates that the processor supports x2APIC feature.
22	MOVBE	A value of 1 indicates that the processor supports MOVBE instruction.
23	POPCNT	A value of 1 indicates that the processor supports the POPCNT instruction.
24	TSC-Deadline	A value of 1 indicates that the processor's local APIC timer supports one-shot operation using a TSC deadline value.
25	AES	A value of 1 indicates that the processor supports the AESNI instruction extensions.
26	XSAVE	A value of 1 indicates that the processor supports the XSAVE/XRSTOR processor extended states feature, the XSETBV/XGETBV instructions, and XCR0.
27	OSXSAVE	A value of 1 indicates that 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.
28	AVX	A value of 1 indicates that processor supports AVX instructions operating on 256-bit YMM state, and three-operand encoding of 256-bit and 128-bit SIMD instructions.
29	F16C	A value of 1 indicates that processor supports 16-bit floating-point conversion instructions.
30	RDRAND	A value of 1 indicates that processor supports RDRAND instruction.
31	Not Used	Always return 0.

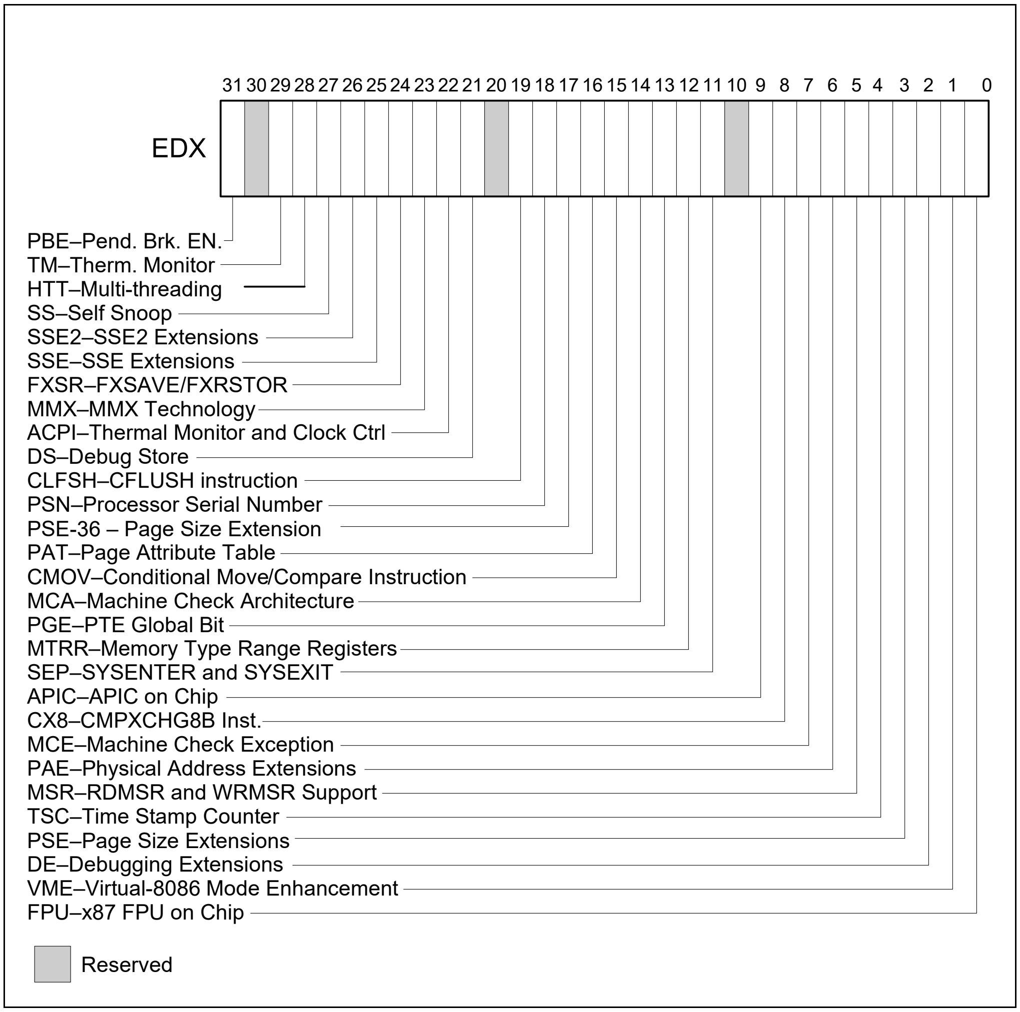


Figure 1-3. Feature Information Returned in the EDX Register

Table 1-6. More on Feature Information Returned in the EDX Register

Bit #	Mnemonic	Description
0	FPU	**Floating-point Unit On-Chip.** The processor contains an x87 FPU.
1	VME	**Virtual 8086 Mode Enhancements.** 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.
2	DE	**Debugging Extensions.** Support for I/O breakpoints, including CR4.DE for controlling the feature, and optional trapping of accesses to DR4 and DR5.
3	PSE	**Page Size Extension.** Large pages of size 4 MByte are supported, 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.
4	TSC	**Time Stamp Counter.** The RDTSC instruction is supported, including CR4.TSD for controlling privilege.
5	MSR	**Model Specific Registers RDMSR and WRMSR Instructions.** The RDMSR and WRMSR instructions are supported. Some of the MSRs are implementation dependent.

Table 1-6. More on Feature Information Returned in the EDX Register (Continued)

Bit #	Mnemonic	Description
6	PAE	**Physical Address Extension.** Physical addresses greater than 32 bits are supported: extended page table entry formats, an extra level in the page translation tables is defined, 2-MByte pages are supported instead of 4 Mbyte pages if PAE bit is 1. The actual number of address bits beyond 32 is not defined, and is implementation specific.
7	MCE	**Machine Check Exception.** Exception 18 is defined for Machine Checks, including CR4.MCE for controlling the feature. This feature does not define the model-specific 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.
8	CX8	**CMPXCHG8B Instruction.** The compare-and-exchange 8 bytes (64 bits) instruction is supported (implicitly locked and atomic).
9	APIC	**APIC On-Chip.** 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).
10	Reserved	Reserved.
11	SEP	**SYSENTER and SYSEXIT Instructions.** The SYSENTER and SYSEXIT and associated MSRs are supported.
12	MTRR	**Memory Type Range Registers.** MTRRs are supported. 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.
13	PGE	**Page Global Bit.** The global bit is supported 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.
14	MCA	**Machine Check Architecture.** The Machine Check Architecture, which provides a compatible mechanism for error reporting in P6 family, Pentium 4, Intel Xeon processors, and future processors, is supported. The MCG_CAP MSR contains feature bits describing how many banks of error reporting MSRs are supported.
15	CMOV	**Conditional Move Instructions.** The conditional move instruction CMOV is supported. In addition, if x87 FPU is present as indicated by the CPUID.FPU feature bit, then the FCOMI and FCMOV instructions are supported
16	PAT	**Page Attribute Table.** Page Attribute Table is supported. 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.
17	PSE-36	**36-Bit Page Size Extension.** 4-MByte pages addressing physical memory beyond 4 GBytes are supported 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.
18	PSN	**Processor Serial Number.** The processor supports the 96-bit processor identification number feature and the feature is enabled.
19	CLFSH	**CLFLUSH Instruction.** CLFLUSH Instruction is supported.
20	Reserved	Reserved.
21	DS	**Debug Store.** The processor supports the ability to write debug information into a memory resident buffer. This feature is used by the branch trace store (BTS) and precise event-based sampling (PEBS) facilities (see Chapter 26, “Introduction to Virtual Machine Extensions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C).
22	ACPI	**Thermal Monitor and Software Controlled Clock Facilities.** The processor implements internal MSRs that allow processor temperature to be monitored and processor performance to be modulated in predefined duty cycles under software control.
23	MMX	**Intel MMX Technology.** The processor supports the Intel MMX technology.

Table 1-6. More on Feature Information Returned in the EDX Register (Continued)

Bit #	Mnemonic	Description
24	FXSR	**FXSAVE and FXRSTOR Instructions.** The FXSAVE and FXRSTOR instructions are supported for fast save and restore of the floating-point context. Presence of this bit also indicates that CR4.OSFXSR is available for an operating system to indicate that it supports the FXSAVE and FXRSTOR instructions.
25	SSE	**SSE.** The processor supports the SSE extensions.
26	SSE2	**SSE2.** The processor supports the SSE2 extensions.
27	SS	**Self Snoop.** The processor supports the management of conflicting memory types by performing a snoop of its own cache structure for transactions issued to the bus.
28	HTT	**Max APIC IDs reserved field is Valid.** A value of 0 for HTT indicates there is only a single logical processor in the package and software should assume only a single APIC ID is reserved. A value of 1 for HTT indicates the value in CPUID.1.EBX[23:16] (the Maximum number of addressable IDs for logical processors in this package) is valid for the package.
29	TM	**Thermal Monitor.** The processor implements the thermal monitor automatic thermal control circuitry (TCC).
30	Reserved	Reserved.
31	PBE	**Pending Break Enable.** The processor supports 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. Bit 10 (PBE enable) in the IA32_MISC_ENABLE MSR enables this capability.

INPUT EAX = 02H: Cache and TLB Information Returned in EAX, EBX, ECX, EDX

When CPUID executes with EAX set to 02H, the processor returns information about the processor's internal caches and TLBs in the EAX, EBX, ECX, and EDX registers.

The encoding is as follows:

- The least-significant byte in register EAX (register AL) indicates the number of times the CPUID instruction must be executed with an input value of 02H to get a complete description of the processor's caches and TLBs. The first member of the family of Pentium 4 processors will return a 01H.
- 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. Table 1-7 shows the encoding of these descriptors. 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 or TLB types. The descriptors may appear in any order.

Table 1-7. Encoding of CPUID Leaf 2 Descriptors

Descriptor Value	Type	Cache or TLB Description
00H	General	Null descriptor, this byte contains no information.
01H	TLB	Instruction TLB: 4 KByte pages, 4-way set associative, 32 entries.
02H	TLB	Instruction TLB: 4 MByte pages, fully associative, 2 entries.
03H	TLB	Data TLB: 4 KByte pages, 4-way set associative, 64 entries.
04H	TLB	Data TLB: 4 MByte pages, 4-way set associative, 8 entries.
05H	TLB	Data TLB1: 4 MByte pages, 4-way set associative, 32 entries.
06H	Cache	1st-level instruction cache: 8 KBytes, 4-way set associative, 32 byte line size.
08H	Cache	1st-level instruction cache: 16 KBytes, 4-way set associative, 32 byte line size.
09H	Cache	1st-level instruction cache: 32KBytes, 4-way set associative, 64 byte line size.
0AH	Cache	1st-level data cache: 8 KBytes, 2-way set associative, 32 byte line size.

Table 1-7. Encoding of CPUID Leaf 2 Descriptors (Continued)

Descriptor Value	Type	Cache or TLB Description
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.

Table 1-7. Encoding of CPUID Leaf 2 Descriptors (Continued)

Descriptor Value	Type	Cache or TLB Description
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-μop, 8-way set associative.
71H	Cache	Trace cache: 16 K-μop, 8-way set associative.
72H	Cache	Trace cache: 32 K-μop, 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.

Table 1-7. Encoding of CPUID Leaf 2 Descriptors (Continued)

Descriptor Value	Type	Cache or TLB Description
C1H	STLB	Shared 2nd-Level TLB: 4 KByte/2MByte pages, 8-way associative, 1024 entries.
C2H	DTLB	DTLB: 2 MByte/4 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: 2 MByte/4 MByte 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 1-1. 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 indicates that CPUID needs to be executed once with an input value of 2 to retrieve complete information about caches and TLBs.
- 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.

INPUT EAX = 04H: Returns Deterministic Cache Parameters for Each Level

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. The architecturally defined fields reported by deterministic cache parameters are documented in Table 1-3.

The CPUID leaf 4 also reports data that can be used to derive the topology of processor cores in a physical package. This information is constant for all valid index values. Software can query the raw data reported by executing CPUID with EAX=04H and ECX=0H and use it as part of the topology enumeration algorithm described in Chapter 11, “Multiple-Processor Management,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A.

INPUT EAX = 05H: Returns MONITOR and MWAIT Features

When CPUID executes with EAX set to 05H, the processor 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. See Table 1-3.

INPUT EAX = 06H: Returns Thermal and Power Management Features

When CPUID executes with EAX set to 06H, the processor returns information about thermal and power management features. See Table 1-3.

INPUT EAX = 07H: Returns Structured Extended Feature Enumeration Information

When CPUID executes with EAX set to 07H and ECX = 0H, the processor returns information about the maximum number of sub-leaves that contain extended feature flags. See Table 1-3.

When CPUID executes with EAX set to 07H and ECX = n (n ≥ 1 and less than the number of non-zero bits in CPUID.(EAX=07H, ECX= 0H).EAX), the processor returns information about extended feature flags. See Table 1-3. In sub-leaf 0, only EAX has the number of sub-leaves. In sub-leaf 0, EBX, ECX & EDX all contain extended feature flags.

INPUT EAX = 09H: Returns Direct Cache Access Information

When CPUID executes with EAX set to 09H, the processor returns information about Direct Cache Access capabilities. See Table 1-3.

INPUT EAX = 0AH: Returns Architectural Performance Monitoring Features

When CPUID executes with EAX set to 0AH, the processor returns information about support for architectural performance monitoring capabilities. Architectural performance monitoring is supported if the version ID (see Table 1-3) is greater than Pn 0. See Table 1-3.

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 20, “Debug, Branch Profile, TSC, and Intel® Resource Director Technology (Intel® RDT) Features,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3.

INPUT EAX = 0BH: Returns Extended Topology Information

CPUID leaf 1FH is a preferred superset to leaf 0BH. Intel recommends first checking for the existence of Leaf 1FH before using leaf 0BH.

When CPUID executes with EAX set to 0BH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 0BH by verifying (a) the highest leaf index supported by CPUID is >= 0BH, and (b) CPUID.0BH:EBX[15:0] reports a non-zero value. See Table 1-3.

INPUT EAX = 0DH: Returns Processor Extended States Enumeration Information

When CPUID executes with EAX set to 0DH and ECX = 0H, the processor returns information about the bit-vector representation of all processor state extensions that are supported in the processor and storage size requirements of the XSAVE/XRSTOR area. See Table 1-3.

When CPUID executes with EAX set to 0DH and ECX = n (n > 1, and is a valid sub-leaf index), the processor returns information about the size and offset of each processor extended state save area within the XSAVE/XRSTOR area. See Table 1-3. Software can use the forward-extendable technique depicted below to query the valid sub-leaves and obtain size and offset information for each processor extended state save area:

```
For i = 2 to 62 // sub-leaf 1 is reserved
   IF (CPUID.(EAX=0DH, ECX=0):VECTOR[i] = 1 ) // VECTOR is the 64-bit value of EDX:EAX
       Execute CPUID.(EAX=0DH, ECX = i) to examine size and offset for sub-leaf i;
   FI;
```

INPUT EAX = 0FH: Returns Intel Resource Director Technology (Intel RDT) Monitoring Enumeration Information

When CPUID executes with EAX set to 0FH and ECX = 0, the processor 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. See Table 1-3.

When CPUID executes with EAX set to 0FH and ECX = n (n >= 1, and is a valid ResID), the processor returns information software can use to program IA32_PQR_ASSOC, IA32_QM_EVTSEL MSRs before reading QoS data from the IA32_QM_CTR MSR.

INPUT EAX = 10H: Returns Intel Resource Director Technology (Intel RDT) Allocation Enumeration Information

When CPUID executes with EAX set to 10H and ECX = 0, the processor 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. See Table 1-3.

When CPUID executes with EAX set to 10H and ECX = n (n >= 1, and is a valid ResID), the processor returns information about available classes of service and range of QoS mask MSRs that software can use to configure each class of services using capability bit masks in the QoS Mask registers, IA32_resourceType_Mask_n.

INPUT EAX = 12H: Returns Intel SGX Enumeration Information

When CPUID executes with EAX set to 12H and ECX = 0H, the processor returns information about Intel SGX capabilities. See Table 1-3.

When CPUID executes with EAX set to 12H and ECX = 1H, the processor returns information about Intel SGX attributes. See Table 1-3.

When CPUID executes with EAX set to 12H and ECX = n (n > 1), the processor returns information about Intel SGX Enclave Page Cache. See Table 1-3.

INPUT EAX = 14H: Returns Intel Processor Trace Enumeration Information

When CPUID executes with EAX set to 14H and ECX = 0H, the processor returns information about Intel Processor Trace extensions. See Table 1-3.

When CPUID executes with EAX set to 14H and ECX = n (n > 0 and less than the number of non-zero bits in CPUID.(EAX=14H, ECX= 0H).EAX), the processor returns information about packet generation in Intel Processor Trace. See Table 1-3.

INPUT EAX = 15H: Returns Time Stamp Counter and Nominal Core Crystal Clock Information

When CPUID executes with EAX set to 15H and ECX = 0H, the processor returns information about Time Stamp Counter and Core Crystal Clock. See Table 1-3.

INPUT EAX = 16H: Returns Processor Frequency Information

When CPUID executes with EAX set to 16H, the processor returns information about Processor Frequency Information. See Table 1-3.

INPUT EAX = 17H: Returns System-On-Chip Information

When CPUID executes with EAX set to 17H, the processor returns information about the System-On-Chip Vendor Attribute Enumeration. See Table 1-3.

INPUT EAX = 18H: Returns Deterministic Address Translation Parameters Information

When CPUID executes with EAX set to 18H, the processor returns information about the Deterministic Address Translation Parameters. See Table 1-3.

INPUT EAX = 19H: Returns Key Locker Information

When CPUID executes with EAX set to 19H, the processor returns information about Key Locker. See Table 1-3.

INPUT EAX = 1AH: Returns Hybrid Information

When CPUID executes with EAX set to 1AH, the processor returns information about hybrid capabilities. See Table 1-3.

INPUT EAX = 1BH: Returns PCONFIG Information

When CPUID executes with EAX set to 1BH, the processor returns information about PCONFIG capabilities. This information is enumerated in sub-leaves selected by the value of ECX (starting with 0).

Each sub-leaf of CPUID function 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). Currently, TME-MK and TSE are the only defined targets. TME-MK is indicated by identifier 1, and TSE is indicated by identifier 2. An identifier of 0 indicates an invalid target. If TME-MK is a supported target, the MKTME_KEY_PROGRAM leaf of PCONFIG is available. If TSE is a supported target, the TSE_KEY_PROGRAM and the TSE_KEY_PROGRAM_WRAPPED leaves of PCONFIG are available. 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.

INPUT EAX = 1CH: Returns Last Branch Record Information

When CPUID executes with EAX set to 1CH, the processor returns information about LBRs (the architectural feature). See Table 1-3.

INPUT EAX = 1DH: Returns Tile Information

When CPUID executes with EAX set to 1DH and ECX = 0H, the processor returns information about tile architecture. See Table 1-3.

When CPUID executes with EAX set to 1DH and ECX = 1H, the processor returns information about tile palette 1. See Table 1-3.

INPUT EAX = 1EH: Returns TMUL Information

When CPUID executes with EAX set to 1EH, the processor returns information about TMUL capabilities. See Table 1-3.

INPUT EAX = 1FH: Returns V2 Extended Topology Information

When CPUID executes with EAX set to 1FH, the processor returns information about extended topology enumeration data. Software must detect the presence of CPUID leaf 1FH by verifying (a) the highest leaf index supported by CPUID is >= 1FH, and (b) CPUID.1FH:EBX[15:0] reports a non-zero value. See Table 1-3.

INPUT EAX = 20H: Returns Processor History Reset Information

When CPUID executes with EAX set to 20H, the processor returns information about processor history reset. See Table 1-3.

INPUT EAX = 23H: Returns Architectural Performance Monitoring Extended Information

When CPUID executes with EAX set to 23H, the processor returns architectural performance monitoring extended information. See Table 1-3.

INPUT EAX = 24H: Returns Intel AVX10 Converged Vector ISA Information

When CPUID executes with EAX set to 24H, the processor returns Intel AVX10 converged vector ISA information. See Table 1-3.

METHODS FOR RETURNING BRANDING INFORMATION

Use the following techniques to access branding information:

1. Processor brand string method; this method also returns the processor's maximum operating frequency
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: "Identification of Earlier IA-32 Processors" in Chapter 21 of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1.

The Processor Brand String Method

Figure 1-4 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 maximum operating frequency of the processor to the EAX, EBX, ECX, and EDX registers.

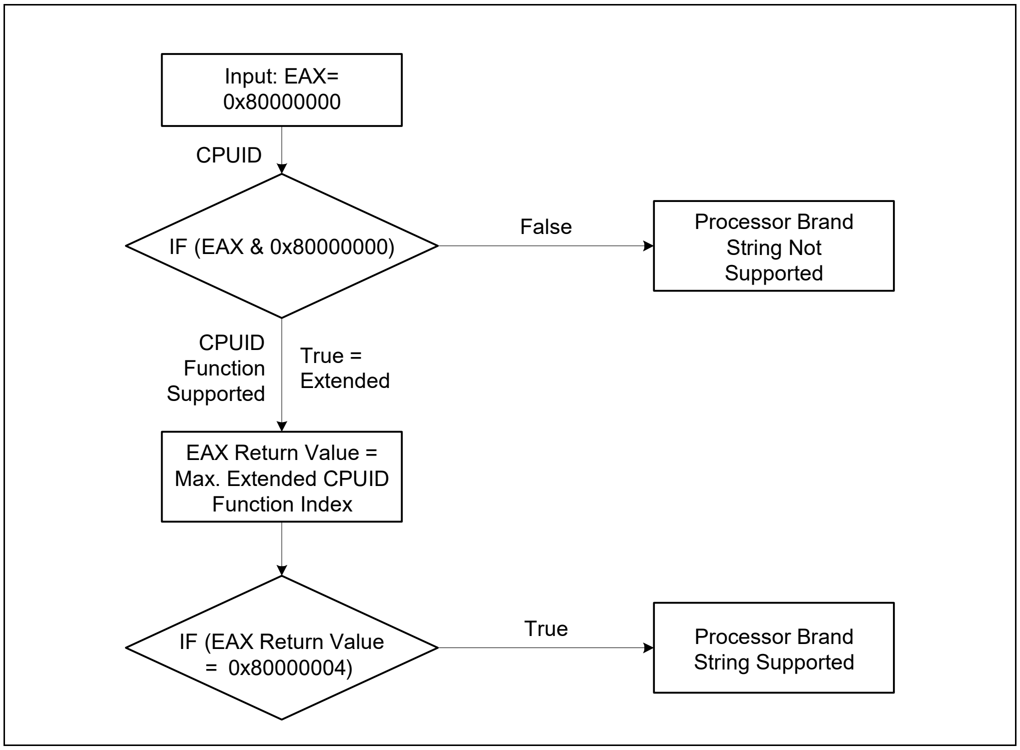


Figure 1-4. Determination of Support for the Processor Brand String

How Brand Strings Work

To use the brand string method, execute CPUID with EAX input of 8000002H through 80000004H. For each input value, CPUID returns 16 ASCII characters using EAX, EBX, ECX, and EDX. The returned string will be NULL-terminated.

Table 1-8 shows the brand string that is returned by the first processor in the Pentium 4 processor family.

Table 1-8. Processor Brand String Returned with Pentium 4 Processor

EAX Input Value	Return Values	ASCII Equivalent
80000002H	EAX = 20202020H	“ ”
	EBX = 20202020H	“ ”
	ECX = 20202020H	“ ”
	EDX = 6E492020H	“nI ”
80000003H	EAX = 286C6574H	“(let”
	EBX = 50202952H	“P)R”
	ECX = 69746E65H	“itne”
	EDX = 52286D75H	“R(mu”
80000004H	EAX = 20342029H	“ 4)”
	EBX = 20555043H	“ UPC”
	ECX = 30303531H	“0051”
	EDX = 007A484DH	“\0zHM”

Extracting the Maximum Processor Frequency from Brand Strings

Figure 1-5 provides an algorithm which software can use to extract the maximum processor operating frequency from the processor brand string.

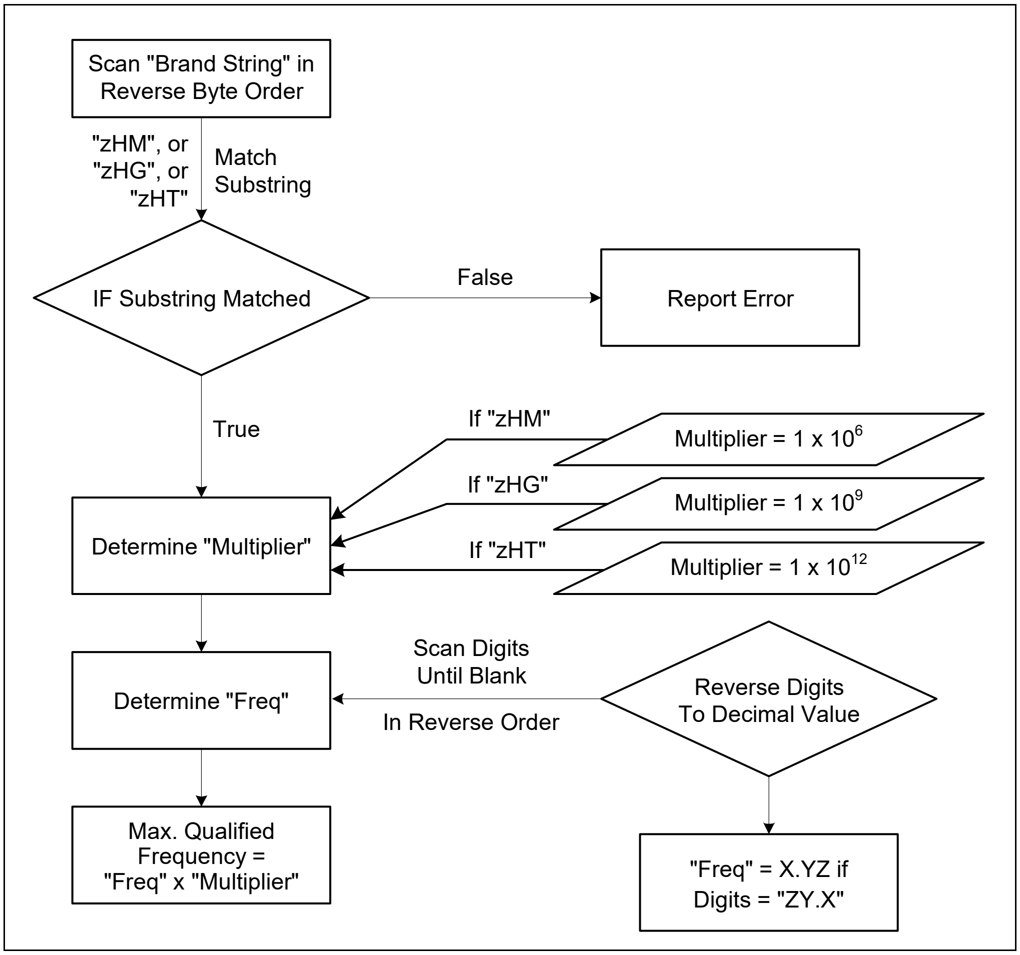


Figure 1-5. Algorithm for Extracting Maximum Processor Frequency

NOTE

When a frequency is given in a brand string, it is the maximum qualified frequency of the processor, not the frequency at which the processor is currently running.

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 system software and is accessible from system- and user-level code. In this table, each brand index is associate with an ASCII brand identification string that identifies the official Intel family and model number of a processor.

When CPUID executes with EAX set to 01H, 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 1-9 shows brand indices that have identification strings associated with them.

Table 1-9. 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(R) Celeron(R) processor[1]
02H	Intel(R) Pentium(R) III processor[1]
03H	Intel(R) Pentium(R) III Xeon(R) processor; If processor signature = 000006B1h, then Intel(R) Celeron(R) processor
04H	Intel(R) Pentium(R) III processor
06H	Mobile Intel(R) Pentium(R) III processor-M
07H	Mobile Intel(R) Celeron(R) processor[1]
08H	Intel(R) Pentium(R) 4 processor
09H	Intel(R) Pentium(R) 4 processor
0AH	Intel(R) Celeron(R) processor[1]
0BH	Intel(R) Xeon(R) processor; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor MP
0CH	Intel(R) Xeon(R) processor MP
0EH	Mobile Intel(R) Pentium(R) 4 processor-M; If processor signature = 00000F13h, then Intel(R) Xeon(R) processor
0FH	Mobile Intel(R) Celeron(R) processor[1]
11H	Mobile Genuine Intel(R) processor
12H	Intel(R) Celeron(R) M processor
13H	Mobile Intel(R) Celeron(R) processor[1]
14H	Intel(R) Celeron(R) processor
15H	Mobile Genuine Intel(R) processor
16H	Intel(R) Pentium(R) M processor
17H	Mobile Intel(R) Celeron(R) processor[1]
18H – 0FFH	RESERVED

NOTES:

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

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.

1.5 COMPRESSED DISPLACEMENT (DISP8*N) SUPPORT IN EVEX

For memory addressing using disp8 form, EVEX-encoded instructions always use a compressed displacement scheme by multiplying disp8 in conjunction with a scaling factor N that is determined based on the vector length, the value of EVEX.b bit (embedded broadcast) and the input element size of the instruction. In general, the factor N corresponds to the number of bytes characterizing the internal memory operation of the input operand (e.g., 64 when the accessing a full 512-bit memory vector). The scale factor N is listed in Table 1-10 and Table 1-11 below, where EVEX encoded instructions are classified using the **tupletype** attribute. The scale factor N of each tupletype is listed based on the vector length (VL) and other factors affecting it.

Table 1-10 covers EVEX-encoded instructions which has a load semantic in conjunction with additional computational or data element movement operation, operating either on the full vector or half vector (due to conversion of numerical precision from a wider format to narrower format). EVEX.b is supported for such instructions for data element sizes which are either dword or qword.

EVEX-encoded instruction that are pure load/store, and "Load+op" instruction semantic that operate on data element size less then dword do not support broadcasting using EVEX.b. These are listed in Table 1-11. Table 1-11 also includes many broadcast instructions which perform broadcast using a subset of data elements without using EVEX.b. These instructions and a few data element size conversion instruction are covered in Table 1-11. Instruction classified in Table 1-11 do not use EVEX.b and EVEX.b must be 0, otherwise #UD will occur.

The tupletype will be referenced in the instruction operand encoding table in the reference page of each instruction, providing the cross reference for the scaling factor N to encoding memory addressing operand.

Note that the disp8*N rules still apply when using 16b addressing.

Table 1-10. Compressed Displacement (DISP8*N) Affected by Embedded Broadcast

TupleType	EVEX.b	InputSize	EVEX.W	Broadcast	N (VL=128)	N (VL=256)	N (VL= 512)	Comment
Full	0	32bit	0	none	16	32	64	Load+Op (Full Vector Dword/Qword)
	1	32bit	0	{1tox}	4	4	4	
	0	64bit	1	none	16	32	64	
	1	64bit	1	{1tox}	8	8	8	
Half	0	32bit	0	none	8	16	32	Load+Op (Half Vector)
	1	32bit	0	{1tox}	4	4	4	

Table 1-11. EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast

Tuple Type	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Full Mem	N/A	N/A	16	32	64	Load/store or subDword full vector
Tuple1 Scalar	8bit	N/A	1	1	1	1Tuple
	16bit	N/A	2	2	2	
	32bit	0	4	4	4	
	64bit	1	8	8	8	
Tuple1 Fixed	32bit	N/A	4	4	4	1 Tuple, memsize not affected by EVEX.W
	64bit	N/A	8	8	8	
Tuple1_4X	32bit	0	16[1]	N/A	16	4FMA(PS)

Table 1-11. EVEX DISP8*N for Instructions Not Affected by Embedded Broadcast (Continued)

Tuple Type	InputSize	EVEX.W	N (VL= 128)	N (VL= 256)	N (VL= 512)	Comment
Tuple2	32bit	0	8	8	8	Broadcast (2 elements)
	64bit	1	NA	16	16	
Tuple4	32bit	0	NA	16	16	Broadcast (4 elements)
	64bit	1	NA	NA	32	
Tuple8	32bit	0	NA	NA	32	Broadcast (8 elements)
Half Mem	N/A	N/A	8	16	32	SubQword Conversion
Quarter Mem	N/A	N/A	4	8	16	SubDword Conversion
Eighth Mem	N/A	N/A	2	4	8	SubWord Conversion
Mem128	N/A	N/A	16	16	16	Shift count from memory
MOVDDUP	N/A	N/A	8	32	64	VMOVDDUP

NOTES:

1. Scalar.

1.6 BFLOAT16 FLOATING-POINT FORMAT

Intel® Deep Learning Boost (Intel® DL Boost) uses bfloat16 format (BF16). Figure 1-6 illustrates BF16 versus FP16 and FP32.

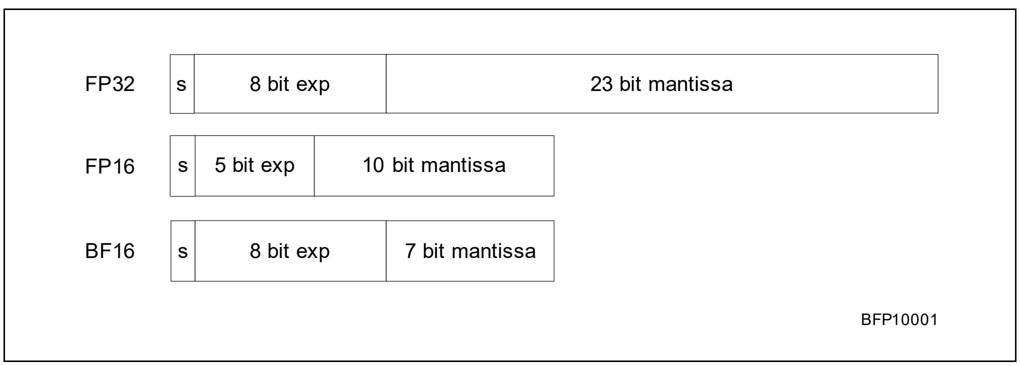


Figure 1-6. Comparison of BF16 to FP16 and FP32

BF16 has several advantages over FP16:

- It can be seen as a short version of FP32, skipping the least significant 16 bits of mantissa.
- There is no need to support denormals; FP32, and therefore also BF16, offer more than enough range for deep learning training tasks.
- FP32 accumulation after the multiply is essential to achieve sufficient numerical behavior on an application level.
- Hardware exception handling is not needed as this is a performance optimization; industry is designing algorithms around checking inf/NaN.

1.7 FP8 FORMAT

The FP8 format defines a new class of 8-bit floating point data types to speed up both training and inference AI workloads. The FP8 format is important for optimizing memory footprint and core AI compute density along with power and performance efficiency. The two new FP8 formats—E5M2 and E4M3—are introduced by the Open Compute Project's *OCP 8-bit Floating Point Specification (OFP8)*. Conversion to and from these formats will be supported as part of Intel® AVX10.2 ISA. In instruction mnemonics and pseudo code throughout this document, E5M2 and E4M3 are denoted by BF8 (or bf8) and HF8 (or hf8), respectively.

1.7.1 Numeric Definition

Two FP8 formats are supported:

- E5M2, which has 1 sign bit, 5 exponent bits, and 2 mantissa bits
- E4M3, which has 1 sign bit, 4 exponent bits, and 3 mantissa bits

Due to the very small range and precision of these data types, both formats are needed to converge and reach the required accuracy across a wide range of AI topologies. Table 1-12 describes the numerics of each format. While E5M2 follows standard floating point representations, the E4M3 format has a non-standard definition, including the same representation for infinity and NaN in order to increase its range.

Table 1-12. FP8 Formats Numeric Definitions

Encoding	E5M2 FP8	E4M3 FP8
Exponent Bias	15	7
Maximum Normal	S.11110.11 = 57344.0 ($1.75 * 2^{15}$)	S.1111.110 = 448.0 ($1.75 * 2^{8}$)
Minimum Normal	S.00001.00 = 6.10e-05 (2^{-14})	S.0001.000 = 1.56e-02 (2^{-6})
Maximum Denormal	S.00000.11 = 4.57e-05 ($0.75 * 2^{-14}$)	S.0000.111 = 1.36e-02 ($0.875 * 2^{-6}$)
Minimum Denormal	S.00000.01 = 1.52e-05 ($0.25 * 2^{-14}$)	S.0000.001 = 1.95e-03 ($0.125 * 2^{-6}$)
NaNs	S.11111.[01, 10, 11]	S.1111.111
Infinity	S.11111.00	N/A

1.7.2 Floating-Point Rounding, Denormal Handling, NaN/Inf/Overflow Handling, and FP Exceptions

Intel architecture supports AMX compute instructions and AVX convert instructions that operate on FP8 values.

The convert instructions have two versions, *NE* and *BIAS*, indicating the rounding modes.

- **Floating-point rounding**. Excluding the *BIAS* convert instructions, all instructions use RNE ("round to nearest even") rounding mode. The *BIAS* convert instructions use RNE in case the input is denormal and truncate (i.e., chop or round towards zero) for normal inputs.
- **Denormal handling**. All instructions function as if floating-point exceptions are masked. For any type of input, instructions behave as if MXCSR.DAZ is not set. For FP8 output type, instructions behave as if MXCSR.FTZ is not set. For any other type of output, instructions behave as if MXCSR.FTZ is set.
- **Special numbers**. NaN/Inf/overflow handling. Excluding convert instructions, all instructions behave in their regular way. Infinity is bypassed, NaN is bypassed as QNaN, and overflow returns infinity. The convert instructions have saturation and non-saturation versions. Table 1-13 describes the convert instruction behavior in these cases.
- Floating-point exceptions:
  — Instructions operating on FP8 values do not consult the MXCSR.
  — Instructions operating on FP8 values do not raise exceptions.

— Instructions operating on FP8 values do not update the MXCSR.
— All convert instructions with FP8 results update the MXCSR.

Table 1-13. FP Numerical Handling of Converts

Version	Scenario	E5M2	E4M3
Regular	NaN at input	S.11111.[10, 11]	S.1111.111
	+/- infinity at input	S.11111.00	S.1111.111
	Overflow due to conversion /rounding	S.11111.00	S.1111.111
Saturated	NaN at input	S.11111.[10, 11]	S.1111.111
	+/- infinity at input	S.11111.00	S.1111.111
	Overflow due to conversion /rounding	S.11110.11	S.1111.110

CHAPTER 2
INSTRUCTION SET REFERENCE, A-Z

Instructions described in this document follow the general documentation convention established in the Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A. Additionally, some instructions use notation conventions as described below.

In the instruction encoding, the ModR/M byte is represented several ways depending on the role it plays. The ModR/M byte has 3 fields: 2-bit mod field, a 3-bit reg field and a 3-bit r/m field. When all bits of the ModR/M 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 ModR/M byte must contain fixed values, those values are specified as follows:

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

NOTE

Historically the Intel® 64 and IA-32 Architectures Software Developer's Manual only specified the reg field restrictions with the notation /0 ... /7 and did not specify restrictions on the mod and r/m fields in the encoding boxes.

2.1 INSTRUCTION SET REFERENCE

AADD—Atomically Add

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NP 0F38 FC !(11):rrr:bbb AADD *my, ry*	A	V/V	RAO-INT	Atomically add *my* with *ry* and store the result in *my*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:r/m (r, w)	ModRM:reg (r)	N/A	N/A

Description

This instruction is a remote atomic integer operation. This instruction atomically adds the destination operand (first operand) and the source operand (second operand), and then stores the result in the destination operand.

The destination operand is a memory location and the source operand is a register. In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. The destination operand must be naturally aligned with respect to the data size, at a 4-byte boundary, or an 8-byte boundary if used with a REX.W prefix in 64-bit mode.

This instruction requires that the destination operand has a write-back (WB) memory type and it is implemented using the weakly-ordered memory consistency model of write combining (WC) memory type. Before the operation, the cache line is written-back (if modified) and invalidated from the processor cache. When the operation completes, the processor may optimize the cacheability of the destination address by writing the result only to specific levels of the cache hierarchy. Because this instructions uses a weakly-ordered memory consistency model, a fencing operation implemented with LFENCE, SFENCE, or MFENCE instruction should be used in conjunction with AADD if a stronger ordering is required. However, note that AADD is not ordered with respect to a younger LFENCE.

Any attempt to execute the AADD instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

AADD dest, src

```
dest := dest + src;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS	For an illegal address in the SS segment.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

AAND—Atomically AND

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F38 FC !(11):rrr:bbb AAND *my, ry*	A	V/V	RAO-INT	Atomically AND *my* with *ry* and store the result in *my*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:r/m (r, w)	ModRM:reg (r)	N/A	N/A

Description

This instruction is a remote atomic integer operation. This instruction atomically performs a bitwise AND operation of the destination operand (first operand) and the source operand (second operand), and then stores the result in the destination operand.

The destination operand is a memory location and the source operand is a register. In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. The destination operand must be naturally aligned with respect to the data size, at a 4-byte boundary, or an 8-byte boundary if used with a REX.W prefix in 64-bit mode.

This instruction requires that the destination operand has a write-back (WB) memory type and it is implemented using the weakly-ordered memory consistency model of write combining (WC) memory type. Before the operation, the cache line is written-back (if modified) and invalidated from the processor cache. When the operation completes, the processor may optimize the cacheability of the destination address by writing the result only to specific levels of the cache hierarchy. Because this instructions uses a weakly-ordered memory consistency model, a fencing operation implemented with LFENCE, SFENCE, or MFENCE instruction should be used in conjunction with AAND if a stronger ordering is required. However, note that AAND is not ordered with respect to a younger LFENCE.

Any attempt to execute the AAND instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

AAND dest, src

```
dest := dest AND src;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS	For an illegal address in the SS segment.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

AOR—Atomically OR

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F38 FC !(11):rrr:bbb AOR *my, ry*	A	V/V	RAO-INT	Atomically OR *my* with *ry* and store the result in *my*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:r/m (r, w)	ModRM:reg (r)	N/A	N/A

Description

This instruction is a remote atomic integer operation. This instruction atomically performs a bitwise OR operation of the destination operand (first operand) and the source operand (second operand), and then stores the result in the destination operand.

The destination operand is a memory location and the source operand is a register. In 64-bit mode, the instruction’s default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. The destination operand must be naturally aligned with respect to the data size, at a 4-byte boundary, or an 8-byte boundary if used with a REX.W prefix in 64-bit mode.

This instruction requires that the destination operand has a write-back (WB) memory type and it is implemented using the weakly-ordered memory consistency model of write combining (WC) memory type. Before the operation, the cache line is written-back (if modified) and invalidated from the processor cache. When the operation completes, the processor may optimize the cacheability of the destination address by writing the result only to specific levels of the cache hierarchy. Because this instructions uses a weakly-ordered memory consistency model, a fencing operation implemented with LFENCE, SFENCE, or MFENCE instruction should be used in conjunction with AOR if a stronger ordering is required. However, note that AOR is not ordered with respect to a younger LFENCE.

Any attempt to execute the AOR instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

AOR dest, src

```
dest := dest OR src;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS	For an illegal address in the SS segment.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

AXOR—Atomically XOR

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F38 FC !(11):rrr:bbb AXOR *my, ry*	A	V/V	RAO-INT	Atomically XOR *my* with *ry* and store the result in *my*.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
A	ModRM:r/m (r, w)	ModRM:reg (r)	N/A	N/A

Description

This instruction is a remote atomic integer operation. This instruction atomically performs a bitwise XOR operation of the destination operand (first operand) and the source operand (second operand), and then stores the result in the destination operand.

The destination operand is a memory location and the source operand is a register. In 64-bit mode, the instruction's default operation size is 32 bits. Using a REX prefix in the form of REX.R permits access to additional registers (R8-R15). Using a REX prefix in the form of REX.W promotes operation to 64 bits. The destination operand must be naturally aligned with respect to the data size, at a 4-byte boundary, or an 8-byte boundary if used with a REX.W prefix in 64-bit mode.

This instruction requires that the destination operand has a write-back (WB) memory type and it is implemented using the weakly-ordered memory consistency model of write combining (WC) memory type. Before the operation, the cache line is written-back (if modified) and invalidated from the processor cache. When the operation completes, the processor may optimize the cacheability of the destination address by writing the result only to specific levels of the cache hierarchy. Because this instructions uses a weakly-ordered memory consistency model, a fencing operation implemented with LFENCE, SFENCE, or MFENCE instruction should be used in conjunction with AXOR if a stronger ordering is required. However, note that AXOR is not ordered with respect to a younger LFENCE.

Any attempt to execute the AXOR instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

AXOR dest, src

```
dest := dest XOR src;
```

Flags Affected

None.

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Real-Address Mode Exceptions

#GP	If any part of the operand lies outside the effective address space from 0 to FFFFH.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS	For an illegal address in the SS segment.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

#GP(0)	If the memory address is in a non-canonical form.
	If the memory address is not naturally aligned to the operand size.
	If the memory address memory type is not write-back (WB).
#SS(0)	If a memory address referencing the SS segment is in a non-canonical form.
#PF(fault-code)	If a page fault occurs.
#UD	If the LOCK prefix is used.
	If CPUID.(EAX=07H, ECX=01H):EAX.RAO-INT[bit 3] = 0.

MOVRS—Move Read-Shared Value

Opcode / Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
NOREP 0F38 8B !(11):rrr:bbb MOVRS rv, mv	A	V/N.E.	MOVRS	Move a read-shared word, doubleword, or quadword from mv to rv.
NOREP 0F38 8A !(11):rrr:bbb MOVRS r8, m8	A	V/N.E.	MOVRS	Move a read-shared byte from m8 to r8.
EVEX.128.F2.MAP5.W0 6F !(11):rrr:bbb VMOVRSB xmm1{k1}{z}, m128	B	V/N.E.	AVX10 and MOVRS	Move read-shared bytes from m128 to xmm1 subject to writemask k1.
EVEX.256.F2.MAP5.W0 6F !(11):rrr:bbb VMOVRSB ymm1{k1}{z}, m256	B	V/N.E.	AVX10 and MOVRS	Move read-shared bytes from m256 to ymm1 subject to writemask k1.
EVEX.512.F2.MAP5.W0 6F !(11):rrr:bbb VMOVRSB zmm1{k1}{z}, m512	B	V/N.E.	AVX10 and MOVRS	Move read-shared bytes from m512 to zmm1 subject to writemask k1.
EVEX.128.F3.MAP5.W0 6F !(11):rrr:bbb VMOVRSD xmm1{k1}{z}, m128	B	V/N.E.	AVX10 and MOVRS	Move read-shared doublewords from m128 to xmm1 subject to writemask k1.
EVEX.256.F3.MAP5.W0 6F !(11):rrr:bbb VMOVRSD ymm1{k1}{z}, m256	B	V/N.E.	AVX10 and MOVRS	Move read-shared doublewords from m256 to ymm1 subject to writemask k1.
EVEX.512.F3.MAP5.W0 6F !(11):rrr:bbb VMOVRSD zmm1{k1}{z}, m512	B	V/N.E.	AVX10 and MOVRS	Move read-shared doublewords from m512 to zmm1 subject to writemask k1.
EVEX.128.F3.MAP5.W1 6F !(11):rrr:bbb VMOVRSQ xmm1{k1}{z}, m128	B	V/N.E.	AVX10 and MOVRS	Move read-shared quadwords from m128 to xmm1 subject to writemask k1.
EVEX.256.F3.MAP5.W1 6F !(11):rrr:bbb VMOVRSQ ymm1{k1}{z}, m256	B	V/N.E.	AVX10 and MOVRS	Move read-shared quadwords from m256 to ymm1 subject to writemask k1.
EVEX.512.F3.MAP5.W1 6F !(11):rrr:bbb VMOVRSQ zmm1{k1}{z}, m512	B	V/N.E.	AVX10 and MOVRS	Move read-shared quadwords from m512 to zmm1 subject to writemask k1.
EVEX.128.F2.MAP5.W1 6F !(11):rrr:bbb VMOVRSW xmm1{k1}{z}, m128	B	V/N.E.	AVX10 and MOVRS	Move read-shared words from m128 to xmm1 subject to writemask k1.
EVEX.256.F2.MAP5.W1 6F !(11):rrr:bbb VMOVRSW ymm1{k1}{z}, m256	B	V/N.E.	AVX10 and MOVRS	Move read-shared words from m256 to ymm1 subject to writemask k1.
EVEX.512.F2.MAP5.W1 6F !(11):rrr:bbb VMOVRSW zmm1{k1}{z}, m512	B	V/N.E.	AVX10 and MOVRS	Move read-shared words from m512 to zmm1 subject to writemask k1.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A
B	FULLMEM	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

Moves data from the source operand (second operand), a memory operand, to the destination operand (first operand), a register. If the destination is a general-purpose register, then both operands are the same size, which can be a byte, a word, a doubleword, or a quadword. If the destination is an XMM, YMM, or ZMM register, then the instruction reads data from a 128-bit, 256-bit, or 512-bit memory location; the data may be 16, 32, or 64 bytes; 8, 16, or 32 words; 4, 8, or 16 doublewords; or 2, 4, or 8 quadwords. Additionally, this instruction indicates the source memory location is likely to become read-shared by multiple processors, i.e., read in the future by at least one other processor before it is written, assuming it is ever written in the future. Implementations may optimize the behavior of the caches, especially shared caches, for this data for future reads by multiple processors. A future write to this data before it becomes read-shared will behave as usual, but its performance may be less optimal than if the current read were done via a load without a read-shared hint.

Operation

```
MOVRS DEST, SRC
DEST := SRC

VMOVRSB DEST, SRC (EVEX Encoded Version)
(KL, VL) = (16, 128), (32, 256), (64, 512)

FOR j := 0 TO KL-1
    i := j * 8
    IF k1[j] OR *no writemask*
        THEN DEST[i+7:i] := SRC[i+7:i]
        ELSE
            IF *merging-masking* ; merging-masking
                THEN *DEST[i+7:i] remains unchanged*
                ELSE DEST[i+7:i] := 0 ; zeroing-masking
            FI;
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0

VMOVRSW DEST, SRC (EVEX Encoded Version)
(KL, VL) = (8, 128), (16, 256), (32, 512)

FOR j := 0 TO KL-1
    i := j * 16
    IF k1[j] OR *no writemask*
        THEN DEST[i+15:i] := SRC[i+15:i]
        ELSE
            IF *merging-masking* ; merging-masking
                THEN *DEST[i+15:i] remains unchanged*
                ELSE DEST[i+15:i] := 0 ; zeroing-masking
            FI
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0
```

VMOVRSD DEST, SRC (EVEX Encoded Version)

```
(KL, VL) = (4, 128), (8, 256), (16, 512)

FOR j := 0 TO KL-1
    i := j * 32
    IF k1[j] OR *no writemask*
        THEN DEST[i+31:i] := SRC[i+31:i]
        ELSE
            IF *merging-masking* ; merging-masking
                THEN *DEST[i+31:i] remains unchanged*
                ELSE DEST[i+31:i] := 0 ; zeroing-masking
            FI
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0
```

VMOVRSQ DEST, SRC (EVEX Encoded Version)

```
(KL, VL) = (2, 128), (4, 256), (8, 512)

FOR j := 0 TO KL-1
    i := j * 64
    IF k1[j] OR *no writemask*
        THEN DEST[i+63:i] := SRC[i+63:i]
        ELSE
            IF *merging-masking* ; merging-masking
                THEN *DEST[i+63:i] remains unchanged*
                ELSE DEST[i+63:i] := 0 ; zeroing-masking
            FI
    FI;
ENDFOR
DEST[MAXVL-1:VL] := 0
```

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Exceptions

Exceptions Type Legacy-MOVRS; see Table 2-1.

EVEX-encoded instruction, see Table 2-49, “Type E4 Class Exception Conditions,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2A.

Table 2-1. Type Legacy-MOVRS Class Exception Conditions

Exception	Real	Virtual-8086	Protected and Compatibility	64-bit	Cause of Exception
Invalid Opcode, #UD	X	X	X		Only supported in 64-bit mode.
				X	If preceded by a LOCK prefix (F0H) or REP prefix (F2H, F3H).
				X	If any corresponding CPUID feature flag is '0'.
Stack, #SS(0)				X	If a memory address referencing the SS segment is in a non-canonical form.
Alignment Check #AC(0)				X	If alignment checking is enabled and an unaligned memory access is made while CPL=3.
General Protection, #GP(0)				X	If the memory address is in a non-canonical form.
Page Fault, #PF(fault-code)				X	For a page fault.

PREFETCHRST2—Prefetch Data into Caches Using a Read-Shared Hint

Opcode / Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 18 !(11):100:bbb PREFETCHRST2 m8	A	V/V	MOVRS	Move data from m8 closer to the processor using a read-shared hint.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:r/m (r)	N/A	N/A	N/A

Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy, in anticipation of this processor and at least one other reading the line before it is written, assuming it is ever written in the future. Brings data into an implementation-specific choice of cache, optimized for read sharing through shared cache levels. A future write to this data before it becomes read-shared will behave as usual, but its performance may be less optimal than if a prefetch or load without a read-shared hint were used for this access.

The source operand is a byte memory location.

Prefetches from uncacheable or WC memory are ignored.

The PREFETCHRST2 instruction is merely a hint and does not affect program behavior. If executed, this instruction moves data closer to the processor in anticipation of future use. The effect on the caches is implementation-dependent, and can be overloaded or ignored by a processor implementation. The amount of data prefetched is also processor implementation-dependent. It will, however, be a minimum of 32 bytes.

It should be noted that processors are free to speculatively fetch and cache data from system memory regions that are assigned a memory-type that permits speculative reads (that is, the WB, WC, and WT memory types). A PREFETCHRST2 instruction is considered a hint to this speculative behavior. Because this speculative fetching can occur at any time and is not tied to instruction execution, a PREFETCHRST2 instruction is not ordered with respect to the fence instructions (MFENCE, SFENCE, and LFENCE) or locked memory references. A PREFETCHRST2 instruction is also unordered with respect to CLFLUSH and CLFLUSHOPT instructions, other PREFETCHRST2 instructions, or any other general instruction. It is ordered with respect to serializing instructions such as CPUID, WRMSR, OUT, and MOV CR.

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

Operation

```
PREFETCHRST2 m8
FETCH (m8)
```

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Exceptions (All Operating Modes)

#UD If the LOCK prefix is used.

VSM4KEY4—Perform Four Rounds of SM4 Key Expansion

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F3.0F38.W0 DA /r VSM4KEY4 xmm1, xmm2, xmm3/m128	A	V/V	AVX AND SM4	Performs four rounds of SM4 key expansion.
VEX.256.F3.0F38.W0 DA /r VSM4KEY4 ymm1, ymm2, ymm3/m256	A	V/V	AVX AND SM4	Performs four rounds of SM4 key expansion.
EVEX.128.F3.0F38.W0 DA /r VSM4KEY4 xmm1, xmm2, xmm3/m128	B	V/V	AVX10 AND SM4	Performs four rounds of SM4 key expansion.
EVEX.256.F3.0F38.W0 DA /r VSM4KEY4 ymm1, ymm2, ymm3/m256	B	V/V	AVX10 AND SM4	Performs four rounds of SM4 key expansion.
EVEX.512.F3.0F38.W0 DA /r VSM4KEY4 zmm1, zmm2, zmm3/m512	B	V/V	AVX10 AND SM4	Performs four rounds of SM4 key expansion.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	N/A
B	FULLMEM	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	N/A

Description

The VSM4KEY4 instruction performs four rounds of SM4 key expansion. The instruction operates on independent 128-bit lanes.

Additional details can be found at: https://tools.ietf.org/html/draft-ribose-cfrg-sm4-10.

Both SM4 instructions use a common sbox table:

```
BYTE sbox[256] = {
0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C, 0x05,
0x2B, 0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99,
0x9C, 0x42, 0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC, 0x62,
0xE4, 0xB3, 0x1C, 0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F, 0xA6,
0x47, 0x07, 0xA7, 0xFC, 0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F, 0xA8,
0x68, 0x6B, 0x81, 0xB2, 0x71, 0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D, 0x35,
0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58, 0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78, 0x87,
0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27, 0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8, 0x9E,
0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5, 0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15, 0xA1,
0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD, 0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1, 0xE3,
0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29, 0x23, 0xAB, 0x0D, 0x53, 0x4E, 0x6F,
0xD5, 0xDB, 0x37, 0x45, 0xDE, 0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A, 0x72, 0x6D, 0x6C, 0x5B, 0x51,
0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD, 0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41, 0x1F, 0x10, 0x5A, 0xD8,
0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B, 0xBD, 0x2D, 0x74, 0xD0, 0x12, 0xB8, 0xE5, 0xB4, 0xB0,
0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E, 0x65, 0xB9, 0xF1, 0x09, 0xC5, 0x6E, 0xC6, 0x84,
0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79, 0xEE, 0x5F, 0x3E, 0xD7, 0xCB, 0x39, 0x48
}
```

Operation

```
define ROL32(dword, n):
    count := n % 32
    dest := (dword << count) | (dword >> (32-count))
    return dest

define SBOX_BYTE(dword, i):
    // sbox[] array defined in introduction
    return sbox[dword.byte[i]]

define lower_t(dword):
    tmp.byte[0] := SBOX_BYTE(dword, 0)
    tmp.byte[1] := SBOX_BYTE(dword, 1)
    tmp.byte[2] := SBOX_BYTE(dword, 2)
    tmp.byte[3] := SBOX_BYTE(dword, 3)
    return tmp

define L_KEY(dword):
    return dword ^ ROL32(dword, 13) ^ ROL32(dword, 23)

define T_KEY(dword):
    return L_KEY(lower_t(dword))

define F_KEY(X0, X1, X2, X3, round_key):
    return X0 ^ T_KEY(X1 ^ X2 ^ X3 ^ round_key)
```

VSM4KEY4 DEST, SRC1, SRC2

```
VL = (128, 256) // VEX versions
// or
VL = (128,256,512) // EVEX versions
KL := VL/128

for i in 0..KL-1:
    P[0] := SRC1.xmm[i].dword[0]
    P[1] := SRC1.xmm[i].dword[1]
    P[2] := SRC1.xmm[i].dword[2]
    P[3] := SRC1.xmm[i].dword[3]

    C[0] := F_KEY(P[0], P[1], P[2], P[3], SRC2.xmm[i].dword[0])
    C[1] := F_KEY(P[1], P[2], P[3], C[0], SRC2.xmm[i].dword[1])
    C[2] := F_KEY(P[2], P[3], C[0], C[1], SRC2.xmm[i].dword[2])
    C[3] := F_KEY(P[3], C[0], C[1], C[2], SRC2.xmm[i].dword[3])

    DEST.xmm[i].dword[0] := C[0]
    DEST.xmm[i].dword[1] := C[1]
    DEST.xmm[i].dword[2] := C[2]
    DEST.xmm[i].dword[3] := C[3]

DEST[MAXVL-1:VL] := 0
```

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Other Exceptions

VEX-encoded instructions, see Exceptions Type 6 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

EVEX-encoded instructions, see Exceptions Type E6 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

VSM4RNDS4—Performs Four Rounds of SM4 Encryption

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F2.0F38.W0 DA /r VSM4RNDS4 xmm1, xmm2, xmm3/m128	A	V/V	AVX AND SM4	Performs four rounds of SM4 encryption.
VEX.256.F2.0F38.W0 DA /r VSM4RNDS4 ymm1, ymm2, ymm3/m256	A	V/V	AVX AND SM4	Performs four rounds of SM4 encryption.
EVEX.128.F2.0F38.W0 DA /r VSM4RNDS4 xmm1, xmm2, xmm3/m128	B	V/V	AVX10 AND SM4	Performs four rounds of SM4 encryption.
EVEX.256.F2.0F38.W0 DA /r VSM4RNDS4 ymm1, ymm2, ymm3/m256	B	V/V	AVX10 AND SM4	Performs four rounds of SM4 encryption.
EVEX.512.F2.0F38.W0 DA /r VSM4RNDS4 zmm1, zmm2, zmm3/m512	B	V/V	AVX10 AND SM4	Performs four rounds of SM4 encryption.

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	N/A
B	FULLMEM	ModRM:reg (w)	EVEX.vvvv (r)	ModRM:r/m (r)	N/A

Description

The SM4RNDS4 instruction performs four rounds of SM4 encryption. The instruction operates on independent 128-bit lanes.

Additional details can be found at: https://tools.ietf.org/html/draft-ribose-cfrg-sm4-10.

See “VSM4KEY4—Perform Four Rounds of SM4 Key Expansion” for the sbox table.

Operation

```
// see the VSM4KEY4 instruction for the definition of ROL32, lower_t

define L_RND(dword):
    tmp := dword
    tmp := tmp ^ ROL32(dword, 2)
    tmp := tmp ^ ROL32(dword, 10)
    tmp := tmp ^ ROL32(dword, 18)
    tmp := tmp ^ ROL32(dword, 24)
    return tmp

define T_RND(dword):
    return L_RND(lower_t(dword))

define F_RND(X0, X1, X2, X3, round_key):
    return X0 ^ T_RND(X1 ^ X2 ^ X3 ^ round_key)
```

VSM4RNDS4 DEST, SRC1, SRC2

```
VL = (128, 256) // VEX versions
//or
VL = (128, 256, 512) // EVEX versions
KL := VL/128

for i in 0..KL-1:
    P[0] := SRC1.xmm[i].dword[0]
    P[1] := SRC1.xmm[i].dword[1]
    P[2] := SRC1.xmm[i].dword[2]
    P[3] := SRC1.xmm[i].dword[3]

    C[0] := F_RND(P[0], P[1], P[2], P[3], SRC2.xmm[i].dword[0])
    C[1] := F_RND(P[1], P[2], P[3], C[0], SRC2.xmm[i].dword[1])
    C[2] := F_RND(P[2], P[3], C[0], C[1], SRC2.xmm[i].dword[2])
    C[3] := F_RND(P[3], C[0], C[1], C[2], SRC2.xmm[i].dword[3])

    DEST.xmm[i].dword[0] := C[0]
    DEST.xmm[i].dword[1] := C[1]
    DEST.xmm[i].dword[2] := C[2]
    DEST.xmm[i].dword[3] := C[3]

DEST[MAXVL-1:VL] := 0
```

Flags Affected

None.

SIMD Floating-Point Exceptions

None.

Other Exceptions

VEX-encoded instructions, see Exceptions Type 6 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

EVEX-encoded instructions, see Exceptions Type E6 in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A.

CHAPTER 3
INTEL® AMX INSTRUCTION SET REFERENCE, A-Z

NOTES

The following Intel® AMX instructions have moved to the Intel® 64 and IA-32 Architectures Software Developer's Manual: LDTILECFG, STTILECFG, TDPBF16PS, TDPBSSD/TDPBSUD/TDPBUSD/TDPBUUD, TDPFP16PS, TILELOADD/TILELOADDT1, TILERELEASE, TILESTORED, and TILEZERO.

The Intel Advanced Matrix Extensions introductory material and helper functions will be maintained here, as well as in the Intel® 64 and IA-32 Architectures Software Developer's Manual, for the reader's convenience. For information on Intel AMX and the XSAVE feature set, and recommendations for system software, see the latest version of the Intel® 64 and IA-32 Architectures Software Developer's Manual.

3.1 INTRODUCTION

Intel® Advanced Matrix Extensions (Intel® AMX) is a new 64-bit programming paradigm consisting of two components: a set of 2-dimensional registers (tiles) representing sub-arrays from a larger 2-dimensional memory image, and an accelerator able to operate on tiles, the first implementation is called TMUL (tile matrix multiply unit).

An Intel AMX implementation enumerates to the programmer how the tiles can be programmed by providing a palette of options. Two palettes are supported; palette 0 represents the initialized state, and palette 1 consists of 8 KB of storage spread across 8 tile registers named TMM0..TMM7. Each tile has a maximum size of 16 rows x 64 bytes, (1 KB), however the programmer can configure each tile to smaller dimensions appropriate to their algorithm. The tile dimensions supplied by the programmer (rows and bytes_per_row, i.e., **colsb**) are metadata that drives the execution of tile and accelerator instructions. In this way, a single instruction can launch autonomous multi-cycle execution in the tile and accelerator hardware. The palette value (**palette_id**) and metadata are held internally in a tile related control register (TILECFG). The TILECFG contents will be commensurate with that reported in the palette_table (see "CPUID—CPU Identification" in Chapter 1 for a description of the available parameters).

Intel AMX is an extensible architecture. New accelerators can be added, or the TMUL accelerator may be enhanced to provide higher performance. In these cases, the state (TILEDATA) provided by tiles may need to be made larger, either in one of the metadata dimensions (more rows or colsb) and/or by supporting more tile registers (names). The extensibility is carried out by adding new palette entries describing the additional state. Since execution is driven through metadata, an existing Intel AMX binary could take advantage of larger storage sizes and higher performance TMUL units by selecting the most powerful palette indicated by CPUID and adjusting loop and pointer updates accordingly.

Figure 3-1 shows a conceptual diagram of the Intel AMX architecture. An Intel architecture host drives the algorithm, the memory blocking, loop indices and pointer arithmetic. Tile loads and stores and accelerator commands are sent to multi-cycle execution units. Status, if required, is reported back. Intel AMX instructions are synchronous in the Intel architecture instruction stream and the memory loaded and stored by the tile instructions is coherent with respect to the host's memory accesses. There are no restrictions on interleaving of Intel architecture and Intel AMX code or restrictions on the resources the host can use in parallel with Intel AMX (e.g., Intel AVX-512). There is also no architectural requirement on the Intel architecture compute capability of the Intel architecture host other than it supports 64-bit mode.

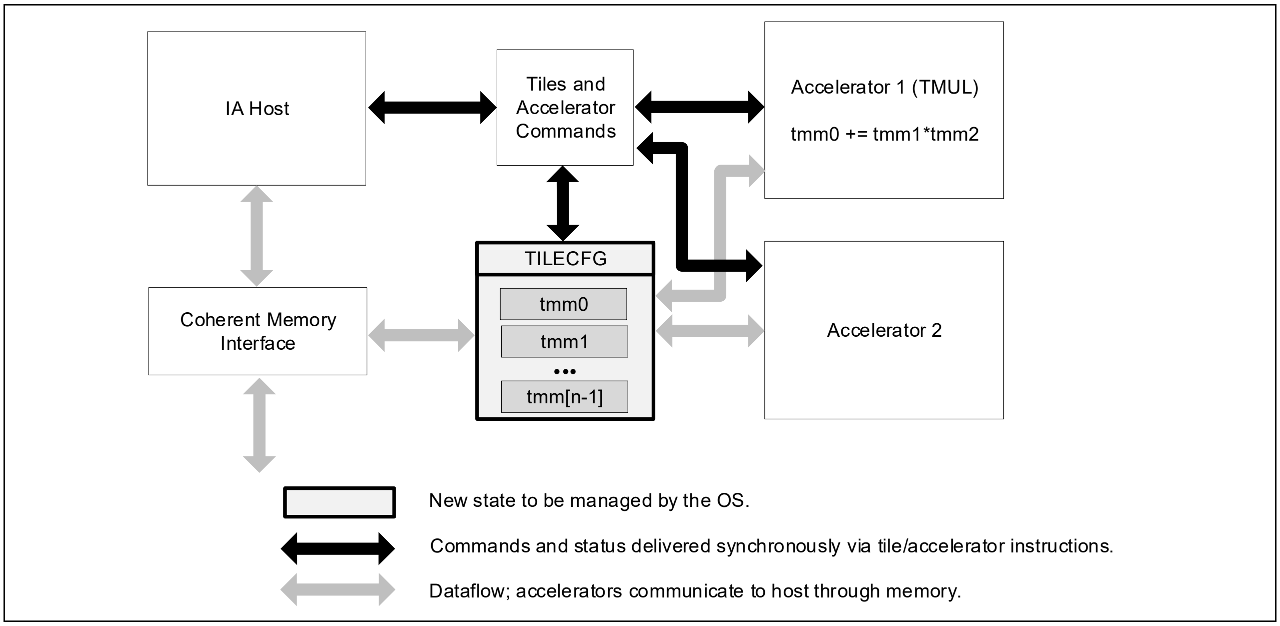


Figure 3-1. Intel® AMX Architecture

Intel AMX instructions use new registers and inherit basic behavior from Intel architecture in the same manner that Intel SSE and Intel AVX did. Tile instructions include loads and stores using the traditional Intel architecture register set as pointers. The TMUL instruction set (defined to be CPUID bits AMX-BF16 and AMX-INT8) only supports reg-reg operations.

TILECFG is programmed using the LDTILECFG instruction. The selected palette defines the available storage and general configuration while the rest of the memory data specifies the number of rows and column bytes for each tile. Consistency checks are performed to ensure the TILECFG matches the restrictions of the palette. A General Protection fault (#GP) is reported if the LDTILECFG fails consistency checks. A successful load of TILECFG with a palette_id other than 0 is represented in this document with TILES_CONFIGURED = 1. When the TILECFG is initialized (palette_id = 0), it is represented in the document as TILES_CONFIGURED = 0. Nearly all Intel AMX instructions will generate a #UD exception if TILES_CONFIGURED is not equal to 1; the exceptions are those that do TILECFG maintenance: LDTILECFG, STTILECFG and TILERELEASE.

If a tile is configured to contain M rows by N column bytes, LDTILECFG will ensure that the metadata values are appropriate to the palette (e.g., that $M \leq 16$ and $N \leq 64$ for palette 1). The four M and N values can all be different as long as they adhere to the restrictions of the palette. Further dynamic checks are done in the tile and the TMUL instruction set to deal with cases where a legally configured tile may be inappropriate for the instruction operation. Tile registers can be set to ‘invalid’ by configuring the rows and colsb to ‘0’.

Tile loads and stores are strided accesses from the application memory to packed rows of data. Algorithms are expressed assuming row major data layout. Column major users should translate the terms according to their orientation.

TILELOAD* and TILESTORE* instructions are restartable and can handle (up to) 2*rows page faults per instruction. Restartability is provided by a **start_row** parameter in the TILECFG register.

The TMUL unit is conceptually a grid of fused multiply-add units able to read and write tiles. The dimensions of the TMUL unit (tmul_maxk and tmul_maxn) are enumerated similar to the maximum dimensions of the tiles (see “CPUID—CPU Identification” in Chapter 1 for details).

The matrix multiplications in the TMUL instruction set compute C[M][N] += A[M][K] * B[K][N]. The M, N, and K values will cause the TMUL instruction set to generate a #UD exception if the dimensions do not match for matrix multiply or do not match the palette.

In Figure 3-2, the number of rows in tile B matches the K dimension in the matrix multiplication pseudocode. K dimensions smaller than that enumerated in the TMUL grid are also possible and any additional computation the TMUL unit can support will not affect the result.

The number of elements specified by colsb of the B matrix is also less than or equal to tmul_maxn. Any remaining values beyond that specified by the metadata will be set to zero.

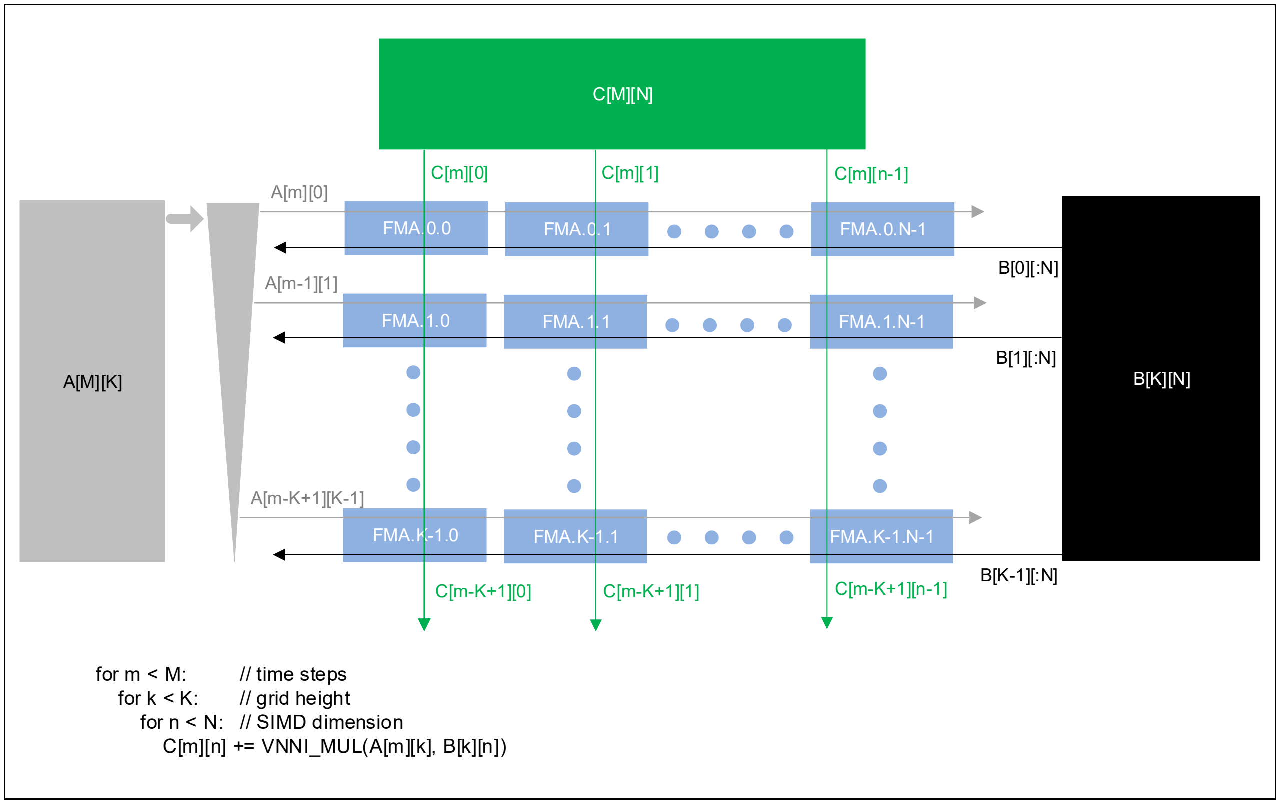


Figure 3-2. The TMUL Unit

The XSAVE feature sets supports context management of the new state defined for Intel AMX. This support is described in Section 3.2.

3.1.1 Tile Architecture Details

The supported parameters for the tile architecture are reported via CPUID; this includes information about how the number of tile registers (max_names) can be configured (the palette). Configuring the tile architecture is intended to be done once when entering a region of tile code using the LDTILECFG instruction specifying the selected palette and describing in detail the configuration for each tile. Incorrect assignments will result in a General Protection fault (#GP). Successful LDTILECFG initializes (zeroes) TILEDATA.

Exiting a tile region is done with the TILERELEASE instruction. It takes no parameters and invalidates all tiles (indicating that the data no longer needs any saving or restoring). Essentially, it is an optimization of LDTILECFG with an implicit palette of 0.

For applications that execute consecutive Intel AMX regions with differing configurations, TILERELEASE is not required between them since the second LDTILECFG will clear all the data while loading the new configuration. There is no instruction set support for automatic nesting of tile regions, though with sufficient effort software can accomplish this by saving and restoring TILEDATA and TILECFG either through the XSAVE architecture or the Intel AMX instructions.

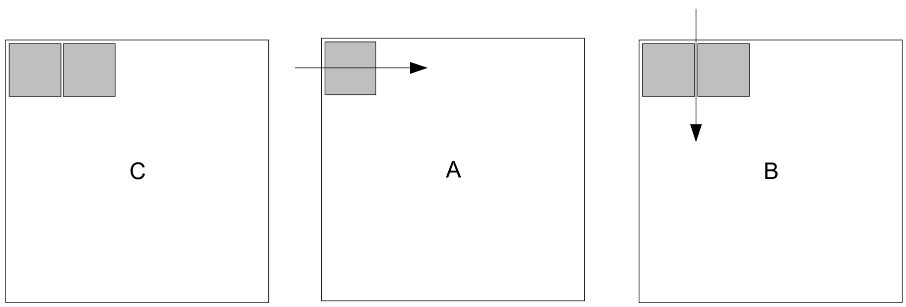
The tile architecture boots in its INIT state, with TILECFG and TILEDATA set to zero. A successfully executing LDTILECFG instruction to a non-zero palette sets the TILES_CONFIGURED=1, indicating the TILECFG is not in the INIT state. The TILERELEASE instruction sets TILES_CONFIGURED = 0 and initializes (zeroes) TILEDATA.

To facilitate handling of tile configuration data, there is a STTILECFG instruction. If the tile configuration is in the INIT state (TILES_CONFIGURED == 0), then STTILECFG will write 64 bytes of zeros. Otherwise STTILECFG will store the TILECFG to memory in the format used by LDTILECFG.

3.1.2 TMUL Architecture Details

The supported parameters for the TMUL architecture are reported via CPUID; see “CPUID—CPU Identification” in Chapter 1, page 1-26, for details. These parameters include a maximum height (**tmul_maxk**) and a maximum SIMD dimension (**tmul_maxn**). The metadata that accompanies the srcdest, src1 and src2 tiles to the TMUL unit will be dynamically checked to see that they match the TMUL unit support for the data type and match the requirements of a meaningful matrix multiplication.

Figure 3-3 shows an example of the inner loop of an algorithm of using the TMUL architecture to compute a matrix multiplication. In this example, we use two result tiles, tmm0 and tmm1, from matrix C to accumulate the intermediate results. One tile from the A matrix (tmm2) is re-used twice as we multiply it by two tiles from the B matrix. The algorithm then advances pointers to load a new A tile and two new B tiles from the directions indicated by the arrows. An outer loop, not shown, adjusts the pointers for the C tiles.




```
   LDTILECFG [rax]
// assume some outer loops driving the cache tiling (not shown)
{
   TILELOADD tmm0, [rsi+rdi]     // srcdst, RSI points to C, RDI is strided value
   TILELOADD tmm1, [rsi+rdi+N]   // second tile of C, unrolling in SIMD dimension N
   MOV r14, 0
LOOP:
   TILELOADD tmm2, [r8+r9]       // src2 is strided load of A, reused for 2 TMUL instr.
   TILELOADD tmm3, [r10+r11]     // src1 is strided load of B
   TDPBUSD   tmm0, tmm2, tmm3  // update left tile of C
   TILELOADD tmm3, [r10+r11+N]   // src1 loaded with B from next rightmost tile
   TDPBUSD   tmm1, tmm2, tmm3  // update right tile of C
   ADD r8, K                 // update pointers by constants known outside of loop
   ADD r10, K*r11
   ADD r14, K
   CMP r14, LIMIT
   JNE LOOP

   TILESTORED [rsi+rdi],   tmm0  // update the C matrix in memory
   TILESTORED [rsi+rdi+M], tmm1
 } // end of outer loop

   TILERELEASE        // return tiles to INIT state
```

Figure 3-3. Matrix Multiply C+= A*B

3.1.3 Handling of Tile Row and Column Limits

Intel AMX operations will zero any rows and any columns beyond the dimensions specified by TILECFG. Tile operations will zero the data beyond the configured number of column bytes as each row is written. For example, with 64-byte rows and a tile configured with 10 rows and 48 columns, an operation writing dword elements would write each of the first 10 rows with 48 bytes of output/result data and zero the remaining 16 bytes in each row. Tile operations also fully zero any rows after the first 10 configured rows. When using a 1 KByte tile with 64-byte rows, there would be 16 rows, so in this example, the last 6 rows would also be zeroed.

Intel AMX instructions will always obey the metadata on reads and the zeroing rules on writes, and so a subsequent XSAVE would see zeros in the appropriate locations. Tiles that are not written by Intel AMX instructions between XRSTOR and XSAVE will write back with the same image they were loaded with regardless of the value of TILECFG.

3.1.4 Exceptions and Interrupts

Tile instructions are restartable so that operations that access strided memory can restart after page faults. To support restarting instructions after these events, the instructions store information in the **TILECFG.start_row** register. TILECFG.start_row indicates the row that should be used for restart; i.e., it indicates **next row after** the rows that have already been successfully loaded (on a TILELOAD) or written to memory (on a TILESTORE) and prevents repeating work that was successfully done.

The TMUL instruction set is not sensitive to the TILECFG.start_row value; this is due to there not being TMUL instructions with memory operands or any restartable faults.

3.2 OPERAND RESTRICTIONS

Floating-point exceptions, denormal handling, and floating-point rounding: some of the Intel AMX instructions operate on floating-point values. These instructions all function as if floating-point exceptions are masked, and use the round-to-nearest-even (RNE) rounding mode. They also do not set any of the floating-point exception flags in MXCSR. Table 3-1 describes the treatment of denormal inputs and outputs for Intel AMX operations.

Table 3-1. Intel® AMX Treatment of Denormal Inputs and Outputs

Data Type	Denormal Input	Denormal Output
FP16	Allowed	N/A
FP32	Treated as zero	Flushed to zero
BF16	Treated as zero	N/A

3.3 IMPLEMENTATION PARAMETERS

The parameters are reported via CPUID leaf 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
```

3.4 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_tileconfig_start():
    tilecfg.start_row :=0



define zero_all_tile_data():
  if XCR0[TILEDATA]:
      b := CPUID(0xD,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


define fma32(acc, x, y, daz, ftz, sae, rc):
   // sae = suppress all exceptions. if 1= no exceptions
   // are raised or denoted in mxcsr
   if daz and denormal(x):
      x = 0
   if daz and denormal(y):
      y = 0
   if daz and denormal(acc):
      acc = 0
   // traditional infinite precision fma
   // using sae and rounding control from rc
   v = (x*y) + acc
   if ftz and denormal(v):
      v = 0
   return v
```

```
define convert_int128_to_fp32( in, type1, type2 ):
   // int128 is an integer value
   // type1 and type2 can be HF8, which stands for E4M3, or BF8, which stands for E5M2

   if (in == 0):
   return 0

   sign = in[127]
   magnitude[127:0] = sign ? -in : in       // get absolute value
   Jbit_position = 126
   while (magnitude[126]==0):
      Jbit_position--    // get Jbit index
      magnitude << 1     // normalize in to the left

   // Jbit index is 126
   // Lbit index is 126-23=103
   // Gbit index is then 102
   sticky = OR(magnitude[101:0])      // get sticky value
   Gbit = magnitude[102]              // get Gbit
   Lbit = magnitude[103]              // get Lbit
   RndAdd1 = Gbit & (Lbit | sticky)   // RNE parameter

   Mantissa[24:0] = magnitude >> 103  // get mantissa[24:0] (Jbit in Mantissa[23])

   RndMantissa = Mantissa + RndAdd1

   Ovf = RndMantissa >> 24      // check if rounded mantissa overflow

   match (type1, type2):        // get exponent factor
      case [BF8 BF8] : factor=32      // 2*16
      case [HF8 HF8] : factor=18      // 2*9
      case _ : factor=25              // 16+9

   // Destination exponent = BIAS+(Jbit_position+Ovf)-factor
   exp = 127 + Jbit_position - factor + Ovf       // set destination exponent
   frac = RndMantissa & 0x7FFFFF                  // set destination fraction

   res = sign<<31        // set sign in bit[32]
   res |= exp << 23      // set exp in bits[30:23]
   res |= frac           // get frac in bits[22:0]

   return res            // get fp32 format, RNE
```

```
define convert_fp8_to_int64(x, type):
   // The x parameter is the data,
   //the type parameter indicates the data format: BF8 for E5M2 FP8 or HF8 for E4M3 FP8.
   if *type is bf8*:
      return convert_bf8_to_int64(x)
   else *if type is HF8*:
      return convert_hf8_to_int64(x)



define convert_fp32_to_bfloat16(x):
   IF x is zero or denormal:
      dest[15] := x[31] // sign preserving zero (denormal go to zero)
      dest[14:0] := 0
   ELSE IF x is infinity:
      dest[15:0] := x[31:16]
   ELSE IF x is nan:
      dest[15:0] := x[31:16] // truncate and set msb of the mantisa force qnan
      dest[6] := 1
   ELSE // normal number
      lsb := x[16]
      rounding_bias := 0x00007FFF + lsb
      temp[31:0] := x[31:0] + rounding_bias // integer add
      dest[15:0] := temp[31:16]
return dest



define convert_bf8_to_int64( in ):
   // BF8 stands for E5M2 FP8
   sign = ( in & 0x80 ) >> 7
   exp = ( in & 0x7C ) >> 2
   frac = ( in & 0x03 )
   mant = (exp==0) ? frac : (frac | 0x4) // set Jbit
   e_count = (exp==0) ? 0 : exp - 1 // set integer alignment shift count

   // convert E5M2 FP8 number into an integer 64-bit number
   magnitude.int64= mant << e_count
   res = sign ? -magnitude.int64: magnitude.int64

   return res // value is 2^16*in
```

```
define convert_hf8_to_int64( in ):
   // HF8 stands for E4M3 FP8
   sign = ( in & 0x80 ) >> 7
   exp = ( in & 0x78 ) >> 3
   frac = ( in & 0x07 )
   mant = (exp==0) ? frac : (frac | 0x8) // set Jbit
   e_count = (exp=0) ? 0 : exp - 1 // set integer alignment shift count

   // convert E4M3 FP8 number into an integer 64-bit number
   magnitude.int64 = mant << e_count
   res = sign ? -magnitude.int64 : magnitude.int64

   return res // value is 2^9*in
```

3.5 NOTATION

Instructions described in this chapter follow the general documentation convention established in *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A*. Additionally, Intel® Advanced Matrix Extensions use notation conventions as described below.

In the instruction encoding boxes, **sibmem** is used to denote an encoding where a MODRM byte and SIB byte are used to indicate a memory operation where the base and displacement are used to point to memory, and the index register (if present) is used to denote a stride between memory rows. The index register is scaled by the sib.scale field as usual. The base register is added to the displacement, if present.

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 MODMR.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.

NOTE

Historically the Intel® 64 and IA-32 Architectures Software Developer's Manual only specified the MODRM.REG field restrictions with the notation /0 ... /7 and did not specify restrictions on the MODRM.MOD and MODRM.RM fields in the encoding boxes.

3.6 EXCEPTION CLASSES

Alignment exceptions: The Intel AMX instructions that access memory will never generate #AC exceptions.

Table 3-2. Intel® AMX Exception Classes

Class	Description
AMX-E1	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.
	• #GP based on palette and configuration checks (see pseudocode). • #GP if the memory address is in a non-canonical form.
	• #SS(0) if the memory address referencing the SS segment is in a non-canonical form.
	• #PF if a page fault occurs.
AMX-E1-EVEX	All of AMX-E1 exceptions. Additionally: • #UD if EVEX.z ≠ 0b0 // P2[7]. • #UD if EVEX.LL' ≠ 0b00 // P2[6:5]. • #UD if EVEX.b ≠ 0b0 // P2[4]. • #UD if EVEX.aaa ≠ 0b000 // P2[2:0]. • #UD if EVEX.VVVV ≠ 0b1111 // P1[6:3]. • #UD if EVEX.V' ≠ 0b1 // P2[3].
AMX-E2	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.
	• #GP if the memory address is in a non-canonical form.
	• #SS(0) if the memory address referencing the SS segment is in a non-canonical form.
	• #PF if a page fault occurs.
AMX-E2-EVEX	All of AMX-E2 exceptions. Additionally: • #UD if EVEX.z ≠ 0b0 // P2[7]. • #UD if EVEX.LL' ≠ 0b00 // P2[6:5]. • #UD if EVEX.b ≠ 0b0 // P2[4]. • #UD if EVEX.aaa ≠ 0b000 // P2[2:0]. • #UD if EVEX.VVVV ≠ 0b1111 // P1[6:3]. • #UD if EVEX.V' ≠ 0b1 // P2[3].
AMX-E3	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #UD if not using SIB addressing. • #UD if TILES_CONFIGURED == 0. • #UD if tsrc or tdest are not valid tiles. • #UD if tsrc/tdest are ≥ palette_table[tilecfg.palette_id].max_names. • #UD if tsrc.colbytes mod 4 ≠ 0 OR tdest.colbytes mod 4 ≠ 0. • #UD if tilecfg.start_row ≥ tsrc.rows OR tilecfg.start_row ≥ tdest.rows.
	• #GP if the memory address is in a non-canonical form.
	• #SS(0) if the memory address referencing the SS segment is in a non-canonical form.
	• #PF if any memory operand causes a page fault.
	• #NM if XFD[18] == 1.

Table 3-2. Intel® AMX Exception Classes (Continued)

Class	Description
AMX-E3-EVEX	All of AMX-E3 exceptions. Additionally: • #UD if EVEX.z ≠ 0b0 // P2[7]. • #UD if EVEX.LL’ ≠ 0b00 // P2[6:5]. • #UD if EVEX.b ≠ 0b0 // P2[4]. • #UD if EVEX.aaa ≠ 0b000 // P2[2:0]. • #UD if EVEX.VVVV ≠ 0b1111 // P1[6:3]. • #UD if EVEX.V’ ≠ 0b1 // P2[3].
AMX-E4	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if srcdest == src1 OR src1 == src2 OR srcdest == src2. • #UD if TILES_CONFIGURED == 0. • #UD if srcdest.colbytes mod 4 ≠ 0. • #UD if src1.colbytes mod 4 ≠ 0. • #UD if src2.colbytes mod 4 ≠ 0. • #UD if srcdest/src1/src2 are not valid tiles. • #UD if srcdest/src1/src2 are ≥ palette_table[tilecfg.palette_id].max_names. • #UD if srcdest.colbytes ≠ src2.colbytes. • #UD if srcdest.rows ≠ src1.rows. • #UD if src1.colbytes / 4 ≠ src2.rows. • #UD if srcdest.colbytes > tmul_maxn. • #UD if src2.colbytes > tmul_maxn. • #UD if src1.colbytes/4 > tmul_maxk. • #UD if src2.rows > tmul_maxk.
	• #NM if XFD[18] == 1.
AMX-E5	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111. • #UD if TILES_CONFIGURED == 0. • #UD if tdest is not a valid tile. • #UD if tdest is ≥ palette_table[tilecfg.palette_id].max_names.
	• #NM if XFD[18] == 1.
AMX-E6	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if VVVV ≠ 0b1111.

Table 3-2. Intel® AMX Exception Classes (Continued)

Class	Description
AMX-E7-EVEX	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if XCR0[7:5] ≠ 0b111. • #UD if XCR0[2:1] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if TILES_CONFIGURED == 0. • #UD if tsrc is not a valid tile (i.e., not configured). • #UD if tsrc is not a valid tile name for configured palette. • #UD if tsrc.colsb % 4 != 0. • #UD if EVEX.z ≠ 0b0 // P2[7]. • #UD if EVEX.LL' ≠ 0b10 // P2[6:5]. • #UD if EVEX.b ≠ 0b0 // P2[4]. • #UD if EVEX.aaa ≠ 0b000 // P2[2:0]. • #UD if EVEX.u ≠ 0b1 // P1[2]. • #UD if EVEX.V' ≠ 0b1 // P2[3]. • #UD if EVEX.VVVV ≠ 0b1111 // P1[6:3].
	• #NM if CR0[3] == 1 // TS. • #NM if XFD[18] == 1.
AMX-E8-EVEX	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if XCR0[7:5] ≠ 0b111. • #UD if XCR0[2:1] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if TILES_CONFIGURED == 0. • #UD if tsrc is not a valid tile (i.e., not configured). • #UD if tsrc is not a valid tile name for configured palette. • #UD if tsrc.colsb % 4 ≠ 0. • #UD if EVEX.z ≠ 0b0 // P2[7]. • #UD if EVEX.LL' ≠ 0b10 // P2[6:5]. • #UD if EVEX.b ≠ 0b0 // P2[4]. • #UD if EVEX.aaa ≠ 0b000 // P2[2:0]. • #UD if EVEX.u ≠ 0b1 // P1[2]. • #UD if EVEX.V' ≠ 0b1 // P2[3].
	• #NM if CR0[3] == 1. // TS • #NM if XFD[18] == 1.
AMX-E9	• #UD if preceded by LOCK, 66H, F2H, F3H or REX prefixes. • #UD if CR4.OSXSAVE ≠ 1. • #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if palette expresses state not supported by XCR0. • #UD if TILES_CONFIGURED == 0. • #UD if VEX.VVVV ≠ 0b1111. • #UD if src1/tdest is not a valid tile. • #UD if src1/tdest.colbytes % 4 ≠ 0. • #UD if tsrc.rows ≠ tdst.colsb/4. • #UD if tsrc.colsb/4 ≠ tdst.rows.
	• #NM if XFD[18] == 1.

Table 3-2. Intel® AMX Exception Classes (Continued)

Class	Description
AMX-E10	• #UD if XCR0[18:17] ≠ 0b11. • #UD if IA32_EFER.LMA ≠ 1 OR CS.L ≠ 1. • #UD if palette expresses state not supported by XCR0 (future expansion). • #UD if TILES_CONFIGURED == 0. • #UD if tsrcdest == tsrc1 OR tsrc1 == tsrc2 OR tsrcdest == tsrc2. • #UD if tsrc2.colbytes % 4 ≠ 0. • #UD if tsrc1.colbytes % 4 ≠ 0. • #UD if any of tsrdest, tsrc1, or tsrc2 are not valid tiles (i.e., not configured). • #UD if any of tsrdest, tsrc1, or tsrc2 are not valid tile names for configured palette. • #UD if tsrcdest.colbytes ≠ tsrc2.colbytes. • #UD if tsrcdest.rows ≠ tsrc1.colbytes/4. • #UD if tsrc1.rows ≠ tsrc2.rows. • #UD if tsrcdest.colbytes > tmul_maxn. • #UD if tsrc2.colbytes > tmul_maxn. • #UD if tsrc1.rows > tmul_maxk. • #UD if tsrc2.rows > tmul_maxk.
	• #NM if XFD[18] == 1.

3.7 INSTRUCTION SET REFERENCE

TCVTROWD2PS—Tile Move Row and Convert INT32 to Single Precision

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.F3.0F38.W0 4A 11:rrr:bbb TCVTROWD2PS zmm1, tmm2, r32	A	V/N.E.	AMX-AVX512	Move a row selected by r32 from tmm2 to zmm1, converting the int32 source elements to fp32.
EVEX.512.F3.0F3A.W0 07 11:rrr:bbb /ib TCVTROWD2PS zmm1, tmm2, imm8	B	V/N.E.	AMX-AVX512	Move a row selected by imm8 from tmm2 to zmm1, converting the int32 source elements to fp32.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	EVEX.vvvv (r)	N/A
B	N/A	ModRM:reg (w)	ModRM:r/m (r)	imm8 (r)	N/A

Description

This instruction moves a row from a tile register to a zmm destination register, converting the int32 source elements to FP32. The row of the tile is selected by an imm8, or a 32-bit general-purpose register. If the row indicated is outside the configured range, the destination register is zeroed.

No SIMD exceptions are generated. Rounding is done as if MXCSR.RC=RNE. Embedded rounding is not supported. MXCSR is neither consulted nor updated.

Any attempt to execute the TCVTROWD2PS instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

TCVTROWD2PS zdest, tsrc, r32

```
VL = (512)
VL_bytes := VL >> 3 // bits to bytes
row_index := r32&0xf
if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... (VL_bytes/4)-1:
    if i >= tsrc.colsb/4:
        zdest.dword[i] := 0
    else:
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(RNE);
        zdest.f32[i] := Convert_Integer_To_Single_Precision_Floating_Point(tsrc.row[row_index].dword[i]);

EXIT_AND_SKIP:
zero_tileconfig_start()
```

TCVTROWD2PS zdest, tsrc, imm8

```
VL = (512)
VL_bytes = VL >> 3 //bits to bytes
row_index := imm8&0xf

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 … (VL_bytes/4)-1:
    if i >= tsrc.colsb/4:
        zdest.dword[i] := 0
    else:
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(RNE);
        zdest.f32[i] := Convert_Integer_To_Single_Precision_Floating_Point(tsrc.row[row_index].dword[i]);

EXIT_AND_SKIP:
zero_tileconfig_start()
```

Flags Affected

None.

Exceptions

Instruction	Exception Type
TCVTROWD2PS zdest, tsrc, r32	AMX-E8-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWD2PS zdest, tsrc, imm8	AMX-E7-EVEX. See Section 3.6, “Exception Classes,” for details.

TCVTROWPS2BF16[H,L]—Tile Move Row and Convert FP32 Elements to BF16 Elements

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.F2.0F38.W0 6D 11:rrr:bbb TCVTROWPS2BF16H zmm1, tmm2, r32	A	V/N.E.	AMX-AVX512	Move a row selected by r32 from tmm2 to zmm1, converting the fp32 source elements to bf16 and place the resulting bf16 elements in the high 16 bits within each dword.
EVEX.512.F2.0F3A.W0 07 11:rrr:bbb /ib TCVTROWPS2BF16H zmm1, tmm2, imm8	B	V/N.E.	AMX-AVX512	Move a row selected by imm8 from tmm2 to zmm1, converting the fp32 source elements to bf16 and place the resulting bf16 elements in the high 16 bits within each dword.
EVEX.512.F3.0F38.W0 6D 11:rrr:bbb TCVTROWPS2BF16L zmm1, tmm2, r32	A	V/N.E.	AMX-AVX512	Move a row selected by r32 from tmm2 to zmm1, converting the fp32 source elements to bf16 and place the resulting bf16 elements in the low 16 bits within each dword.
EVEX.512.F3.0F3A.W0 77 11:rrr:bbb /ib TCVTROWPS2BF16L zmm1, tmm2, imm8	B	V/N.E.	AMX-AVX512	Move a row selected by imm8 from tmm2 to zmm1, converting the fp32 source elements to bf16 and place the resulting bf16 elements in the low 16 bits within each dword.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	EVEX.vvvv (r)	N/A
B	N/A	ModRM:reg (w)	ModRM:r/m (r)	imm8 (r)	N/A

Description

This instruction moves a row from a tile register to a zmm destination register, converting the FP32 source elements to BF16. The row of the tile is selected by an imm8, or a 32-bit general-purpose register. If the row indicated is outside the configured range, the destination register is zeroed.

TCVTROWPS2BF16H places the resulting BF16 elements in the high 16 bits within each dword, while TCVTROWPS2BF16L places them in the low 16 bits.

No SIMD exceptions are generated. Rounding is done as if MXCSR.RC=RNE. Embedded rounding is not supported. MXCSR is neither consulted nor updated.

Any attempt to execute the TCVTROWPS2BF16[H,L] instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

TCVTROWPS2BF16[H,L] zdest, tsrc, imm8

```
VL = (512)
VL_bytes := VL>>3 // bits to bytes
row_index := imm8&0xf

if instruction is TCVTROWPS2BF16H:
    pos := 1
    zeropos := 0
else:
    pos := 0
    zeropos := 1

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... (VL_bytes/4)-1:
    if i >= tsrc.colsb/4:
        zdest.dword[i] := 0
    else:
        zdest.word[2*i+zeropos] := 0
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(RNE);
        zdest.bf16[2*i+pos] := convert_fp32_to_bfloat16(tsrc.row[row_index].fp32[i]):

EXIT_AND_SKIP:
zero_tileconfig_start()
```

TCVTROWPS2BF16[H,L] zdest, tsrc, r32

```
VL = (512)
VL_bytes := VL >> 3 // bits to bytes
row_index := r32&0xf

if instruction is TCVTROWPS2BF16H:
    pos := 1
    zeropos := 0
else:
    pos := 0
    zeropos := 1

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... (VL_bytes/4)-1:
    if i >= tsrc.colsb/4:
        zdest.dword[i] := 0
else:
    zdest.word[2*i+zeropos] := 0
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(RNE);
        zdest.bf16[2*i+pos] := convert_fp32_to_bfloat16(tsrc.row[row_index].fp32[i]):

EXIT_AND_SKIP:
zero_tileconfig_start()
```

Flags Affected

None.

Exceptions

Instruction	Exception Type
TCVTROWPS2BF16H zmm1, tmm2, r32	AMX-E8-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWPS2BF16H zmm1, tmm2, imm8	AMX-E7-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWPS2BF16L zmm1, tmm2, r32	AMX-E8-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWPS2BF16L zmm1, tmm2, imm8	AMX-E7-EVEX. See Section 3.6, “Exception Classes,” for details.

TCVTROWPS2PH[H,L]—Tile Move Row and Convert Single Precision to FP16

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.NP.0F38.W0 6D 11:rrr:bbb TCVTROWPS2PHH zmm1, tmm2, r32	A	V/N.E.	AMX-AVX512	Move a row selected by r32 from tmm2 to zmm1, converting the fp32 source elements to fp16 and place the resulting fp16 elements in the high 16 bits within each dword.
EVEX.512.NP.0F3A.W0 07 11:rrr:bbb /ib TCVTROWPS2PHH zmm1, tmm2, imm8	B	V/N.E.	AMX-AVX512	Move a row selected by imm8 from tmm2 to zmm1, converting the fp32 source elements to fp16 and place the resulting fp16 elements in the high 16 bits within each dword.
EVEX.512.66.0F38.W0 6D 11:rrr:bbb TCVTROWPS2PHL zmm1, tmm2, r32	A	V/N.E.	AMX-AVX512	Move a row selected by r32 from tmm2 to zmm1, converting the fp32 source elements to fp16 and place the resulting fp16 elements in the low 16 bits within each dword.
EVEX.512.F2.0F3A.W0 77 11:rrr:bbb /ib TCVTROWPS2PHL zmm1, tmm2, imm8	B	V/N.E.	AMX-AVX512	Move a row selected by imm8 from tmm2 to zmm1, converting the fp32 source elements to fp16 and place the resulting fp16 elements in the low 16 bits within each dword.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	EVEX.vvvv (r)	N/A
B	N/A	ModRM:reg (w)	ModRM:r/m (r)	imm8 (r)	N/A

Description

This instruction moves a row from a tile register to a zmm destination register, converting the FP32 source elements to FP16. The row of the tile is selected by an imm8, or a 32-bit general-purpose register. If the row indicated is outside the configured range, the destination register is zeroed.

TCVTROWPS2PHH places the resulting FP16 elements in the high 16 bits within each dword, while TCVTROWPS2PHL places them in the low 16 bits.

No SIMD exceptions are generated. Rounding is done as if MXCSR.RC=RNE. Embedded rounding is not supported. MXCSR is neither consulted nor updated.

Input FP32 denormals become FP16 zeros on outputs. This instruction can produce FP16 denormal outputs. (MXCSR.FTZ only controls FP32 and FP64 denormal outputs).

Any attempt to execute the TCVTROWPS2PH[H,L] instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

TCVTROWPS2PH[H,L] zdest, tsrc, imm8

```
VL = (512)
VL_bytes := VL>>3 // bits to bytes
row_index := imm8&0xf

if instruction is TCVTROWPS2PHH:
    pos := 1
    zeropos := 0
else:
    pos := 0
    zeropos := 1

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... (VL_bytes/4)-1:
    if i >= tsrc.colsb/4:
        zdest.dword[i] := 0
    else:
        zdest.word[2*i+zeropos] := 0
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(RNE);
        zdest.f16[2*i+pos] := vCvt_s2h(tsrc.row[row_index].fp32[+i]);

EXIT_AND_SKIP:
zero_tileconfig_start()
```

TCVTROWPS2PH[H,L] zdest, tsrc, r32

```
VL = (512)
VL_bytes := VL>>3 // bits to bytes
row_index := r32&0xf

if instruction is TCVTROWPS2PHH:
    pos := 1
    zeropos := 0
else:
    pos := 0
    zeropos := 1

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... (VL_bytes/4)-1:
    if i >= tsrc.colsb/4:
        zdest.dword[i] := 0
    else:
        zdest.word[2*i+zeropos] := 0
        SET_ROUNDING_MODE_FOR_THIS_INSTRUCTION(RNE);
        zdest.f16[2*i+pos] := vCvt_s2h(tsrc.row[row_index].fp32[+i]);

EXIT_AND_SKIP:
zero_tileconfig_start()
```

Flags Affected

None.

Exceptions

Instruction	Exception Type
TCVTROWPS2PHH zmm1, tmm2, r32	AMX-E8-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWPS2PHH zmm1, tmm2, imm8	AMX-E7-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWPS2PHL zmm1, tmm2, r32	AMX-E8-EVEX. See Section 3.6, “Exception Classes,” for details.
TCVTROWPS2PHL zmm1, tmm2, imm8	AMX-E7-EVEX. See Section 3.6, “Exception Classes,” for details.

TDP[B,H,BH,HB]F8PS—Dot Product of FP8 Tiles Accumulated into Packed Single Precision Tile

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.NP.MAP5.W0 FD 11:rrr:bbb TDPBF8PS tmm1, tmm2, tmm3	A	V/N.E.	AMX-FP8	Matrix multiply E5M2 FP8 elements from tmm2 and tmm3 and accumulate the packed single precision elements in tmm1.
VEX.128.F2.MAP5.W0 FD 11:rrr:bbb TDPBHF8PS tmm1, tmm2, tmm3	A	V/N.E.	AMX-FP8	Matrix multiply E5M2 FP8 elements from tmm2 and E4M3 FP8 elements from tmm3 and accumulate the packed single precision elements in tmm1.
VEX.128.F3.MAP5.W0 FD 11:rrr:bbb TDPHBF8PS tmm1, tmm2, tmm3	A	V/N.E.	AMX-FP8	Matrix multiply E4M3 FP8 elements from tmm2 and E5M2 FP8 elements from tmm3 and accumulate the packed single precision elements in tmm1.
VEX.128.66.MAP5.W0 FD 11:rrr:bbb TDPHF8PS tmm1, tmm2, tmm3	A	V/N.E.	AMX-FP8	Matrix multiply E4M3 FP8 elements from tmm2 and tmm3 and accumulate the packed single precision elements in tmm1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (r, w)	ModRM:r/m (r)	VEX.vvvv (r)	N/A

Description

These instructions compute the dot product of E5M2 FP8 or E4M3 FP8 values, accumulating the results into single precision (FP32). The input elements can be E5M2 FP8 or E4M3 FP8 format. These instructions have three tile operands, one source/dest accumulator operand, and two source operands, src1 and src2. The src1 and src2 operands can be of E5M2 FP8 or E4M3 FP8 type independently, and the source/dest operand is always FP32.

TDPBF8PS is the dot product of a E5M2 FP8 value (src1) by a E5M2 FP8 value (src2) accumulating into a Single Precision (FP32) source/dest.

TDPHF8PS is the dot product of an E4M3 FP8 value (src1) by an E4M3 FP8 value (src2) accumulating into a Single Precision (FP32) source/dest.

TDPBHF8PS is the dot product of a E5M2 FP8 value (src1) by an E4M3 FP8 value (src2) accumulating into a Single Precision (FP32) source/dest.

TDPHBF8PS is the dot product of an E4M3 FP8 value (src1) by a E5M2 FP8 value (src2) accumulating into a Single Precision (FP32) source/dest.

All operations treat the presence of an input NaN value (in src and srcdest) as resulting in an FP32 QNaN_Indefinite response in the destination of that operation, where a FP32 QNaN indefinite value is 0xFFC0.0000:

- Sign='1
- Exp[7:0]= 0xFF
- Fraction[22:0]=0x40.0000

The tile registers specified must be distinct from one another, no repeats.

Rounding is always RNE. For the inputs, DAZ==0 is assumed, for the output FTZ==1 is assumed.

MXCSR is neither consulted nor updated, no exceptions are raised or denoted.

Any attempt to execute the TDP[B,H,BH,HB]F8PS instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

```
TDP[B,H,BH,HB]F8PS tsrcdest, tsrc1, tsrc2
// C = m x n (tsrcdest), A = m x k (tsrc1), B = k x n (tsrc2)
# src1 and src2 elements are 4-tuples of E5M2 FP8 or E4M3 FP8
elements_src1 = tsrc1.colsb / 4
elements_dest = tsrcdest.colsb / 4
elements_temp1 = tsrcdest.colsb / 4

if *tsrc1 is of type E5M2 FP8*:
    type1 = bf8
else: *if tsrc1 is of type E4M3 FP8*
    type1 = hf8

if *tsrc2 is of type E5M2 FP8*:
    type2 = bf8
else: *if tsrc2 is of type E4M3 FP8*
    type2 = hf8

for m in 0 ... tsrcdest.rows-1:
    temp1[ 0 ... elements_temp1-1 ] = 0
    for k in 0 ... elements_src1-1:
        for n in 0 ... elements_dest-1:
            // FP32 MUL with DAZ=1, FTZ=1, RNE rounding.
            // MXCSR is neither consulted nor updated.
            // No exceptions raised or denoted.
            s1e0.int64 = convert_fp8_to_fixpoint64(tsrc1.row[m].float8[4*k+0], type1)
            s2e0.int64 = convert_fp8_to_fixpoint64(tsrc2.row[k].float8[4*n+0], type2)
            s1e1.int64 = convert_fp8_to_fixpoint64(tsrc1.row[m].float8[4*k+1], type1)
            s2e1.int64 = convert_fp8_to_fixpoint64(tsrc2.row[k].float8[4*n+1], type2)

            s1e2.int64 = convert_fp8_to_fixpoint64(tsrc1.row[m].float8[4*k+2], type1)
            s2e2.int64 = convert_fp8_to_fixpoint64(tsrc2.row[k].float8[4*n+2], type2)

            s1e3.int64 = convert_fp8_to_fixpoint64(tsrc1.row[m].float8[4*k+3], type1)
            s2e3.int64 = convert_fp8_to_fixpoint64(tsrc2.row[k].float8[4*n+3], type2)

            temp.int128 = s1e0.int64 * s2e0.int64 + s1e1.int64 * s2e1.int64 + s1e2.int64 * s2e2.int64 + s1e3.int64 * s2e3.int64
            temp1.int128[n] += temp.int128

            // INF Computation treatment:
            // -------------------------------
            // mult(INF, normal) = INF (dest.sign == negative if INF.sign != normal.sign)
            // add(INF, normal) = INF (dest.sign == INF.sign)
            // mult(INF, ZERO) = QNaN_indefinite (i.e. invalid operation)
            // add(+INF, -INF) = QNaN_indefinite (i.e. invalid operation)

            // NaN treatment:
            // ---------------------------
            // if any element:
            // * tsrc1.row[m].float8[4k, 4k+1, 4k+2, 4k+3],
            // * tsrc2.row[k].float8[4n, 4n+1, 4n+2, 4n+3], or
            // * tsrcdest.row[m].float32[n]
            // has any NaN value, then the corresponding result
            // tsrcdest.row[m].float32[n] is 0xFFC0.000 (QNaN Indefinite)
```

```
    for n in 0 ... elements_dest-1:
        // FTZ=1, RNE rounding.
        // MXCSR is neither consulted nor updated.
        // No exceptions raised or denoted.
        tmpf32 = convert_fixpoint128_to_fp32(temp1.int128[n], type1, type2)
        tmp.row[m].fp32[n] = srcdest.row[m].fp32[n] + tmpf32

write_row_and_zero(tsrcdest, m, tmp, tsrcdest.colsb)
zero_upper_rows(tsrcdest, tsrcdest.rows)
zero_tileconfig_start()
```

Flags Affected

None.

Exceptions

AMX-E4; see Section 3.6, “Exception Classes” for details.

TILELOADDRS[,T1]—Load Tile Format Optimized for Read Only Shared Data

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.F2.0F38.W0 4A !(11):rrr:100 TILELOADDRS tmm1, sibmem	A	V/N.E.	AMX-MOVRS	Load data into tmm1 as specified by information in sibmem with read-shared indication.
VEX.128.66.0F38.W0 4A !(11):rrr:100 TILELOADDRST1 tmm1, sibmem	A	V/N.E.	AMX-MOVRS	Load data into tmm1 as specified by information in sibmem with read-shared indication.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	N/A	N/A

Description

This instruction moves data from the source operand (second operand), a memory operand, to the destination operand (first operand), a register. This instruction may be used to load a TMM register from memory, The amount of data moved is specified by the tile register operand's tile configuration. Additionally, this instruction indicates the source memory location is likely to become read-shared by multiple processors, i.e., read in the future by at least one other processor before it is written, assuming it is ever written in the future.

Implementations may optimize the behavior of the caches, especially shared caches, for this data for future reads by multiple processors. A future write to this data before it becomes read-shared will behave as usual, but its performance may be less optimal than if the current read were done via a load without a read-shared hint.

The TILECFG.start_row in the XTILECFG data should be initialized to '0' in order to load or store the entire tile and is set to zero on successful completion of the instruction. This is a restartable instruction and the TILECFG.start_row will be non-zero when restartable events occur during the instruction execution.

Any attempt to execute these instructions inside an Intel TSX transaction will result in a transaction abort.

Operation

TILELOADDRS[T1] tdest, tsib

```
start := tilecfg.start_row
zero_upper_rows(tdest,start)
membegin := tsib.base + displacement
// if no index register in the SIB encoding, the value zero is used.
stride := tsib.index << tsib.scale
nbytes := tdest.colsb
while start < tdest.rows:
    memptr := membegin + start * stride
    write_row_and_zero(tdest, start, read_memory(memptr, nbytes), nbytes)
    start := start + 1
zero_tilecfg_start()
// In the case of a memory fault in the middle of an instruction, the tilecfg.start_row := start
```

Flags Affected

None.

Exceptions

VEX-encoded instructions, AMX-E3. See Section 3.6, "Exception Classes" for details.

EVEX-encoded instructions, AMX-E3-EVEX. See Section 3.6, "Exception Classes" for details.

TILEMOVROW—Tile Move Row

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
EVEX.512.66.0F3A.W0 07 11:rrr:bbb /ib TILEMOVROW zmm1, tmm2, imm8	A	V/N.E.	AMX-AVX512	Move a row specified by imm8 from tmm2 to zmm1.
EVEX.512.66.0F38.W0 4A 11:rrr:bbb TILEMOVROW zmm1, tmm2, r32	B	V/N.E.	AMX-AVX512	Move a row specified by r32 from tmm2 to zmm1.

Instruction Operand Encoding

Op/En	Tuple	Operand 1	Operand 2	Operand 3	Operand 4
A	N/A	ModRM:reg (w)	ModRM:r/m (r)	imm8 (r)	N/A
B	N/A	ModRM:reg (w)	ModRM:r/m (r)	EVEX.vvvv (r)	N/A

Description

These instructions move one row of a tile register to a zmm register. The row of the tile is selected by an imm8, or a 32-bit general-purpose register. If the row indicated is outside the configured range, the destination register is zeroed.

Any attempt to execute the TILEMOVROW instruction inside an Intel TSX transaction will result in a transaction abort.

Operation

TILEMOVROW zdest, tsrc, imm8

```
VL = (512)
VL_bytes := VL>>3 // bits to bytes
row_index := imm8&0xf

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... VL_bytes-1:
    if i >= tsrc.colsb:
        zdest.byte[i] := 0
    else:
        zdest.byte[i] := tsrc.row[row_index].byte[i]

EXIT_AND_SKIP:
zero_tileconfig_start()
```

```
TILEMOVROW zdest, tsrc, r32
VL = (512)
VL_bytes := VL>>3 // bits to bytes
row_index := r32&0xf

if (row_index >= tsrc.rows):
    zdest[VL-1:0] := 0
    GOTO EXIT_AND_SKIP;

for i in 0 ... VL_bytes-1:
    if i >= tsrc.colsb:
        zdest.byte[i] := 0
    else:
        zdest.byte[i] := tsrc.row[row_index].byte[i]

EXIT_AND_SKIP:
zero_tileconfig_start()
```

Flags Affected

None.

Exceptions

Instruction	Exception Type
TILEMOVROW zmm1, tmm2, r32	AMX-E7-EVEX. See Section 3.6, “Exception Classes,” for details.
TILEMOVROW zmm1, tmm2, imm8	AMX-E8-EVEX. See Section 3.6, “Exception Classes,” for details.

CHAPTER 4
INTEL® RESOURCE DIRECTOR TECHNOLOGY FEATURE UPDATES

Intel® Resource Director Technology (Intel® RDT) provides several monitoring and control capabilities for shared resources in multiprocessor systems. This chapter covers updates to the Cache Bandwidth Allocation feature of Intel RDT.

Previous versions of this document contained additional information on Intel RDT. This information, along with more details on RDT-specific terminology and usage guidelines, can now be found in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, as well as in a new document titled "Intel® Resource Director Technology Architecture Specification," available here: https://cdrdv2.intel.com/v1/dl/getContent/789566.

4.1 CACHE BANDWIDTH ALLOCATION (CBA)

4.1.1 Introduction to Cache Bandwidth Allocation

The Cache Bandwidth Allocation (CBA) feature provides control over bandwidth available between Level 1 (L1) caches, Level 2 (L2) Caches, and Level 3 (L3) Caches (as applicable) for each of the logical processors. Since reducing upstream bandwidth can also reduce bandwidth to external memory, this also provides an indirect control of memory bandwidth. The CBA feature, along with the MBA, provides a mechanism to control the bandwidth of different applications.

A given CLOS used for L3 CAT, for instance, means the same thing as a CLOS used for CBA. Infrastructure such as the MSR used to associate a logical processor with a CLOS (the IA32_PQR_ASSOC_MSR) and some elements of the CPUID enumeration (such as CPUID leaf 10H (Cache Allocation Technology Enumeration Leaf)) are shared. For more information, refer to the "Intel® Resource Director Technology Architecture Specification."

The following sections describe the CPUID enumeration and configuration interfaces (Model-Specific Registers) applicable to the Cache Bandwidth Allocation feature.

4.1.2 Cache Bandwidth Allocation Enumeration

As with certain other Intel RDT features, enumeration of the presence and details of the CBA feature is provided via a sub-leaf of the CPUID instruction.

Key components of the enumeration include support for the CBA feature on the processor, and if CBA is supported, the following details:

- Number of supported Classes of Service for the processor.
- Scope of the CBA feature MSRs.
- The maximum CBA throttle Level supported.
- An indication of whether the throttle values that can be programmed are linearly spaced or not.

The presence of any of the Intel RDT features that enable control over shared platform resources is enumerated by executing CPUID instruction with EAX = 07H and ECX = 0H as input. If CPUID.(EAX=07H, ECX=0H):EBX.PQE[bit 15] reports 1, the processor supports software control over shared processor resources. Software may then use CPUID leaf 10H to enumerate additional details on the specific controls provided.

Using CPUID leaf 10H, software may determine whether CBA is supported on the platform. Specifically, as shown in Figure 17-31, bit 5 of the EBX register indicates whether CBA is supported on the processor, and the bit position (5) constitutes a Resource ID (ResID), which allows the enumeration of CBA details. For instance, if bit 5 is supported, this implies the presence of CPUID.10H.[ResID=5] as shown in CPUID.(EAX=10H, ECX=5H), CBA Feature Details Identification, which provides the following details (this information can also be found in Table 1-3, "Information Returned by CPUID Instruction"):

- CPUID.(EAX=10H, ECX=ResID=5):EAX
  - EAX[7:0] reports the maximum CBA throttling value supported.
  - EAX[11:8] reports the scope of CBA IA32_QoS_Core_BW_Thrtl_n MSRs. If EAX[11:8]=1, this indicates the logical processor scope of the MSRs.
  - EAX[31:12] is reserved.
- CPUID.(EAX=10H, ECX=ResID=5):EBX
  - EBX[31:0] is reserved.
- CPUID.(EAX=10H, ECX=ResID=5):ECX
  - ECX[3] reports whether the response of the bandwidth control is approximately linear. If ECX[3] is 1, the response of the bandwidth control is approximately linear. If ECX[3] is 0, the response of the bandwidth control is non-linear.
  - ECX[2:0] and ECX[31:4] are reserved.
- CPUID.(EAX=10H, ECX=ResID=5):EDX
  - EDX[15:0] reports the number of Classes of Service supported for the feature. Add one to the return value to get the result. For instance, a reported value of 15 implies a maximum of 16 supported CBA CLOS.
  - EDX[31:16] is reserved.

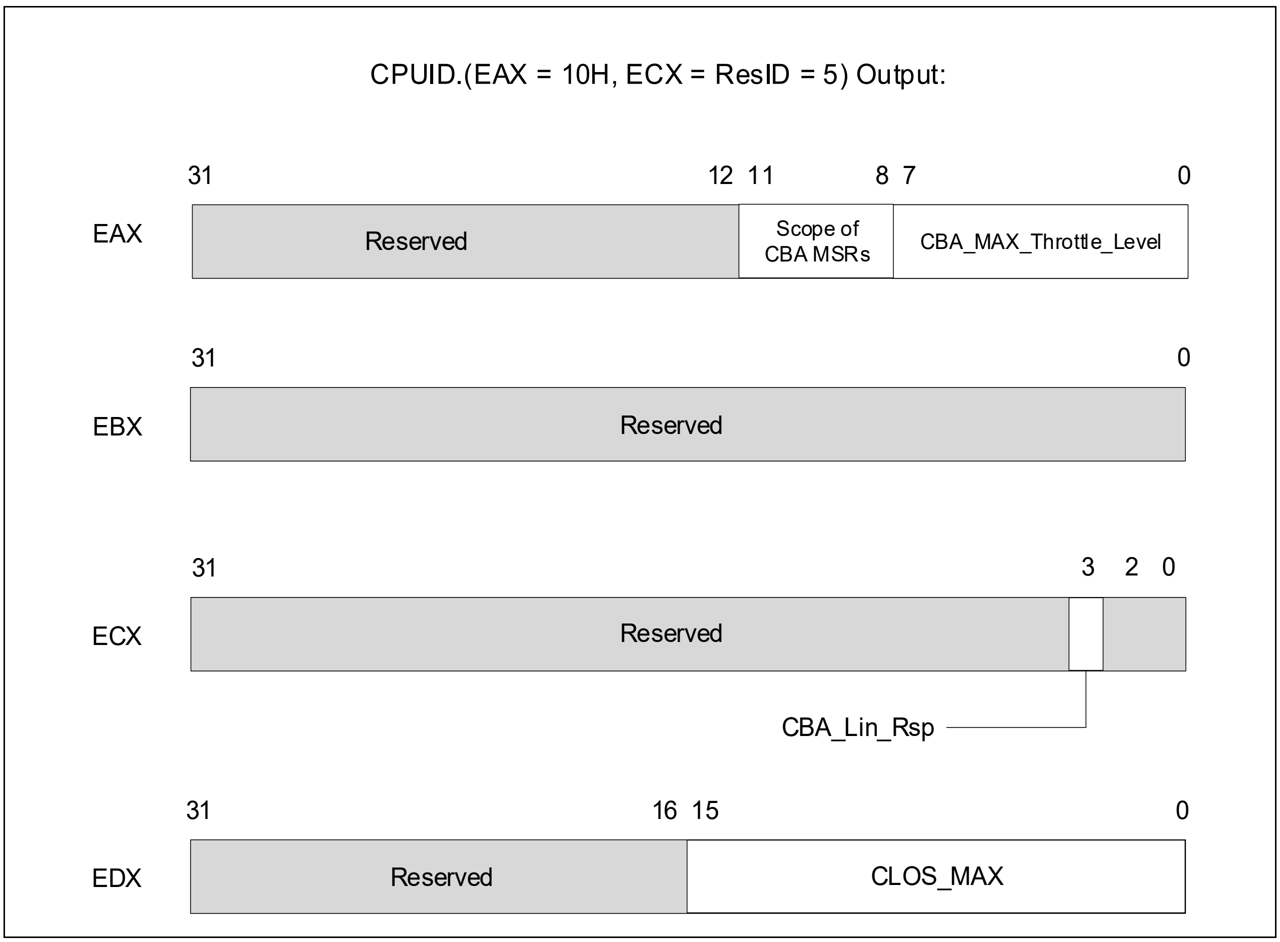


Figure 4-1. CPUID.(EAS=10H, ECX=5H), CBA Feature Details Identification

4.1.3 Cache Bandwidth Allocation Configuration

The configuration of CBA consists of two processes once enumeration is complete:

- The association of logical processors to Classes of Service (CLOS) is accomplished commonly across Intel RDT features through the IA32_PQR_ASSOC MSR. Software may update the CLOS field of the PQR MSR dynamically, including at context swap time, to maintain the proper association of logical processors to Classes of Service on the hardware.
- A new set of architectural MSRs is added to enable software to communicate the memory bandwidth QoS requirements of the application running on the logical processor. The scope of these MSRs, IA32_QoS_Core_BW_Thrtl_n, is per logical processor. Each MSR encodes packed 8-bit fields indexed by Class of Service, which specify the Throttle Level for each CLOS.

The CLOS field value of the IA32_PQR_ASSOC MSR is used to index into the MSRs and select the software-specified Throttle Level. Each logical processor uses this level to control the bandwidth across the cache hierarchy. The hardware ensures coordination with the MBA feature (where present). The reset value of the CLOS[i].Level=0 indicates unthrottled bandwidth. This field may be programmed from 0 to CBA_MAX_Throttle_Level (see Figure 4-1). Any values outside this range will generate a #GP(0). A higher value of CLOS[i].Level implies a higher level of bandwidth throttling, and a lower number indicates lesser throttling. The number of supported CLOS for a given logical processor is enumerated in CPUID.(EAX=10H, ECX=5H):EDX. In an example where a logical processor supports 16 CLOS, two 64-bit MSRs with packed Throttling Levels (TLs) are defined, IA32_QoS_Core_BW_Thrtl_0 (defining packed TLs for CLOS[7:0]) and IA32_QoS_Core_BW_Thrtl_1 (defining TLs for CLOS[15:8]). For example, within the MSR IA32_QoS_Core_BW_Thrtl_0, bits [7:0] define the TL field for CLOS 0 (see Figure 4-2).

Advanced versions of the MBA feature may manage the external memory bandwidth associated with the CLOS by dynamically increasing or decreasing the bandwidth, under software guidance, to maintain throttling priorities while maximizing system performance as described in the Intel Software Developer's Manual and the Intel RDT Architecture Specification. The CBA feature, along with the MBA, provides a mechanism to control the bandwidth of different applications. Software should understand that the effective throttling of an application may be whichever of the CBA or MBA specifies more throttling. Software may use CBA, MBA, or a combination to achieve bandwidth and performance management goals if supported on a processor.

Table 4-1. Cache Bandwidth Allocation (CBA) MSRs

Delay Value MSR	Address
IA32_QoS_Core_BW_Thrtl_0	E00H
IA32_QoS_Core_BW_Thrtl_1	E01H
IA32_QoS_Core_BW_Thrtl_2	E02H
...	...
IA32_QoS_Core_BW_Thrtl _'(((COS_MAX+1)/8) - 1)'	E00H + ((COS_MAX from CPUID.10H.5 + 1)/8 - 1)

Base MSR Address = E00H (bits 63 to 0)

IA32_QoS_Core_BW_Thrtl_0 MSR							
[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
CLOS[7].Level	CLOS[6].Level	CLOS[5].Level	CLOS[4].Level	CLOS[3].Level	CLOS[2].Level	CLOS[1].Level	CLOS[0].Level

Base MSR Address = E01H (bits 63 to 0)

IA32_QoS_Core_BW_Thrtl_1 MSR							
[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
CLOS[15].Level	CLOS[14].Level	CLOS[13].Level	CLOS[12].Level	CLOS[11].Level	CLOS[10].Level	CLOS[9].Level	CLOS[8].Level

Figure 4-2. IA32_QoS_Core_BW_Thrtl_n MSR Definition

Note that the throttling values provided to the software are calibrated through specific traffic patterns; however, as workload characteristics may vary, the response precision and linearity of the bandwidth threshold values will vary across products and should be treated as approximate values only.

4.1.4 Cache Bandwidth Allocation Usage Considerations

Cache Bandwidth Allocation has various software usage considerations and improves efficiency over product generations. See the "Intel® Resource Director Technology Architecture Specification" for additional details.

4.2 MEMORY REGIONS

A Memory Region is defined as comprising one or more physical-addressed memory ranges. Certain processors support a system-level enumeration of Memory Regions. Multiple Memory Regions may be defined by the platform to independently describe physical addresses backed by a particular type of memory, which may exhibit varying capacity, latency, bandwidth, and locality characteristics. Examples include DRAM or CXL attached memories, whether attached locally or to a different processor over a coherent interconnect link. The Memory Regions populated on a particular processor are described by the system BIOS in the ACPI Memory Range and Region Mapping (MRRM) table. See the "Intel® Resource Director Technology Architecture Specification" for additional details.

For a modern platform, it becomes important for the processor to directly measure and allocate memory bandwidth across these multiple memory regions simultaneously. Such processor capabilities, when enabled, allow software to gather usage telemetry, adjust Memory Bandwidth Allocation (MBA) policies, and build control loops to ensure performance goals are met. As described below, Intel provides these capabilities as Region Aware Memory Bandwidth Monitoring and Allocation

4.3 REGION AWARE MEMORY BANDWIDTH MONITORING (MBM)

The Region Aware Memory Bandwidth Monitoring (MBM) feature provides a set of counters simultaneously indexed by Resource Monitoring ID (RMID) and Region to measure the memory bandwidth utilization of an RDT RMID to a Memory Region in the system. The RDT RMID is typically mapped to software threads, applications, containers or virtual machines. Typical hardware feature support for Region Aware MBM includes the ability to independently track many RMIDs simultaneously accessing several Memory Regions. Software may consult the Enhanced Resource Director Technology (ERDT) ACPI table for enumeration of specific capabilities of this feature on a given processor generation. The ERDT table provides information regarding capabilities, SoC topology mappings to the scope of specific RDT features, Region Aware MBM MMIO interface register locations, and architectural parameters such as the number of RMIDs supported. See the "Intel® Resource Director Technology Architecture Specification" for additional details.

4.4 REGION AWARE MEMORY BANDWIDTH ALLOCATION (MBA)

Region Aware Memory Bandwidth Allocation (MBA) for CPU Agents extends the existing third-generation (per-Agent throttling) MBA capabilities to include Region Aware bandwidth controls per RDT Class of Service (CLOS). The Memory Region definitions used for Region Aware MBM and MBA are shared across the features, as specified in the ACPI MRRM table, allowing simultaneous and consistent monitoring and allocation of memory bandwidth. Independent throttling of per-CLOS bandwidth to multiple regions is supported, allowing software to dynamically rebalance bandwidth throttling limits across different Memory Regions, which may have varying bandwidth, latency and capacity characteristics. Example uses include rebalancing bandwidth between VMs of different priority across a shared coherent interprocessor interconnect under the direction of a software control loop or rebalancing bandwidth for threads of varying priorities across DRAM or CXL-backed memories. See Figure 4-3 for a high-level overview of Region Aware MBA. See the "Intel® Resource Director Technology Architecture Specification" for additional details.

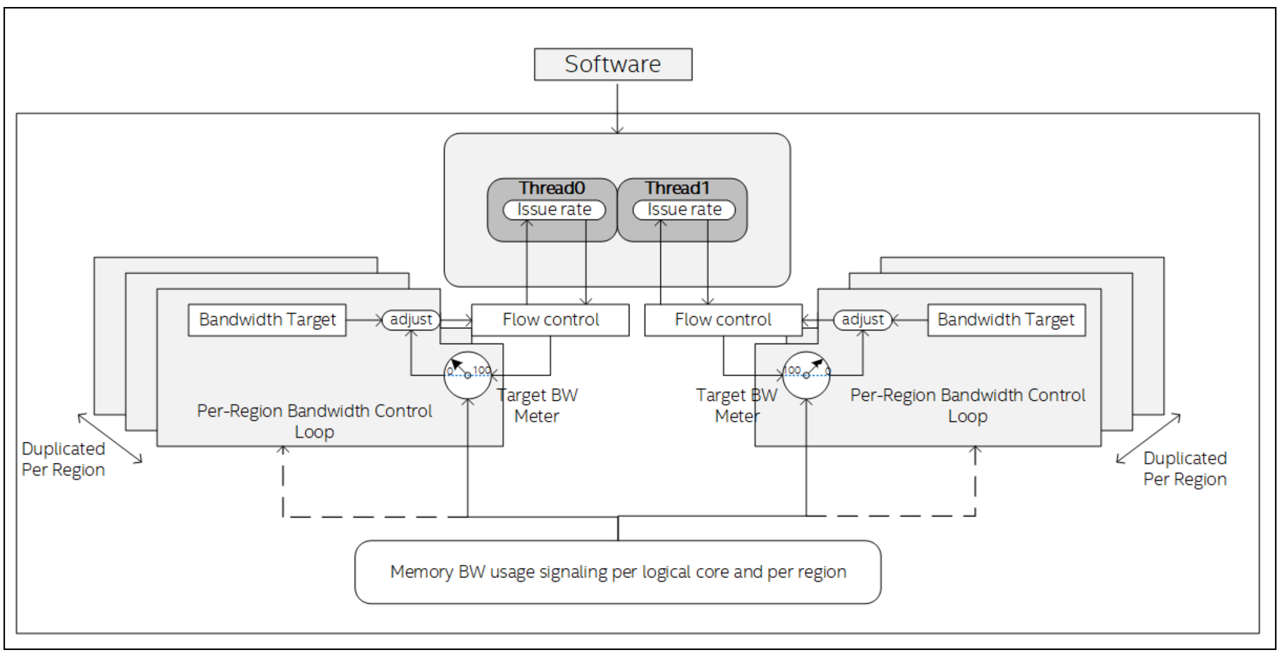


Figure 4-3. Region Aware Memory Bandwidth Allocation (MBA)

4.5 USE OF RDT TAGS BY OTHER FEATURES

Other features may utilize RDT tags, such as Resource Monitoring IDs, to track and report other telemetry events in the processor. Examples include per-RMID telemetry available both in-band and out-of-band as specified in the Intel® Platform Monitoring Technology (Intel® PMT) specification and associated platform-specific telemetry events lists.

CHAPTER 5
REMOTE ATOMIC OPERATIONS IN INTEL ARCHITECTURE

5.1 INTRODUCTION

Remote Atomic Operations (RAO) are a set of instructions to improve synchronization performance. RAO is especially useful in multiprocessor applications that have a set of characteristics commonly found together:

- A need to update, i.e., read and modify, one or more variables atomically, e.g., because multiple processors may attempt to update the same variable simultaneously.
- Updates are not expected to be interleaved with other reads or writes of the variables.
- The order in which the updates happen is unimportant.

One example of this scenario is a multiprocessor histogram computation, where multiple processors cooperate to compute a shared histogram, which is then used in the next phase of computation. This is described in more detail in Section 5.8.1.

RAO instructions aim to provide high performance in this scenario by:

- Atomically updating memory without returning any information to the processor itself.
- Relaxing the ordering of RAO instructions with respect to other updates or writes to the variables.

RAO instructions are defined such that, unlike conventional atomics (e.g., LOCK ADD), their operations may be performed closer to memory, such as at a shared cache or memory controller. Performing operations closer to memory reduces or even eliminates movement of data between memory and the processor executing the instruction. They also have weaker ordering guarantees than conventional atomics. This facilitates execution closer to memory, and can also lead to reduced stalls in the processor pipeline. These properties mean that using RAO instead of conventional atomics may provide a significant performance boost for the scenario outlined above.

5.2 INSTRUCTIONS

The current set of RAO instructions can be found in Chapter 2, “Instruction Set Reference, A-Z.” These instructions include integer addition and bitwise AND, OR, and XOR. These operations may be performed on 32-bit (doubleword) or 64-bit (quadword) data elements. The destination, which is also one of the inputs, is always a location in memory. The other input is a general-purpose register, ry, in Table 5-1. The instructions do not change any registers or flags.

Table 5-1. RAO Instructions

Instruction	Operation	Function	Data Types
AADD	Atomic addition	mem = mem + ry	Doubleword, quadword
AAND	Atomic bitwise AND	mem = mem AND ry	Doubleword, quadword
AOR	Atomic bitwise OR	mem = mem OR ry	Doubleword, quadword
AXOR	Atomic bitwise XOR	mem = mem XOR ry	Doubleword, quadword

5.3 ALIGNMENT REQUIREMENTS

The memory location updated by an RAO instruction must be naturally aligned. That is, a doubleword update must be four-byte aligned and a quadword update must be eight-byte aligned. This facilitates implementations closer to memory; otherwise, a single update may straddle a cache line boundary.

5.4 MEMORY ORDERING

RAO instructions have weaker memory ordering guarantees than conventional atomic instructions. Thus, other instructions are not ordered with respect to RAO instructions as they are with conventional atomics.

More specifically, the memory operations from RAO instructions follow the Write Combining (WC) memory protocol. From software's point of view, they behave similarly to non-temporal stores. Unlike non-temporal stores, RAO instructions update a memory location, i.e., use the value in that location as an input, rather than overwrite the current contents. Another critical difference is that with RAO, the memory location may be cached upon completion of the instruction.

RAO instructions are not reordered with other memory accesses to the same memory location. That is, reads, writes, and RAO instructions to the same location by the same processor will execute in program order.

However, RAO instructions may be reordered with certain memory accesses to other memory locations. In particular, RAO instructions may be reordered with writes or RAO instructions to other memory locations. This means, for example, that if a processor executes a set of RAO instructions to a set of distinct addresses, those instructions may appear to update memory in any order.

If a stronger ordering is required, software should use a fencing operation such as those implemented by the LFENCE, SFENCE, and MFENCE instructions. However, note that RAO instructions are not ordered with respect to younger LFENCE instructions since they do not load data from memory into the processor.

5.5 MEMORY TYPE

RAO instructions are restricted to operating on Write Back (WB) memory. Other memory types place restrictions on the writing of and/or cacheability of data, which conflicts with RAO instructions' ability to cache data. Use of an RAO instruction to access non-WB memory results in a general-protection exception (#GP).

5.6 WHEN MULTIPLE RAO MEMORY OPERATIONS ARE COMBINED

Processor implementations may combine multiple RAO memory operations before execution to increase their throughput. Such implementations that combine RAO instructions take advantage of spatial locality, i.e., that a cache line contains multiple data elements, and that separate instructions may update distinct elements in a given cache line.

Note, implementations may have restrictions on which RAO operations can be combined. For example, they may only be able to combine operations doing the same type of update (e.g., addition) and/or the same data element size. Lastly, RAO operations to the same cache line that are not combined must be serialized by the processor implementation; this could hurt performance. Therefore, as a general rule for good performance across processor implementations, if multiple RAO operations to the same cache line are of different types, data sizes, or to the same memory location, software should increase the execution distance between such RAO operations when possible.

5.7 PERFORMANCE EXPECTATIONS

RAO instructions are expected to provide higher performance than conventional atomics under certain conditions. The actual performance depends on both the implementation and the data access pattern for the memory location (at the cache line granularity) updated with RAO instructions.

5.7.1 Interaction Between RAO and Other Accesses

As discussed in Section 5.4, weak ordering allows RAO instructions to be reordered with respect to other memory operations. This is a key difference from conventional atomics, which follow strong memory ordering, and can allow a processor to execute RAO instructions with higher throughput. However, only certain reordering is allowed. If a fence is used to enforce stronger ordering, or if a processor interleaves RAO updates with reads of the same memory location, for example, this may result in serialized accesses, and hurt performance. If software performs an RAO update to a memory location, and soon after reads that memory location, then the read needs to wait for the update to complete. If the RAO is done close to memory, then the cache closest to the processor may not hold a copy of the cache line after the RAO instruction executes, and the read may need to access a cache farther away from the processor, or even go all the way to memory.

Mixing of RAO updates to a given memory location from one or more processors with non-RAO accesses to the same memory location can also reduce the benefits of RAO. Implementations that perform RAO updates close to memory can reduce data movement between a series of RAO updates to the same location. However, a non-RAO access may cause a processor to cache the data close to itself; a subsequent RAO instruction from another processor may require the line to be moved to a lower level of the cache hierarchy. Therefore, interleaving RAO and non-RAO accesses to a given memory location can reduce or eliminate the data movement and/or performance benefits of RAO.

5.7.2 Updates of Contended Data

Contended data is defined as data for which the memory system has memory accesses from multiple processors in-flight simultaneously. That is, for contended data, the memory system is at some point in time handling at least two accesses from different processors. Contended read-only data does not present a fundamental performance problem, but if at least one of the contending processors attempts to write the data, e.g., perform an update on it, the writer needs exclusive access to the data. Gaining exclusive access can be costly, in terms of latency and traffic; in a system with caches, hardware must invalidate all other copies of the data to provide a processor exclusive access.

For software performing a set of contended updates to a memory location with conventional atomic instructions, data may “ping-pong” between processors. As each processor executes its update, it will obtain exclusive access to the data, perform its update, and then have to send its new version of the data to the next processor wanting to update it. The time to pass data from one processor to another, and the time that a processor takes to perform its atomic update, limits the throughput in this scenario.

In contrast, if software uses RAO for such contended updates, and if the implementation performs the updates in a central location such as a shared cache or at the memory controller, then this bottleneck is alleviated. In such a scenario, each update will not have to fetch the current contents of the memory location or invalidate any other copies of the data because the only valid copy is already at the hardware performing the update. The only fundamental limit to the throughput in this case is the time taken for each update. Therefore, we may expect that for updates to contended lines, throughput is much higher with RAO. Further, reducing data movement means reducing traffic between processors and memory. This may improve the performance of other memory accesses.

5.7.3 Updates of Uncontended Data

In contrast to contended data, uncontended data is data that is accessed by only a single processor or by multiple processors, but far enough apart in time that at most a single memory access is executed at a time.

For uncontended data accessed by multiple processors, most of the above discussion about contended data still applies. However, the frequency of updates is by definition lower for uncontended data. Therefore, the performance benefits of RAO are expected to be lower in this situation.

For data accessed by only a single processor, data movement between processors is not an issue, and conventional atomics can take advantage of the processor's caches. Performance may still be impacted by the strong ordering of conventional atomics; memory accesses to other memory locations may not be reordered with these instructions. If software uses RAO instructions instead, the weaker ordering may provide some performance benefits. However, if an implementation performs RAO updates closer to memory, it may not take advantage of all of the processor's caches, and may even require removing the data from some of those caches. This could lead to an increase in data movement, and potentially lower performance. Of course, if software is aware that only a single processor will access the data, then it does not need to use atomic updates, but it may not always be so aware.

5.8 EXAMPLES

5.8.1 Histogram

Histogram is a common computational pattern, including in multiprocessor programming, but achieving an efficient parallel implementation can be tricky. In a conventional histogram computation, software sweeps over a set of input values; it maps each input value to a histogram bin, and increments that bin.

Common multiprocessor histogram implementations partition the inputs across the processors, so each processor works on a subset of the inputs. Straightforward implementations have each processor directly update the shared histogram. To ensure correctness, since multiple processors may attempt updates to the same histogram bin simultaneously, the updates must use atomics. As described above, using conventional atomics can be expensive, especially when we have highly contended cache lines in the histogram. That may occur for small histograms or for histograms where many inputs map to a small number of histogram bins.

A common alternative approach uses a technique called privatization, where each processor gets its own "local" histogram; as each processor works on its subset of the inputs, it updates its local histogram. As a final "extra" step, software must accumulate the local histograms into the globally shared histogram, a step called a reduction. This reduction step is where processors synchronize and communicate; using it allows the computation of local histograms to be embarrassingly parallel and require no atomics or inter-processor communication, and can often lead to good performance. However, privatization has downsides:

- The reduction step can take a lot of time if the histogram has many bins.
- The time for a reduction is relatively constant regardless of the number of processors. As the number of processors grows, therefore, the fraction of time spent on the reduction tends to grow.
- The local histograms require extra memory, and that memory footprint grows with the number of processors.
- The reduction is an "extra" step that complicates the software.

With RAO, software can use the simpler multiprocessor algorithm and achieve reliably good performance. The following pseudo-code lists a RAO-based histogram implementation.

```
int *histogram; // "histogram" is a global histogram array

// in each processor:
double *data; // "data" is a per-processor array, holding a subset of all inputs
data = get_data(); // populate "data" values

for (size_t i = 0; i < data_size; ++i) {
    int bin = map(data[i]); // map data[i] to a histogram bin
    _aadd(&histogram[bin], 1); // RAO AADD instruction
}
```

The above code can provide good performance under various scenarios, i.e., sizes of histograms and biases in which histogram bins are updated. RAO avoids data "ping-ponging" between processors, even under high contention. Further, the weak ordering of RAO allows a series of AADD instructions to overlap with each other in the pipeline, and thus provide for instruction level parallelism.

In addition to the performance benefits, the RAO code is simple and is thus easier to maintain.

While we specifically show and discuss histogram above, this computation pattern is very common, e.g., software packet processing workloads exhibit this in how they track statistics of the packets. Other algorithms exhibiting this pattern should similarly see benefits from RAO.

5.8.2 Event Handler

A user-mode event handler, running either in a dedicated thread or preemptively in a specific processor, notifies a set of receivers (e.g., all processors or threads in a waiting list) of the occurrence of an event by atomically setting flags in the receivers' specific data fields. The example below shows how this may be done with RAO instructions.

```
// One processor sets event bits to notify other processors:
01: void handle_event(event_t *e) {
02:    uint32_t event_bits = process_event(e);
03:    for (int i = 0; i < num_of_receivers; ++i) {
04:       core_t *core = receivers[i];
05:       _aor(&core->flags, event_bits); // RAO AOR instruction
06:       if (some_condition) {
07:          _aor(&core->extra_flags, event_bits); // combining of RAO could occur
08:       }             // if "extra_flags" and "flags" are in the same cache line
09:    }
10:    _mm_sfence();  // ensure event_bits are visible before leaving the handler
11: }

// In other processors:
12: if (my_core->flags & SOME_EVENT) {
13:    ...... // react to the occurrence of SOME_EVENT
14:    clear_bits(&my_core->flags, SOME_EVENT);
15: }
```

With conventional atomics (e.g., LOCK OR), a significant portion of execution time of handle_event would be spent accessing core->flags (line 5) and core->extra_flags (line 7). It is likely that when handle_event begins, the two fields are in another processor's cache, e.g., if that processor updated some bits in the fields. Therefore, the data would need to migrate to the cache of the processor executing handle_event.

In contrast, for the above code example, for RAO implementations that perform updates close to memory, the RAO AOR instruction should reduce data movement of core->flags and core->extra_flags and thus result in a lower execution latency. Further, when other processors later access these fields (lines 12-15), they will also benefit from a lower latency due to reduced data movement, since they may get the data from a more central location.

Also note that since the order of notifications does not matter in this case, the function further takes advantage of RAO's weak ordering, allowing multiple RAO AOR instructions to be executed concurrently. It does, however, include a memory fence at the end (line 10), to ensure that all updates are visible to all processors before leaving the handler.

CHAPTER 6
MOVRS INSTRUCTIONS

The mnemonic MOVRS identifies a set of load instructions that carry a "read-shared" hint. Functionally, these instructions are equivalent to existing load instructions (and can be used as "drop-in" replacements) for which the read-shared hint is appropriate. From a performance point of view, the hint indicates that the data being accessed is expected to be read concurrently by multiple cores.

Intel processors are typically optimized assuming that, on a first read, data is private and is reasonably likely to be written. This assumption influences the hardware's determination of the cache locations into which data are placed on an access as well as the coherence state of the data after the access.

These determinations are especially relevant for hardware with non-inclusive last-level caches. Accessing read-shared data, i.e., data concurrently read by multiple cores, with conventional loads can cause performance issues. In particular, for both a multi-readers scenario where data starts in DRAM and a single-producer multi-consumer scenario where data starts dirty in one core's private cache, snoops are required to distribute the data to the readers.

Loads with a read-shared hint are expected to trigger uncore[1] behavior different from that used for conventional loads. The read-shared hint tells caches that the data is expected to be read by multiple (unspecified) cores within the socket. Thus, hardware, to the extent possible, installs the cache line into the appropriate cache(s) and in the appropriate coherence state to minimize snoops triggered by future reads.

On a cache miss, the read-shared hint may influence which cache(s) hardware installs data into, and the uncore transactions used to fetch data to the core. On a cache hit, the hint is not expected to trigger different behavior. Also, at least for currently planned implementations, the hint is not stored in the caches; hardware will act (or not) on the hint immediately. The exception to this is hardware prefetchers, which will train on the hint, and attempt to issue requests matching the presence/absence of the hint.

Mixing conventional loads and read-shared loads to the same cache line and/or access stream will behave as expected functionally. However, the cache behavior will depend on the order of the requests and the behavior of the hardware prefetchers. It is easier to predict and understand the hardware behavior if software consistently uses (or not) read-shared loads for a given access pattern.

The exception to this is phased program behavior. Software may have phases, where a given data structure is read-shared in one phase, and not in the next phase. It is expected that the different phases will use the appropriate type of load instruction, i.e., read-shared loads for the phase with read sharing and conventional loads for other phases.

Since the read-shared hint simply influences which caches hold a copy of a line and the coherence state, it does not limit how the data is accessed by software in the future. In particular, a future write to a cache line that has been read with a read-shared load instruction, by the same core that executed the load instruction or another core, will behave as expected architecturally. The performance of such a write may be worse than if the data was read with a conventional load instruction, if the read-shared load leaves caches in a state such that additional cache and/or uncore transactions are needed for the write.

The MOVRS family includes scalar, vector, and tile variants. It also includes a software prefetch instruction, PREFETCHRST2, that carries the same read-shared hint. This instruction is for applications that want to apply software prefetching to read-shared data structures.

1. The term "uncore" roughly equates to logic outside the CPU cores but residing on the same die.

CHAPTER 7
INTEL® SOFTWARE GUARD EXTENSIONS FEATURE UPDATES

Intel® Software Guard Extensions (Intel® SGX) is a set of instructions and mechanisms for memory accesses in Intel® Architecture processors. This chapter covers updates of data structures and instructions for SGX support of 256-bit cryptography and enabling local attestation between SGX and TDX.

Further details of the Intel® SGX can be found in the Intel® 64 and IA-32 Architectures Software Developers Manual, Volume 3B. In general, various functionality is encoded as leaf functions within the ENCLS (supervisor), ENCLU (user), and the ENCLV (virtualization operation) instruction mnemonics. Different leaf functions are encoded by specifying an input value in the EAX register of the respective instruction mnemonic.

Further details of Intel® Trust Domain Extensions (Intel® TDX) can be found in the Intel® Trust Domain CPU Architectural Extensions specification.

7.1 ENCLAVE INSTRUCTIONS AND INTEL® SGX

The enclave instructions available with Intel SGX are organized as leaf functions under two instruction mnemonics: ENCLS (supervisor) and ENCLU (user). Each leaf function uses EAX to specify the leaf function index, and may require additional implicit input registers as parameters. The use of EAX is implied implicitly by the ENCLS and ENCLU instructions; ModR/M byte encoding is not used with ENCLS and ENCLU. The use of additional registers does not use ModR/M encoding and is implied implicitly by the respective leaf function index.

Each leaf function index is also associated with a unique, leaf-specific mnemonic. A long-form expression of Intel SGX instruction takes the form of ENCLx[LEAF_MNEMONIC], where 'x' is either 'S' or 'U'. The long-form expression provides clear association of the privilege-level requirement of a given "leaf mnemonic". For simplicity, the unique "Leaf_Mnemonic" name is used (omitting the ENCLx for convenience) throughout in this document.

Table 7-1. Supervisor and User Mode Enclave Instruction Leaf Function in Long-Form of 256-bit SGX

Supervisor Instruction	Description	User Instruction	Description
		ENCLU[EGETKEY256]	Create a 256-bit cryptographic key.
		ENCLU[EREPORT2]	Create a cryptographic report that contains SHA384 measurements.

7.2 INTEL® SGX RESOURCE ENUMERATION LEAVES

If CPUID.(EAX=07H, ECX=0H):EBX.SGX[bit 02]=1, the processor also supports querying CPUID with EAX=12H on Intel SGX resource capability and configuration. If CPUID.(EAX=12H, ECX=0H):EAX.256BITSGX[bit 12] = 1, Intel® SGX supports leaf functions EGETKEY256 and EREPORT2, MSRs IA32_SGXLEPUBKEYHASH[4,5] and MSR IA32_SGXLECONFIG.

If CPUID.(EAX=12H, ECX=00H):EAX.256BITSGX[bit 12] = 1, SGX Launch Control can be configured to require a stronger HMAC-SHA256 Message Authentication Code rather than the default AES128-CMAC. The IA32_SGXLE-CONFIG MSR configures this option.

7.3 INTEL® SGX MSR UPDATES

Table 7-2. IA-32 Architectural MSR Updates for Intel® SGX

Register Address: Hex, Decimal		Architectural MSR Name (Former MSR Name)
Bit Fields	**MSR/Bit Description**	**Comment**
Register Address: 8AH, 138	IA32_SGXLECONFIG	
IA32 SGXLECONFIG (R/W) Additional SGX Launch Control configuration.		Read permitted If CPUID.(EAX=12H,ECX=0H): EAX[0]=1 AND CPUID.(EAX=07H, ECX=0H):ECX[30]=1 AND CPUID.(EAX=12H,ECX=0H): EAX[12]=1 Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 AND CPUID.(EAX=12H,ECX=0H): EAX[12]=1 AND IA32_FEATURE_CONTROL[17] = 1 AND IA32_FEATURE_CONTROL[0]=1.
0	Launch Control Key Strength 0: 128-bit (AES128-CMAC) 1: 256-bit (HMAC-SHA256)	
63:1	Reserved.	
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.		Read permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 AND CPUID.(EAX=07H,ECX=0H): ECX[30]=1 AND CPUID.(EAX=12H,ECX=0H): EAX[12]=1 Write permitted if CPUID.(EAX=12H,ECX=0H): EAX[0]=1 AND CPUID.(EAX=12H,ECX=0H): EAX[12]=1 AND IA32_FEATURE_CONTROL[17] = 1 AND IA32_FEATURE_CONTROL[0] = 1.
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_SGXLEPUBKEYHASH4 (MSR address 90H, 144) applies here.

7.4 MEMORY ACCESSES

Table 7-3. 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
EEXTEND	SGX1	SECS	EPCPAGE		
EGETKEY	SGX1	KEYREQUEST	KEY		SECS
EGETKEY256	256BITSGX	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
EREPORT2	256BITSGX	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 the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3.					

7.5 INTEL® SGX DATA STRUCTURES

7.5.1 SGX Enclave Control Structure (SECS)

The SECS data structure requires 4K-bytes alignment.

MRENCLAVE384 and MRSIGNER384 are added to the Reserved field, which consists of:

- 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.

- VIRTCHILDCNT: An 8 byte Count of virtual children that have been paged out by a VMM. Its location is implementation specific.
- ENCLAVECONTEXT: An 8 byte Enclave context pointer. Its 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.
- MRENCLAVE384: Measurement Register of the enclave build process.
- MRSIGNER384: Measurement Register extended with the public key that verified the enclave. See Section 7.5.3, "Enclave Signature Structure (SIGSTRUCT)" for format.
- The remaining 3130 bytes are reserved area.

The Reserved field is at offset 262 and the entire 3834-byte field must be cleared prior to executing ECREATE.

7.5.2 Attributes

The ATTRIBUTES data structure is comprised of bit-granular fields that are used in the SECS, the REPORT and the KEYREQUEST structures. CPUID.(EAX=12H, ECX=1) enumerates a bitmap of permitted 1-setting of bits in ATTRIBUTES.

Table 7-4. 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 EDBGRD and EDBGWR.
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.(EAX=12H, ECX=1):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.
SHA384	11	This bit enables SHA384 measurements for the enclave.
RESERVED	63:12	Must be zero.
XFRM	127:64	XSAVE Feature Request Mask. See "Interactions with Processor Extended State and Miscellaneous State" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3.

7.5.3 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 a SHA256 or SHA384 digest, as defined in FIPS PUB 180-4. The digests are byte strings of length 32 or 48 bytes. 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 string 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 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

SIGSTRUCT must be page aligned

In column 5 of Table 7-5, 'Y' indicates that this field should be included in the signature generated by the developer.

Table 7-5. 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 yyyymmdd 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
SIGHASHTYPE	44	4	If this field is 0, SIGHASHTYPE is SHA256. If this field is 1, SIGHASHTYPE is SHA384.	Y
RESERVED	48	80	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.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Enclave CET attributes that must be set. When CPUID.(EAX=12H, ECX=1):EAX[6] is 0, this field is reserved and must be 0.	Y
CET_ATTRIBUTES_MASK	909	1	When CPUID.(EAX=12H, ECX=1):EAX[6] is 1, this field provides the Mask of CET attributes to enforce. When CPUID.(EAX=12H, ECX=1):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

Table 7-5. Layout of Enclave Signature Structure (SIGSTRUCT) (Contd.)

Field	OFFSET (Bytes)	Size (Bytes)	Description	Signed
Q1	1040	384	Q1 value for RSA Signature Verification.	N
Q2	1424	384	Q2 value for RSA Signature Verification.	N

NOTES:

1. If CPUID.(EAX=12H, ECX=0):EBX[31:0] = 0, MISCSELECT must be 0.

7.5.4 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 7-6. Layout of EINIT Token (EINITTOKEN)

Field	OFFSET (Bytes)	Size (Bytes)	MACed	Description
VALID	0	4	Y	Bit 0: If this bit is 1, the token is valid; if 0, the token is invalid. Bit 1: LE256. If this bit is 1, 256-bit security is on the token; if 0, 128-bit security is on the token. All other bits reserved.
RESERVED	4	44	Y	Must be zero.
ATTRIBUTES	48	16	Y	ATTRIBUTES of the Enclave.
MRENCLAVE	64	48	Y	MRENCLAVE of the Enclave. SHA256 digests must be aligned to offset 64, and offsets 111:96 must be zero.
RESERVED	112	16	Y	Reserved.
MRSIGNER	128	48	Y	MRSIGNER of the Enclave. SHA256 digests must be aligned to offset 64, and offsets 111:96 must be zero.
RESERVED	176	16	Y	Reserved.
CPUSVNLE	192	16	N	Launch Enclave's CPUSVN.
ISVPRODIDLE	208	2	N	Launch Enclave's ISVPRODID.
ISVSVNLE	210	2	N	Launch Enclave's ISVSVN.
CET_MASKED_ATTRIBUTES_LE	212	1	N	When CPUID.(EAX=12H, ECX=1):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_ATTTRIBUTES_MASK of the LE's KEYREQUEST. When CPUID.(EAX=12H, ECX=1):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		Launch Enclave's MASKEDATTRIBUTES: This should be set to the LE's ATTRIBUTES masked with ATTRIBUTEMASK of the LE's KEYREQUEST.
KEYID (or HMAC)	256	32		If VALID[1]=0, this is the value for key wear-out protection. If VALID[1]=1, this field is the value of the 256-bit Hash Message Authentication Code of the token using EINITTOKEN_KEY.
MAC	288	16		If VALID[1]=0, this field is used for the 128-bit Message Authentication Code on EINITTOKEN using EINITTOKEN_KEY. If VALID[1]=1, this field must be zero.

7.5.5 REPORT2 (REPORT2)

The REPORT2 structure is the output of the ENCLU[EREPORT2] leaf function. REPORT2.TEEINFO includes enclave properties, measurements, and important values from the SECS. The measurements are integrity protected using an HMAC in REPORTMACSTRUCT. The REPORTMACSTRUCT.TEEINFOHASH is a hash of the REPORT2.TEEINFO establishing a binding between the TEEINFO and the MAC.

Table 7-7. Layout of REPORT2

Field	OFFSET (Bytes)	Size (Bytes)	Description
REPORTMAC STRUCT	0	256	Report MAC structure.
TEEINFO	256	512	Enclave measurements.
RESERVED	768	256	Zero.

7.5.5.1 REPORT MAC STRUCTURE (REPORTMACSTRUCT)

REPORTMACSTRUCT is the first field in REPORT2 and SEAMREPORT. It is MAC-protected and contains hashes of the remainder of the report structure, which includes the TEE's measurements and the measurements of additional TCB elements not reflected in REPORTMACSTRUCT.CPUSVN.

Software verifying a REPORT2 or SEAMREPORT structure should first confirm that REPORTMACSTRUCT.TEE_INFO_HASH equals the SHA384(TEEINFO). If REPORTMACSTRUCT.TEE_TCB_INFO_HASH is not zero, then software confirms REPORTMACSTRUCT.TEE_TCB_INFO_HASH equals the SHA384(TEE_TCB_INFO). Then, software uses ENCLU[EVERIFYREPORT2] to verify the integrity of the REPORTMACSTRUCT. If all checks pass, the measurements in the structure describe a TEE on this platform.

The size of REPORTMACSTRUCT is 256 bytes.

Table 7-8. Layout of REPORTMACSTRUCT

Field	OFFSET (Bytes)	Size (Bytes)	Description
REPORTTYPE	0	4	Type header structure.
RESERVED	4	12	Zero.
CPUSVN	16	16	Security version number of the processor.
TEE_TCB_INFO_HASH	32	48	SEAM: SHA384 of TCC_TCB_INFO SGX: 0000 … 0000H
TEE_INFO_HASH	80	48	SHA384 of TEE type-specific structure containing the measurements of that TEE.
REPORTDATA	128	64	Set of data used for communication between the TEE and relying party.
RESERVED	192	32	Must be zero.
MAC	224	32	HMAC-SHA256 over the REPORTMACSTRUCT using the Report2 key.

7.5.5.2 REPORT TYPE (REPORTTYPE)

REPORTTYPE describes the contents of REPORT2 or SEAMREPORT and is found in the report's REPORTMACSTRUCT.

Table 7-9. Layout of REPORT2

Field	OFFSET (Bytes)	Size (Bytes)	Description
TYPE	0	1	Trusted Execution Environment (TEE) Type: 00H: SGX 81H: SEAM-based TEE
SUBTYPE	1	1	TYPE-specific subtype.
VERSION	2	1	TYPE-specific version.
RESERVED	3	1	Zero.

7.5.5.3 SGX TEEINFO (TEEINFO)

Table 7-10. Layout of REPORTMACSTRUCT

Field	OFFSET (Bytes)	Size (Bytes)	Description
MISCSELECT	0	4	SSA Frame extended feature set.
ISVPRODID	4	2	Enclave Product ID.
ISVSVN	6	2	Security version of the enclave.
CONFIGSVN	8	2	Value provided by software indicating expected security version of enclave's configuration.
RESERVED	10	6	Zero.
ISVEXTPRODID	16	16	Value of SECS.ISVEXTPRODID.
ATTRIBUTES	32	16	Values of the attributes flags for the enclave. See Section 7.5.2.
MRENCLAVE	48	32	Value of SECS.MRENCLAVE.
RESERVED	80	32	Zero.
MRSIGNER	112	32	Value of SECS.MRSIGNER.
RESERVED	144	32	Zero.
CONFIGID	176	64	Value provided by software to identify enclave's configuration.
ISVFAMILYID	240	16	Value of SECS.ISVFAMILYID.
MRENCLAVE384	256	48	SECS.MRENCLAVE384
MRSIGNER384	304	48	SECS.MRSIGNER384
RESERVED	352	160	Zero.

7.5.6 REPORT2.REPORTMACSTRUCT.REPORTDATA

The REPORTDATA is a 64-byte data structure provided by the enclave and included in REPORT2. It can be used to securely pass information from the enclave to the target enclave.

7.6 INTEL® SGX INSTRUCTION SYNTAX AND OPERATION

ENCLS, ENCLU, and ENCLV instruction mnemonics for all leaf functions are covered in this section.

For all instructions, the value of CS.D is ignored; addresses and operands are 64 bits in 64-bit mode and are otherwise 32 bits. Aside from EAX specifying the leaf number as input, each instruction leaf may require all or some subset of the RBX/RCX/RDX as input parameters. Some leaf functions may return data or status information in one or more of the general purpose registers.

7.6.1 ENCLU Register Usage Summary

Table 7-11 summarizes the implicit register usage of user mode enclave instructions.

Table 7-11. Register Usage of Unprivileged Enclave Instruction Leaf Functions

Instr. Leaf	EAX	RBX	RCX	RDX
EREPORT	00H (In)	TARGETINFO (In, EA)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)
EGETKEY	01H (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EENTER	02H (In)	TCS (In, EA)	AEP (In, EA)	
	RBX.CSSA (Out)		Return (Out, EA)	
ERESUME	03H (In)	TCS (In, EA)	AEP (In, EA)	
EEXIT	04H (In)	Target (In, EA)	Current AEP (Out)	
EACCEPT	05H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EMODPE	06H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	
EACCEPTCOPY	07H (In)	SECINFO (In, EA)	EPCPAGE (In, EA)	EPCPAGE (In, EA)
EDECCSSA	09H (In)			
EREPORT2	0AH (In)	REPORTDATA (In, EA)	OUTPUTDATA (In, EA)	REPORT2 Subtype
EGETKEY256	0BH (In)	KEYREQUEST (In, EA)	KEY (In, EA)	
EA: Effective Address				

7.6.2 Information and Error Codes

Information and error codes are reported by various instruction leaf functions to show an abnormal termination of the instruction or provide information which may be useful to the developer. Table 7-12 shows the various codes and the instruction which generated the code. Details of the meaning of the code is provided in the individual instruction.

Table 7-12. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
No Error	0	
SGX_INVALID_SIG_STRUCT	1	EINIT
SGX_INVALID_ATTRIBUTE	2	EINIT, EGETKEY
SGX_BLKSTATE	3	EBLOCK
SGX_INVALID_MEASUREMENT	4	EINIT
SGX_NOTBLOCKABLE	5	EBLOCK
SGX_PG_INVLD	6	EBLOCK
SGX_EPC_PAGE_CONFLICT	7	EBLOCK, EMODPR, EMODT, EUPDATESVN
SGX_INVALID_SIGNATURE	8	EINIT
SGX_MAC_COMPARE_FAIL	9	ELDB, ELDU
SGX_PAGE_NOT_BLOCKED	10	EWB
SGX_NOT_TRACKED	11	EWB, EACCEPT
SGX_VA_SLOT_OCCUPIED	12	EWB
SGX_CHILD_PRESENT	13	EWB, EREMOVE
SGX_ENCLAVE_ACT	14	EREMOVE
SGX_ENTRYEPOCH_LOCKED	15	EBLOCK

Table 7-12. Error or Information Codes for Intel® SGX Instructions

Name	Value	Returned By
SGX_INVALID_EINITTOKEN	16	EINIT
SGX_PREV_TRK_INCMPL	17	ETRACK
SGX_PG_IS_SECS	18	EBLOCK
SGX_PAGE_ATTRIBUTES_MISMATCH	19	EACCEPT, EACCEPTCOPY
SGX_PAGE_NOT_MODIFIABLE	20	EMODPR, EMODT
SGX_PAGE_NOT_DEBUGGABLE	21	EDBGRD, EDBGWR
SGX_INSUFFICIENT_ENTROPY	29	EUPDATESVN
SGX_EPC_NOT_READY	30	EUPDATESVN
SGX_NO_UPDATE	31	EUPDATESVN
SGX_INVALID_CPUSVN	32	EINIT, EGETKEY
SGX_INVALID_REPORT_SUBTYPE	33	EREPORT2
SGX_INVALID_ISVSVN	64	EGETKEY
SGX_UNMASKED_EVENT	128	EINIT
SGX_INVALID_KEYNAME	256	EGETKEY

7.6.3 Internal CREGs

The CREGs as shown in Table 5-4 are hardware specific registers used in this document to indicate values kept by the processor. These values are used while executing in enclave mode or while executing an Intel SGX instruction. These registers are not software visible and are implementation specific. The values in Table 7-13 appear at various places in the pseudo-code of this document. They are used to enhance understanding of the operations.

Table 7-13. List of Internal CREG

Name	Size (Bits)	Scope
CR_ENCLAVE_MODE	1	LP
CR_DBGOPTIN	1	LP
CR_TCS_LA	64	LP
CR_TCS_PA	64	LP
CR_ACTIVE_SECS	64	LP
CR_ELRANGE	128	LP
CR_SAVE_TF	1	LP
CR_SAVE_FS	64	LP
CR_GPR_PA	64	LP
CR_XSAVE_PAGE_n	64	LP
CR_SAVE_DR7	64	LP
CR_SAVE_PERF_GLOBAL_CTRL	64	LP
CR_SAVE_DEBUGCTL	64	LP
CR_SAVE_PEBS_ENABLE	64	LP
CR_CPUSVN	128	PACKAGE
CR_SGXOWNEREPOCH	128	PACKAGE
CR_SAVE_XCR0	64	LP
CR_SGX_ATTRIBUTES_MASK	128	LP
CR_PAGING_VERSION	64	PACKAGE

Table 7-13. List of Internal CREG

Name	Size (Bits)	Scope
CR_VERSION_THRESHOLD	64	PACKAGE
CR_NEXT_EID	64	PACKAGE
CR_BASE_PK	256	PACKAGE
CR_SEAL_FUSES	128	PACKAGE
CR_CET_SAVE_AREA_PA	64	LP
CR_ENCLAVE_SS_TOKEN_PA	64	LP
CR_SAVE_IA32_U_CET	64	LP
CR_SAVE_SSP	64	LP
CR_REPORT_MAC_KEY	256	PACKAGE

7.6.4 Intel® SGX Instructions Concurrency Tables

Table 7-14. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions	
			Access	On Conflict
EACCEPT	Target	[DS:RCX]	Shared	#GP
	SECINFO	[DS:RBX]	Concurrent	
EACCEPTCOPY	Target	[DS:RCX]	Concurrent	
	Source	[DS:RDX]	Concurrent	
	SECINFO	[DS:RBX]	Concurrent	
EADD	Target	[DS:RCX]	Exclusive	#GP
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP
EAUG	Target	[DS:RCX]	Exclusive	#GP
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP
EBLOCK	Target	[DS:RCX]	Shared	SGX_EPC_PAGE_CONFLICT
ECREATE	SECS	[DS:RCX]	Exclusive	#GP
EDBGRD	Target	[DS:RCX]	Shared	#GP
EDBGWR	Target	[DS:RCX]	Shared	#GP
EENTER	TCS	[DS:RBX]	Shared	#GP
EEXIT			Concurrent	
EEXTEND	Target	[DS:RCX]	Shared	#GP
	SECS	[DS:RBX]	Concurrent	
EGETKEY/EGETKEY256	KEYREQUEST	[DS:RBX]	Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent	
EINIT	SECS	[DS:RCX]	Shared	#GP

Table 7-14. Base Concurrency Restrictions

Leaf	Parameter		Base Concurrency Restrictions	
			Access	On Conflict
ELDB/ELDU	Target	[DS:RCX]	Exclusive	#GP
	VA	[DS:RDX]	Shared	#GP
	SECS	[DS:RBX]PAGEINFO.SECS	Shared	#GP
EMODPE	Target	[DS:RCX]	Concurrent	
	SECINFO	[DS:RBX]	Concurrent	
EMODPR	Target	[DS:RCX]	Shared	#GP
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT
EPA	VA	[DS:RCX]	Exclusive	#GP
EREMOVE	Target	[DS:RCX]	Exclusive	#GP
EREPORT	TARGETINFO	[DS:RBX]	Concurrent	
	REPORTDATA	[DS:RCX]	Concurrent	
	OUTPUTDATA	[DS:RDX]	Concurrent	
EREPORT2	REPORTDATA	[DS:RBX]	Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent	
ERESUME	TCS	[DS:RBX]	Shared	#GP
ETRACK	SECS	[DS:RCX]	Shared	#GP
EWB	Source	[DS:RCX]	Exclusive	#GP
	VA	[DS:RDX]	Shared	#GP

Table 7-15. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EACCEPT	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EACCEPTCOPY	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	Source	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EADD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Exclusive	#GP	Concurrent	
EAUG	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EBLOCK	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ECREATE	SECS	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EDBGRD	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	

Table 7-15. Additional Concurrency Restrictions

Leaf	Parameter		Additional Concurrency Restrictions					
			vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
			Access	On Conflict	Access	On Conflict	Access	On Conflict
EDBGWR	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EENTER	TCS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EEXIT			Concurrent		Concurrent		Concurrent	
EEXTEND	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	
EGETKEY/EGETKEY256	KEYREQUEST	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EINIT	SECS	[DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	
ELDB/ELDU	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS	[DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	
EMODPE	Target	[DS:RCX]	Exclusive	#GP	Concurrent		Concurrent	
	SECINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
EMODPR	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	
EMODT	Target	[DS:RCX]	Exclusive	SGX_EPC_PAGE_CONFLICT	Concurrent		Concurrent	
EPA	VA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREMOVE	Target	[DS:RCX]	Concurrent		Concurrent		Concurrent	
EREPORT	TARGETINFO	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RDX]	Concurrent		Concurrent		Concurrent	
EREPOR2	REPORTDATA	[DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA	[DS:RCX]	Concurrent		Concurrent		Concurrent	
ERESUME	TCS	[DS:RBX]	Concurrent		Concurrent		Concurrent	
ETRACK	SECS	[DS:RCX]	Concurrent		Concurrent		Exclusive	SGX_EPC_PAGE_CONFLICT[1]
EUPDATESVN	EPCM		Exclusive	SGX_EPC_PAGE_CONFLICT	Exclusive	SGX_EPC_PAGE_CONFLICT	Exclusive	SGX_EPC_PAGE_CONFLICT
EWB	Source	[DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA	[DS:RDX]	Concurrent		Concurrent		Concurrent	

NOTES:

1. SGX_CONFLICT VM Exit Qualification =TRACKING_RESOURCE_CONFLICT.

7.7 INTEL® SGX SYSTEM LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLS instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional implicit registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of each implicit register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EADD—Add a Page to an Uninitialized Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLS[EADD]	IR	V/V	SGX1	This leaf function adds a page to an uninitialized enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	EADD (In)	Address of a PAGEINFO (In)	Address of the destination EPC page (In)

Description

This leaf function copies a source page from non-enclave memory into the EPC, associates the EPC page with an SECS page residing in the EPC, and stores the linear address and security attributes in EPCM. As part of the association, the enclave offset and the security attributes are measured and extended into the SECS.MRENCLAVE. This instruction can only be executed when current privilege level is 0.

RBX contains the effective address of a PAGEINFO structure while RCX contains the effective address of an EPC page. The table below provides additional information on the memory parameter of EADD leaf function.

EADD Memory Parameter Semantics

PAGEINFO	PAGEINFO.SECS	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read/Write access permitted by Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

The instruction faults if any of the following:

EADD Faulting Conditions

The operands are not properly aligned.	Unsupported security attributes are set.
Refers to an invalid SECS.	Reference is made to an SECS that is locked by another thread.
The EPC page is locked by another thread.	RCX does not contain an effective address of an EPC page.
The EPC page is already valid.	If security attributes specifies a TCS and the source page specifies unsupported TCS values or fields.
The SECS has been initialized.	The specified enclave offset is outside of the enclave address space.

Concurrency Restrictions

Table 7-16. Base Concurrency Restrictions of EADD

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EADD	Target [DS:RCX]	Exclusive	#GP
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP

Table 7-17. Additional Concurrency Restrictions of EADD

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EADD	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGE-INFO.SECS	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EADD Operational Flow

Name	Type	Size (bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the page to be added.
SCRATCH_SECINFO	SECINFO	512	Scratch storage for holding the contents of DS:TMP_SECINFO.
TMP_LINADDR	Unsigned Integer	64	Holds the linear address to be stored in the EPCM and used to calculate TMP_ENCLAVEOFFSET.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA Buffer	1024	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_SECINFO := DS:RBX.SECINFO;
TMP_LINADDR := DS:RBX.LINADDR;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECS is not 4KByte aligned or
    DS:TMP_SECINFO is not 64Byte aligned or TMP_LINADDR is not 4KByte aligned)
    THEN #GP(0); FI;

IF (DS:TMP_SECS does not resolve within an EPC)
    THEN #PF(DS:TMP_SECS); FI;

SCRATCH_SECINFO := DS:TMP_SECINFO;

(* Check for misconfigured SECINFO flags*)
IF (SCRATCH_SECINFO reserved fields are not zero or
```

```
    ! (SCRATCH_SECINFO.FLAGS.PT is PT_REG or SCRATCH_SECINFO.FLAGS.PT is PT_TCS or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
    (SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1)) )
    THEN #GP(0); FI;

(* If PT_SS_FIRST/PT_SS_REST page types are requested then CR4.CET must be 1 *)
IF ( (SCRATCH_SECINFO.FLAGS.PT is PT_SS_FIRST OR
    SCRATCH_SECINFO.FLAGS.PT is PT_SS_REST) AND CR4.CET == 0)
    THEN #GP(0); FI;

(* Check the EPC page for concurrency *)
IF (EPC page is not available for EADD)
    THEN
        #GP(0); FI;

IF (EPCM(DS:RCX).VALID ≠ 0)
    THEN #PF(DS:RCX); FI;

(* Check the SECS for concurrency *)
IF (SECS is not available for EADD)
    THEN #GP(0); FI;

IF (EPCM(DS:TMP_SECS).VALID = 0 or EPCM(DS:TMP_SECS).PT ≠ PT_SECS)
    THEN #PF(DS:TMP_SECS); FI;

(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPGE[32767:0];

CASE (SCRATCH_SECINFO.FLAGS.PT)

    PT_TCS:
        IF (DS:RCX.RESERVED ≠ 0) #GP(0); FI;
        IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and
            ((DS:TCS.FSLIMIT & 0FFFH ≠ 0FFFH) or (DS:TCS.GSLIMIT & 0FFFH ≠ 0FFFH) )) #GP(0); FI;
        (* Ensure TCS.PREVSSP is zero *)
        IF (CPUID.(EAX=07H, ECX=00h):ECX[CET_SS] = 1) and (DS:RCX.PREVSSP != 0) #GP(0); FI;
        BREAK;
    PT_REG:
        IF (SCRATCH_SECINFO.FLAGS.W = 1 and SCRATCH_SECINFO.FLAGS.R = 0) #GP(0); FI;
        BREAK;
    PT_SS_FIRST:
    PT_SS_REST:
    (* SS pages cannot be created on first or last page of ELRANGE *)
    IF ( TMP_LINADDR = DS:TMP_SECS.BASEADDR or TMP_LINADDR = (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE - 0x1000) )
        THEN #GP(0); FI;
    IF ( DS:RCX[4087:0] != 0 ) #GP(0); FI;
    IF (SCRATCH_SECINFO.FLAGS.PT == PT_SS_FIRST)
        THEN
            (* Check that valid RSTORSSP token exists *)
            IF ( DS:RCX[4095:4088] != ((TMP_LINADDR + 0x1000) | DS:TMP_SECS.ATTRIBUTES.MODE64BIT) ) #GP(0); FI;
        ELSE
            (* Check the 8 bytes are zero *)
            IF ( DS:RCX[4095:4088] != 0 ) #GP(0); FI;
    FI;
```

```
    IF (SCRATCH_SECINFO.FLAGS.W = 0 OR SCRATCH_SECINFO.FLAGS.R = 0 OR
     SCRATCH_SECINFO.FLAGS.X = 1) #GP(0); FI;
        BREAK;
ESAC;

(* Check the enclave offset is within the enclave linear address space *)
IF (TMP_LINADDR < DS:TMP_SECS.BASEADDR or TMP_LINADDR ≥ DS:TMP_SECS.BASEADDR + DS:TMP_SECS.SIZE)
    THEN #GP(0); FI;

(* Check concurrency of measurement resource*)
IF (Measurement being updated)
    THEN #GP(0); FI;

(* Check if the enclave to which the page will be added is already in Initialized state *)
IF (DS:TMP_SECS already initialized)
    THEN #GP(0); FI;

(* For TCS pages, force EPCM.rwx bits to 0 and no debug access *)
IF (SCRATCH_SECINFO.FLAGS.PT = PT_TCS)
    THEN
        SCRATCH_SECINFO.FLAGS.R := 0;
        SCRATCH_SECINFO.FLAGS.W := 0;
        SCRATCH_SECINFO.FLAGS.X := 0;
        (DS:RCX).FLAGS.DBGOPTIN := 0; // force TCS.FLAGS.DBGOPTIN off
        DS:RCX.CSSA := 0;
        DS:RCX.AEP := 0;
        DS:RCX.STATE := 0;
FI;

(* Add enclave offset and security attributes to MRENCLAVE *)
TMP_ENCLAVEOFFSET := TMP_LINADDR - DS:TMP_SECS.BASEADDR;
TMPUPDATEFIELD[63:0] := 0000000044444145H; // "EADD"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[511:128] := SCRATCH_SECINFO[375:0]; // 48 bytes
TMPUPDATEFIELD[1023:512] := 0x00; // 64 bytes
IF (DS:TMP_SECS.ATTRIBUTES.SHA384 == 0)
    THEN
        DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD[511:0])
    ELSE
        DS:TMP_SECS.MRENCLAVE384 := SHA384UPDATE(DS:TMP_SECS.MRENCLAVE384, TMPUPDATEFIELD[1023:0]);
FI;

INC enclave's MRENCLAVE update counter;

(* Add enclave offset and security attributes to MRENCLAVE *)
EPCM(DS:RCX).R := SCRATCH_SECINFO.FLAGS.R;
EPCM(DS:RCX).W := SCRATCH_SECINFO.FLAGS.W;
EPCM(DS:RCX).X := SCRATCH_SECINFO.FLAGS.X;
EPCM(DS:RCX).PT := SCRATCH_SECINFO.FLAGS.PT;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_LINADDR;

(* associate the EPCPAGE with the SECS by storing the SECS identifier of DS:TMP_SECS *)
Update EPCM(DS:RCX) SECS identifier to reference DS:TMP_SECS identifier;
```

```
(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
	If the TCS page PREVSSP field is not zero.
	If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
	If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If an enclave memory operand is outside of the EPC.
	If an enclave memory operand is the wrong type.
	If a memory operand is locked.
	If the enclave is initialized.
	If the enclave's MRENCLAVE is locked.
	If the TCS page reserved bits are set.
	If the TCS page PREVSSP field is not zero.
	If the PT_SS_REST or PT_SS_REST page is the first or last page in the enclave.
	If the PT_SS_FIRST or PT_SS_REST page is not initialized correctly.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the EPC page is valid.

ECREATE—Create an SECS page in the Enclave Page Cache

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLS[ECREATE]	IR	V/V	SGX1	This leaf function begins an enclave build by creating an SECS page in EPC.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX
IR	ECREATE (In)	Address of a PAGEINFO (In)	Address of the destination SECS page (In)

Description

ENCLS[ECREATE] is the first instruction executed in the enclave build process. ECREATE copies an SECS structure outside the EPC into an SECS page inside the EPC. The internal structure of SECS is not accessible to software.

ECREATE will set up fields in the protected SECS and mark the page as valid inside the EPC. ECREATE initializes or checks unused fields.

Software sets the following fields in the source structure: SECS:BASEADDR, SECS:SIZE in bytes, ATTRIBUTES, CONFIGID, and CONFIGSVN. SECS:BASEADDR must be naturally aligned on an SECS.SIZE boundary. SECS.SIZE must be at least 2 pages (8192).

The source operand RBX contains an effective address of a PAGEINFO structure. PAGEINFO contains an effective address of a source SECS and an effective address of an SECINFO. The SECS field in PAGEINFO is not used.

The RCX register is the effective address of the destination SECS. It is an address of an empty slot in the EPC. The SECS structure must be page aligned. SECINFO flags must specify the page as an SECS page.

ECREATE Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.SECINFO	EPCPAGE
Read access permitted by Non Enclave	Read access permitted by Non Enclave	Read access permitted by Non Enclave	Write access permitted by Enclave

ECREATE will fault if the SECS target page is in use; already valid; outside the EPC. It will also fault if addresses are not aligned; unused PAGEINFO fields are not zero.

If the amount of space needed to store the SSA frame is greater than the amount specified in SECS.SSAFRAME-SIZE, a #GP(0) results. The amount of space needed for an SSA frame is computed based on DS:TMP_-SECS.ATTRIBUTES.XFRM size. Details of computing the size can be found in “Interactions with the Processor Extended State and Miscellaneous State” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3.

Concurrency Restrictions

Table 7-18. Base Concurrency Restrictions of ECREATE

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
ECREATE	SECS [DS:RCX]	Exclusive	#GP

Table 7-19. Additional Concurrency Restrictions of ECREATE

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ECREATE	SECS [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ECREATE Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Effective Address	32/64	Effective address of the SECS source page.
TMP_SECS	Effective Address	32/64	Effective address of the SECS destination page.
TMP_SECINFO	Effective Address	32/64	Effective address of an SECINFO structure which contains security attributes of the SECS page to be added.
TMP_XSIZE	SSA Size	64	The size calculation of SSA frame.
TMP_MISC_SIZE	MISC Field Size	64	Size of the selected MISC field components.
TMPUPDATEFIELD	SHA Buffer	1023	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
IF (DS:RBX is not 32Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX is not 4KByte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECINFO := DS:RBX.SECINFO;

IF (DS:TMP_SRCPGE is not 4KByte aligned or DS:TMP_SECINFO is not 64Byte aligned)
    THEN #GP(0); FI;

IF (DS:RBX.LINADDR ! = 0 or DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

(* Check for misconfigured SECINFO flags*)
IF (DS:TMP_SECINFO reserved fields are not zero or DS:TMP_SECINFO.FLAGS.PT ≠ PT_SECS)
    THEN #GP(0); FI;

TMP_SECS := RCX;

IF (EPC entry in use)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;
```

```
(* Copy 4KBytes from source page to EPC page*)
DS:RCX[32767:0] := DS:TMP_SRCPGE[32767:0];

(* Check lower 2 bits of XFRM are set *)
IF ( ( DS:TMP_SECS.ATTRIBUTES.XFRM BitwiseAND 03H) ≠ 03H)
    THEN #GP(0); FI;

IF (XFRM is illegal)
    THEN #GP(0); FI;

(* Check legality of CET_ATTRIBUTES *)
IF ((DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_ATTRIBUTES ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.CET = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_LEG_BITMAP_OFFSET ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[5:2] ≠ 0) ||
    (CPUID.(EAX=7, ECX=0):ECX[CET_SS] = 0 and DS:TMP_SECS.CET_ATTRIBUTES[1:0] ≠ 0) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1 and
     (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) not canonical) ||
    (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0 and
     (DS:TMP_SECS.BASEADDR + DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) & 0xFFFFFFFF00000000) ||
    (DS:TMP_SECS.CET_ATTRIBUTES.reserved fields not 0) or
     (DS:TMP_SECS.CET_LEG_BITMAP_OFFSET) is not page aligned))
    THEN
        #GP(0);
FI;

(* Make sure that the SECS does not have any unsupported MISCSELECT options*)
IF ( !(CPUID.(EAX=12H, ECX=0):EBX[31:0] & DS:TMP_SECS.MISCSELECT[31:0]) )
    THEN
        EPCM(DS:TMP_SECS).EntryLock.Release();
        #GP(0);
FI;

( * Compute size of MISC area *)
TMP_MISC_SIZE := compute_misc_region_size();

(* Compute the size required to save state of the enclave on async exit, see "SECS.SSAFRAMESIZE" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3. *)
TMP_XSIZE := compute_xsave_size(DS:TMP_SECS.ATTRIBUTES.XFRM) + GPR_SIZE + TMP_MISC_SIZE;

(* Ensure that the declared area is large enough to hold XSAVE and GPR stat *)
IF ( DS:TMP_SECS.SSAFRAMESIZE*4096 < TMP_XSIZE)
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.BASEADDR is not canonical) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.BASEADDR and 0FFFFFFFF00000000H) )
    THEN #GP(0); FI;

IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 0) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[7:0]) ) )
    THEN #GP(0); FI;
```

```
IF ( (DS:TMP_SECS.ATTRIBUTES.MODE64BIT = 1) and (DS:TMP_SECS.SIZE ≥ 2 ^ (CPUID.(EAX=12H, ECX=0):.EDX[15:8]) ) )
    THEN #GP(0); FI;

(* Enclave size must be at least 8192 bytes and must be power of 2 in bytes*)
IF (DS:TMP_SECS.SIZE < 8192 or popcnt(DS:TMP_SECS.SIZE) > 1)
    THEN #GP(0); FI;

(* Ensure base address of an enclave is aligned on size*)
IF ( ( DS:TMP_SECS.BASEADDR and (DS:TMP_SECS.SIZE-1) ) )
    THEN #GP(0); FI;

(* Ensure the SECS does not have any unsupported attributes*)
IF ( DS:TMP_SECS.ATTRIBUTES and (~CR_SGX_ATTRIBUTES_MASK) )
    THEN #GP(0); FI;

IF ( DS:TMP_SECS reserved fields are not zero)
    THEN #GP(0); FI;

(* Verify that CONFIGID/CONFIGSVN are not set with attribute *)
IF ( ((DS:TMP_SECS.CONFIGID ≠ 0) or (DS:TMP_SECS.CONFIGSVN ≠0)) AND (DS:TMP_SECS.ATTRIBUTES.KSS == 0 ))
    THEN #GP(0); FI;

Clear DS:TMP_SECS to Uninitialized;
DS:TMP_SECS.MRENCLAVE := SHA256INITIALIZE(DS:TMP_SECS.MRENCLAVE);
DS:TMP_SECS.MRSIGNER := 0;
DS:TMP_SECS.MRSIGNER384 := 0;
DS:TMP_SECS.ISVSVN := 0;
DS:TMP_SECS.ISVPRODID := 0;

(* Initialize hash updates etc*)
Initialize enclave's MRENCLAVE update counter;

(* Add "ECREATE" string and SECS fields to MRENCLAVE *)
TMPUPDATEFIELD[63:0] := 0045544145524345H; // "ECREATE"
TMPUPDATEFIELD[95:64] := DS:TMP_SECS.SSAFRAMESIZE;
TMPUPDATEFIELD[159:96] := DS:TMP_SECS.SIZE;
IF (CPUID.(EAX=7, ECX=0):EDX[CET_IBT] = 1)
    THEN
        TMPUPDATEFIELD[223:160] := DS:TMP_SECS.CET_LEG_BITMAP_OFFSET;
    ELSE
        TMPUPDATEFIELD[223:160] := 0;
FI;
TMPUPDATEFIELD[1023:160] := 0;

IF (DS:TMP_SECS.ATTRIBUTES.SHA384 == 0)
    THEN
        DS:TMP_SECS.MRENCLAVE := SHA256UPDATE(DS:TMP_SECS.MRENCLAVE, TMPUPDATEFIELD[511:0])
    ELSE
        DS:TMP_SECS.MRENCLAVE384 := SHA384UPDATE(DS:TMP_SECS.MRENCLAVE384, TMPUPDATEFIELD[1023:0])
FI;

INC enclave's MRENCLAVE update counter;

(* Set EID *)
```

```
DS:TMP_SECS.EID := LockedXAdd(CR_NEXT_EID, 1);

(* Set the EPCM entry, first create SECS identifier and store the identifier in EPCM *)
EPCM(DS:TMP_SECS).PT := PT_SECS;
EPCM(DS:TMP_SECS).ENCLAVEADDRESS := 0;
EPCM(DS:TMP_SECS).R := 0;
EPCM(DS:TMP_SECS).W := 0;
EPCM(DS:TMP_SECS).X := 0;

(* Set EPCM entry fields *)
EPCM(DS:RCX).BLOCKED := 0;
EPCM(DS:RCX).PENDING := 0;
EPCM(DS:RCX).MODIFIED := 0;
EPCM(DS:RCX).PR := 0;
EPCM(DS:RCX).VALID := 1;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the reserved fields are not zero.
	If PAGEINFO.SECS is not zero.
	If PAGEINFO.LINADDR is not zero.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the SECS destination is outside the EPC.

64-Bit Mode Exceptions

#GP(0)	If a memory address is non-canonical form.
	If a memory operand is not properly aligned.
	If the reserved fields are not zero.
	If PAGEINFO.SECS is not zero.
	If PAGEINFO.LINADDR is not zero.
	If the SECS destination is locked.
	If SECS.SSAFRAMESIZE is insufficient.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the SECS destination is outside the EPC.

EEXTEND—Extend Uninitialized Enclave Measurement by 256 Bytes

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 06H ENCLS[EEXTEND]	IR	V/V	SGX1	This leaf function measures 256 bytes of an uninitialized enclave page.

Instruction Operand Encoding

Op/En	EAX	EBX	RCX
IR	EEXTEND (In)	Effective address of the SECS of the data chunk (In)	Effective address of a 256-byte chunk in the EPC (In)

Description

This leaf function updates the MRENCLAVE measurement register of an SECS with the measurement of an EXTEND string compromising of "EEXTEND" || ENCLAVEOFFSET || PADDING || 256 bytes of the enclave page. This instruction can only be executed when current privilege level is 0 and the enclave is uninitialized.

RBX contains the effective address of the SECS of the region to be measured. The address must be the same as the one used to add the page into the enclave.

RCX contains the effective address of the 256 byte region of an EPC page to be measured. The DS segment is used to create linear addresses. Segment override is not supported.

EEXTEND Memory Parameter Semantics

EPC[RCX]
Read access by Enclave

The instruction faults if any of the following:

EEXTEND Faulting Conditions

RBX points to an address not 4KBytes aligned.	RBX does not resolve to an SECS.
RBX does not point to an SECS page.	RBX does not point to the SECS page of the data chunk.
RCX points to an address not 256B aligned.	RCX points to an unused page or a SECS.
RCX does not resolve in an EPC page.	If SECS is locked.
If the SECS is already initialized.	May page fault.
CPL > 0.	

Concurrency Restrictions

Table 7-20. Base Concurrency Restrictions of EEXTEND

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EEXTEND	Target [DS:RCX]	Shared	#GP
	SECS [DS:RBX]	Concurrent	

Table 7-21. Additional Concurrency Restrictions of EEXTEND

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EEXTEND	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EEXTEND Operational Flow

Name	Type	Size (Bits)	Description
TMP_SECS		64	Physical address of SECS of the enclave to which source operand belongs.
TMP_ENCLAVEOFFSET	Enclave Offset	64	The page displacement from the enclave base address.
TMPUPDATEFIELD	SHA Buffer	1023	Buffer used to hold data being added to TMP_SECS.MRENCLAVE.

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

IF (DS:RBX is not 4096 Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RBX does not resolve to an EPC page)
    THEN #PF(DS:RBX); FI;

IF (DS:RCX is not 256Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* make sure no other Intel SGX instruction is accessing EPCM *)
IF (Other instructions accessing EPCM)
    THEN #GP(0); FI;

IF (EPCM(DS:RCX). VALID = 0)
    THEN #PF(DS:RCX); FI;

(* make sure that DS:RCX (DST) is pointing to a PT_REG or PT_TCS or PT_SS_FIRST or PT_SS_REST *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) and (EPCM(DS:RCX).PT ≠ PT_TCS)
  and (EPCM(DS:RCX).PT ≠ PT_SS_FIRST) and (EPCM(DS:RCX).PT ≠ PT_SS_REST))
    THEN #PF(DS:RCX); FI;

TMP_SECS := Get_SECS_ADDRESS();

IF (DS:RBX does not resolve to TMP_SECS)
    THEN #GP(0); FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction accessing MRENCLAVE) or (Other instructions checking or updating the initialized state of the SECS))
```

```
    THEN #GP(0); FI;

(* Calculate enclave offset *)
TMP_ENCLAVEOFFSET := EPCM(DS:RCX).ENCLAVEADDRESS - TMP_SECS.BASEADDR;
TMP_ENCLAVEOFFSET := TMP_ENCLAVEOFFSET + (DS:RCX & 0FFFH)

(* Add EEXTEND message and offset to MRENCLAVE *)
TMPUPDATEFIELD[63:0] := 00444E4554584545H; // "EEXTEND"
TMPUPDATEFIELD[127:64] := TMP_ENCLAVEOFFSET;
TMPUPDATEFIELD[1023:128] := 0; // 112 bytes

IF (TMP_SECS.ATTRIBUTES.SHA384 = 0)
    THEN
        TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, TMPUPDATEFIELD[511:0])
        INC enclave's MRENCLAVE update counter;

        (*Add 256 bytes to MRENCLAVE, 64 byte at a time *)
        TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[511:0] );
        TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1023: 512] );
        TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[1535: 1024] );
        TMP_SECS.MRENCLAVE := SHA256UPDATE(TMP_SECS.MRENCLAVE, DS:RCX[2047: 1536] );
        INC enclave's MRENCLAVE update counter by 4;
    ELSE
        TMP_SECS.MRENCLAVE384 := SHA256UPDATE(TMP_SECS.MRENCLAVE, TMPUPDATEFIELD[511:0])
        INC enclave's MRENCLAVE update counter;

        (*Add 256 bytes to MRENCLAVE, 128 byte at a time *)
        TMP_SECS.MRENCLAVE384 := SHA384UPDATE(TMP_SECS.MRENCLAVE384, DS:RCX[1023:0] );
        TMP_SECS.MRENCLAVE384 := SHA384UPDATE(TMP_SECS.MRENCLAVE384, DS:RCX[2047: 1024] );
        INC enclave's MRENCLAVE update counter by 4;
FI
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If the address in RBX is outside the DS segment limit.
	If RBX points to an SECS page which is not the SECS of the data chunk.
	If the address in RCX is outside the DS segment limit.
	If RCX points to a memory location not 256Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

64-Bit Mode Exceptions

#GP(0)	If RBX is non-canonical form.

	If RBX points to an SECS page which is not the SECS of the data chunk.
	If RCX is non-canonical form.
	If RCX points to a memory location not 256 Byte-aligned.
	If another instruction is accessing MRENCLAVE.
	If another instruction is checking or updating the SECS.
	If the enclave is already initialized.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If the address in RBX points to a non-EPC page.
	If the address in RCX points to a page which is not PT_TCS or PT_REG.
	If the address in RCX points to a non-EPC page.
	If the address in RCX points to an invalid EPC page.

EINIT—Initialize an Enclave for Execution

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 02H ENCLS[EINIT]	IR	V/V	SGX1	This leaf function initializes the enclave and makes it ready to execute enclave code.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EINIT (In)	Error code (Out)	Address of SIGSTRUCT (In)	Address of SECS (In)	Address of EINITTOKEN (In)

Description

This leaf function is the final instruction executed in the enclave build process. After EINIT, the MRENCLAVE measurement is complete, and the enclave is ready to start user code execution using the EENTER instruction.

EINIT takes the effective address of a SIGSTRUCT and EINITTOKEN. The SIGSTRUCT describes the enclave including MRENCLAVE, ATTRIBUTES, ISVSVN, a 3072 bit RSA key, and a signature using the included key. SIGSTRUCT must be populated with two values, q1 and q2. These are calculated using the formulas shown below:

q1 = floor(Signature^2 / Modulus);

q2 = floor((Signature^3 - q1 * Signature * Modulus) / Modulus);

The EINITTOKEN contains the MRENCLAVE, MRSIGNER, and ATTRIBUTES. These values must match the corresponding values in the SECS. If the EINITTOKEN was created with a debug launch key, the enclave must be in debug mode as well.

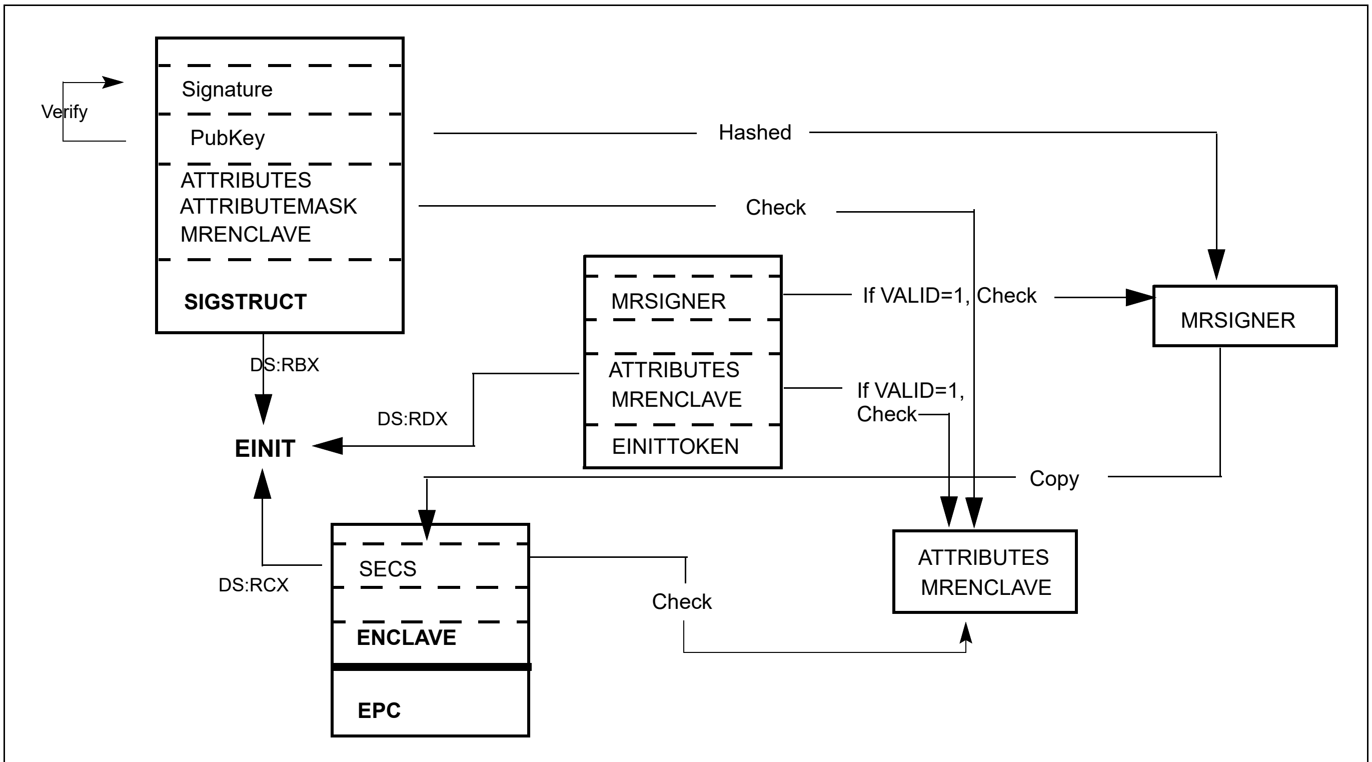


Figure 7-1. Relationships Between SECS, SIGSTRUCT, and EINITTOKEN

EINIT Memory Parameter Semantics

SIGSTRUCT	SECS	EINITTOKEN
Access by non-Enclave	Read/Write access by Enclave	Access by non-Enclave

EINIT performs the following steps, which can be seen in Figure 7-1:

1. Validates that SIGSTRUCT is signed using the enclosed public key.
2. Checks that the completed computation of SECS.MRENCLAVE equals SIGSTRUCT.HASHENCLAVE.
3. Checks that no controlled ATTRIBUTES bits are set in SIGSTRUCT.ATTRIBUTES unless the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
4. Checks that the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SIGSTRUCT.ATTRIBUTES equals the result of bitwise and-ing SIGSTRUCT.ATTRIBUTEMASK with SECS.ATTRIBUTES.
5. If EINITTOKEN.VALID is 0, checks that the SHA256 digest of SIGSTRUCT.MODULUS equals IA32_SGX_LEPUBKEYHASH.
6. If EINITTOKEN.VALID is 1, checks the validity of EINITTOKEN.
7. If EINITTOKEN.VALID is 1, checks that EINITTOKEN.MRENCLAVE equals SECS.MRENCLAVE.
8. If EINITTOKEN.VALID is 1 and EINITTOKEN.ATTRIBUTES.DEBUG is 1, SECS.ATTRIBUTES.DEBUG must be 1.
9. Commits SECS.MRENCLAVE, and sets SECS.MRSIGNER, SECS.ISVSVN, and SECS.ISVPRODID based on SIGSTRUCT.
10. Update the SECS as Initialized.

Periodically, EINIT polls for certain asynchronous events. If such an event is detected, it completes with failure code (ZF=1 and RAX = SGX_UNMASKED_EVENT), and RIP is incremented to point to the next instruction. These events includes external interrupts, non-maskable interrupts, system-management interrupts, machine checks, INIT signals, and the VMX-preemption timer. EINIT does not fail if the pending event is inhibited (e.g., external interrupts could be inhibited due to blocking by MOV SS blocking or by STI).

The following bits in RFLAGS are cleared: CF, PF, AF, OF, and SF. When the instruction completes with an error, RFLAGS.ZF is set to 1, and the corresponding error bit is set in RAX. If no error occurs, RFLAGS.ZF is cleared and RAX is set to 0.

The error codes are:

Table 7-22. EINIT Return Value in RAX

Error Code (see Table 7-12)	Description
No Error	EINIT successful.
SGX_INVALID_SIG_STRUCT	If SIGSTRUCT contained an invalid value.
SGX_INVALID_ATTRIBUTE	If SIGSTRUCT contains an unauthorized attributes mask.
SGX_INVALID_MEASUREMENT	If SIGSTRUCT contains an incorrect measurement. If EINITTOKEN contains an incorrect measurement.
SGX_INVALID_SIGNATURE	If signature does not validate with enclosed public key.
SGX_INVALID_LICENSE	If license is invalid.
SGX_INVALID_CPUSVN	If license SVN is unsupported.
SGX_UNMASKED_EVENT	If an unmasked event is received before the instruction completes its operation.

Concurrency Restrictions

Table 7-23. Base Concurrency Restrictions of EINIT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EINIT	SECS [DS:RCX]	Shared	#GP

Table 7-24. Additional Concurrency Restrictions of EINIT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EINIT	SECS [DS:RCX]	Concurrent		Exclusive	#GP	Concurrent	

Operation

Temp Variables in EINIT Operational Flow

Name	Type	Size	Description
TMP_SIG	SIGSTRUCT	1808 Bytes	Temp space for SIGSTRUCT.
TMP_TOKEN	EINITTOKEN	304 Bytes	Temp space for EINITTOKEN.
TMP_MRENCLAVEBUF		48 Bytes	Temp space for calculating MRENCLAVE or MRENCLAVE384.
TMP_MRSIGNERBUF		48 Bytes	Temp space for calculating MRSIGNER or MRSIGNER384.
CONTROLLED_ATTRIBUTES	ATTRIBUTES	16 Bytes	Constant mask of all ATTRIBUTE bits that can only be set for authorized enclaves.
TMP_KEYDEPENDENCIES	Buffer	224 Bytes	Temp space for key derivation.
TMP_KEYCONTEXT	Buffer		Temp space for key derivation context.
TMP_TOKENKEY		16 Bytes	Temp space for the derived TOKEN Key.
TMP_SIG_PADDING	PKCS Padding Buffer	352 Bytes	The value of the top 352 bytes from the computation of Signature[3] modulo MRSIGNER.
TMP_SIG_PADDING_BUFFER	Buffer	704 Bytes	Buffer to combine TMP_SIG_PADDING with standard PKCS padding.

```
(* make sure SIGSTRUCT and SECS are aligned *)
IF ( (DS:RBX is not 4KByte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* make sure the EINITTOKEN is aligned *)
IF (DS:RDX is not 512Byte Aligned)
    THEN #GP(0); FI;

(* make sure the SECS is inside the EPC *)
IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

TMP_SIG[14463:0] := DS:RBX[14463:0]; // 1808 bytes
```

```
TMP_TOKEN[2423:0] := DS:RDX[2423:0]; // 304 bytes

(* Verify SIGSTRUCT Header. *)
IF ( (TMP_SIG.HEADER ≠ 06000000E10000000000010000000000h) or
    ((TMP_SIG.VENDOR ≠ 0) and (TMP_SIG.VENDOR ≠ 00008086h) ) or
    (TMP_SIG HEADER2 ≠ 01010000600000006000000001000000h) or
    (TMP_SIG.EXPONENT   ≠ 00000003h) or (Reserved space is not 0's) ) or
    (TMP_SIG.SIGHASHTYPE ≠ 0 AND TMP_SIG.SIGHASHTYPE ≠ 1)

    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_SIG_STRUCT;
        GOTO EXIT;
FI;

(* Open "Event Window" Check for Interrupts. Verify signature using embedded public key, q1, and q2 with RSA-SHA256 or RSA-
SHA384 as indicated by SIGHASHTYPE. Save upper bytes of the PKCS1.5 encoded message into the TMP_SIG_PADDING, 352 bytes for
RSA-SHA256 and 336 bytes for RSA-SHA384. *)
IF (interrupt was pending) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_UNMASKED_EVENT;
    GOTO EXIT;
FI
IF (signature failed to verify) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_SIGNATURE;
    GOTO EXIT;
FI;
(*Close "Event Window" *)

(* make sure no other Intel SGX instruction is modifying SECS*)
IF (Other instructions modifying SECS)
    THEN #GP(0); FI;

IF ( (EPCM(DS:RCX). VALID = 0) or (EPCM(DS:RCX).PT ≠ PT_SECS) )
    THEN #PF(DS:RCX); FI;

(* Verify ISVFAMILYID is not used on an enclave with KSS disabled *)
IF ((TMP_SIG.ISVFAMILYID != 0) AND (DS:RCX.ATTRIBUTES.KSS == 0))
    THEN
     RFLAGS.ZF := 1;
     RAX := SGX_INVALID_SIG_STRUCT;
     GOTO EXIT;
FI;

(* make sure no other instruction is accessing MRENCLAVE or ATTRIBUTES.INIT *)
IF ( (Other instruction modifying MRENCLAVE) or (Other instructions modifying the SECS's Initialized state))
    THEN #GP(0); FI;

(* Calculate finalized version of MRENCLAVE *)
IF (DS:RCX.ATTRIBUTES.SHA384 == 0)
    THEN
        (* SHA256 algorithm requires one last update that compresses the length of the hashed message into the output SHA256
        digest *)
```

```
        TMP_ENCLAVEBUF := SHA256FINAL( DS:RCX.MRENCLAVE, enclave's MRENCLAVE update count *512);

        (* Verify MRENCLAVE from SIGSTRUCT *)
        IF ( (TMP_SIG.ENCLAVEHASH[2047:0] ≠ TMP_MRENCLAVEBUF[2047:0]) or
            (TMP_SIG.ENCLAVEHASH[3072:2048) ≠ 0) )
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_MEASUREMENT;
            GOTO EXIT;
    FI;

        TMP_MRSIGNERBUF := SHA256(TMP_SIG.MODULUS);
    ELSE
        (* SHA384 algorithm requires one last update that compresses the length of the hashed message into the output SHA384
        digest *)
        TMP_ENCLAVEBUF := SHA384FINAL( DS:RCX.MRENCLAVE384, enclave's MRENCLAVE update count *1024);

        (* Verify MRENCLAVE from SIGSTRUCT *)
        IF (TMP_SIG.ENCLAVEHASH[3072:0] ≠ TMP_MRENCLAVEBUF[3072:0])
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_MEASUREMENT;
            GOTO EXIT;
    FI;
        TMP_MRSIGNERBUF := SHA384(TMP_SIG.MODULUS);
FI;

// Verify that the signing hash used on the SIGSTRUCT & enclave measurement
// are compatible.
IF (TMP_SIG.SIGHASHTYPE ≠ DS:RCX.ATTRIBUTES.SHA384)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_SIGSTRUCT;
        GOTO EXIT;
FI;

(* If controlled ATTRIBUTES are set, SIGSTRUCT must be signed using an authorized key *)
CONTROLLED_ATTRIBUTES := 0000000000000020H;
IF ( ( (DS:RCX.ATTRIBUTES & CONTROLLED_ATTRIBUTES) ≠ 0) and
    ( ((DS:RCX.ATTRIBUTES.SHA384 = 0) AND
    (TMP_MRSIGNERBUF[2047:0] ≠ IA32_SGXLEPUBKEYHASH[2047:0]) OR
    (IA32_SGXLEPUBKEYHASH[3071:2048] ≠ 0)) OR
    ((DS:RCX.ATTRIBUTES.SHA384  = 1) AND
    (TMP_MRSIGNERBUF[3071:0] ≠ IA32_SGXLEPUBKEYHASH[3071:0])) ) )
THEN
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT;
FI;

(* Verify SIGSTRUCT.ATTRIBUTE requirements are met *)
IF ( (DS:RCX.ATTRIBUTES & TMP_SIG.ATTRIBUTEMASK) ≠ (TMP_SIG.ATTRIBUTE & TMP_SIG.ATTRIBUTEMASK) )
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_ATTRIBUTE;
```

```
    GOTO EXIT;
FI;

( *Verify SIGSTRUCT.MISCSELECT requirements are met *)
IF ( (DS:RCX.MISCSELECT & TMP_SIG.MISCMASK) ≠ (TMP_SIG.MISCSELECT & TMP_SIG.MISCMASK) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_ATTRIBUTE;
    GOTO EXIT
FI;

IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    IF ( DS:RCX.CET_ATTRIBUTES & TMP_SIG.CET_ATTRIBUTES_MASK ≠ TMP_SIG.CET_ATTRIBUTES &
     TMP_SIG.CET_ATTRIB-UTES_MASK )
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_ATTRIBUTE;
            GOTO EXIT
    FI;
FI;

(* If EINITTOKEN.VALID[0] is 0, verify the enclave is signed by an authorized key *)
IF (TMP_TOKEN.VALID[0] = 0)
    IF ( ((DS:RCX.ATTRIBUTES.SHA384 = 0) AND
        (TMP_MRSIGNERBUF[2047:0] ≠ IA32_SGXLEPUBKEYHASH[2047:0]) OR
        (IA32_SGXLEPUBKEYHASH[3071:2048] ≠ 0)) OR
        ((DS:RCX.ATTRIBUTES.SHA384 = 1) AND
        (TMP_MRSIGNERBUF[3071:0] ≠ IA32_SGXLEPUBKEYHASH[3071:0])))
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
    FI;
    GOTO COMMIT;
FI;

(* Debug Launch Enclave cannot launch Production Enclaves *)
IF ( (DS:RDX.MASKEDATTRIBUTESLE.DEBUG = 1) and (DS:RCX.ATTRIBUTES.DEBUG = 0) )
THEN
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;

(* Check reserve space in EINIT token includes reserved regions and upper bits in valid field *)
IF (TMP_TOKEN reserved space is not clear)
THEN
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINITTOKEN;
    GOTO EXIT;
FI;

(* EINIT token must not have been created by a configuration beyond the current CPU configuration *)
IF (TMP_TOKEN.CPUSVN must not be a configuration beyond CR_CPUSVN)
THEN
```

```
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_CPUSVN;
    GOTO EXIT;
FI;

(* Derive Launch key used to calculate EINITTOKEN.MAC *)
HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;
HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;
HARDCODED_PKCS1_5_PADDING[2831:2816] := 0100H;
HARDCODED_PKCS1_5_PADDING[5471:2832] := SignExtend314Byte(-1); // 314 bytes of 0FFH
HARDCODED_PKCS1_5_PADDING[5631:5472] := 3004000501020403650148866009060D30413000H;
HARDCODED_PKCS1_5_PADDING[5759:5632] := 0

HARDCODED_PKCS1_5_PADDING_HASH = SHA384(HARDCODED_PKCS1_5_PADDING);

TMP_KEYCONTEXT.KEYNAME := EINITTOKEN_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := TMP_TOKEN.ISVPRODIDLE;
TMP_KEYDEPENDENCIES.ISVSVN := TMP_TOKEN.ISVSVNLE;
TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_TOKEN.MASKEDATTRIBUTESLE;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
TMP_KEYDEPENDENCIES.MRSIGNER := IA32_SGXLEPUBKEYHASH;
TMP_KEYDEPENDENCIES.KEYID := TMP_TOKEN.KEYID;
TMP_KEYDEPENDENCIES.MISCSELECT := TMP_TOKEN.MASKEDMISCSELECTLE;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING_HASH;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := 0;
TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_TOKEN.CET_MASKED_ATTRIBUTES_ LE;
    TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
FI;

IF (TMP_TOKEN.VALID.LE256 = 0)
THEN
    (* Calculate the derived key*)
    TMP_TOKENKEY := derivekey(TMP_EINITTOKEN.CPUSVNLE,
                                    TRUE, // Include Seal Fuses
                                    TRUE, // Include Owner Epoch
                                    SHA384(TMP_KEYDEPENDENCIES),
                                    TMP_KEYCONTEXT,
                                    128);

    (* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the Launch
    Enclave. Only 192 bytes of EINITTOKEN are CMACed *)
    IF (TMP_TOKEN.MAC ≠ CMAC(TMP_TOKENKEY, TMP_TOKEN[1535:0] ) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
```

```
    FI;
ELSE
    THEN
        (* Calculate the derived key*)
        TMP_TOKENKEY := derivekey(TMP_TOKEN.CPUSVNLE,
                                  TRUE, // Include Seal Fuses
                                  TRUE, // Include Owner Epoch
                                  SHA384(TMP_KEYDEPENDENCIES),
                                  TMP_KEYCONTEXT,
                                  256);

        (* Verify EINITTOKEN was generated using this CPU's Launch key and that it has not been modified since issuing by the
        Launch Enclave. Only 192 bytes of EINITTOKEN are CMACed *)
        (* Note: 256 bit HMAC is stored in 256 bit KEYID field since KEYID is not needed, and MAC field is only 128 bits long. *)
        IF (TMP_TOKEN.KEYID ≠ HMAC(TMP_EINITTOKENKEY, TMP_TOKEN[1535:0]) )
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_EINITTOKEN;
                GOTO EXIT;
        FI;
FI;

(* Ensure token meets Launch control enabling type *)
IF (IA32_SGXLECONFIG[0] ≠ TMP_EINITTOKEN.VALID.LE256)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_EINITTOKEN;
        GOTO EXIT;
FI;

(* Verify EINITTOKEN (RDX) is for this enclave *)
IF (TMP_SECS.ATTRIBUTES.SHA384 = 0)
    IF ((TMP_TOKEN.MRENCLAVE[2047:0] != TMP_MRENCLAVEBUF[2047:0]) OR
        (TMP_TOKEN.MRENCLAVE[3071:2048] != 0) OR
        (TMP_TOKEN.MRSIGNER[2047:0]  != TMP_MRSIGNERBUF[2047:0]) OR
        (TMP_TOKEN.MRSIGNER[3071:2048] != 0))
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_MEASUREMENT;
        GOTO EXIT;
    FI;
ELSE
    IF ((TMP_EINITTOKEN.MRENCLAVE[3071:0] != TMP_MRENCLAVEBUF[3071:0]) OR
        (TMP_EINITTOKEN.MRSIGNER[3071:0]  != TMP_MRSIGNERBUF[3071:0])
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_MEASUREMENT;
        GOTO EXIT;
    FI;
FI;

(* Verify ATTRIBUTES in EINITTOKEN are the same as the enclave's *)
```

```
IF (TMP_TOKEN.ATTRIBUTES ≠ DS:RCX.ATTRIBUTES)
    RFLAGS.ZF := 1;
    RAX := SGX_INVALID_EINIT_ATTRIBUTE;
    GOTO EXIT;
FI;

COMMIT:
(* Commit changes to the SECS; Set ISVPRODID, ISVSVN, MRSIGNER, INIT ATTRIBUTE fields in SECS (RCX) *)
IF (DS:RCX.ATTRIBUTES.SHA384 == 0)
    THEN
        (* Use MRENCLAVE and MRSIGNER, zero MRENCLAVE384 and MRSIGNER384. *)
        DS:RCX.MRENCLAVE384 := 0;
        DS:RCX.MRENCLAVE := TMP_MRENCLAVEBUF[2047:0];
         (* MRSIGNER stores a SHA256 in little endian implemented natively on x86. *)
        DS:RCX.MRSIGNER384 := 0;
        DS:RCX.MRSIGNER := TMP_MRSIGNERBUF[2047:0];

        (* Padding buff strategy is a correctly signed enclave using either SHA256 or SHA384. *)
        (* Results in a Padding Buff = PKCS (SHA256 padding || PKCS SHA384 padding) *)
        (* Populate Sig Padding Buffer with enclave's actual PKCS SHA256 padding and standard PKCS SHA384 padding. *)
        TMP_SIG_PADDING_BUFFER[351:0B]  = TMP_SIG_PADDING;
        TMP_SIG_PADDING_BUFFER[2831:2816] := 0100H;
        TMP_SIG_PADDING_BUFFER[5471:2832] := SignExtend314Byte(-1); // 314 bytes of 0FFH
        TMP_SIG_PADDING_BUFFER [5631:5472] := 3004000501020403650148866009060D30413000H;
        TMP_SIG_PADDING_BUFFER[5759:5632] := 0;

ELSE
    THEN
        // Use MRENCLAVE384 and MRSIGNER384, compute MRENCLAVE and MRSIGNER
        DS:RCX.MRENCLAVE384 := TMP_MRENCLAVEBUF[3071:0];
        DS:RCX.MRENCLAVE := SHA256(TMP_MRENCLAVEBUF[3071:0]);
        DS:RCX.MRSIGNER384 := TMP_MRSIGNERBUF[3071:0];
        DS:RCX.MRSIGNER := SHA256(TMP_SIG.MODULUS);

        (* Padding buff strategy is a correctly signed enclave using either SHA256 or SHA384. *)
        (* Results in a Padding Buff = PKCS (SHA256 padding || PKCS SHA384 padding) *)
        (* Populate Sig Padding Buffer with standard PKCS SHA256 padding and enclave's actual PKCS SHA384 padding. *)
        TMP_SIG_PADDING_BUFFER[15:0] := 0100H;
        TMP_SIG_PADDING_BUFFER[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH
        TMP_SIG_PADDING_BUFFER[2815:2656] := 2004000501020403650148866009060D30313000H;
        TMP_SIG_PADDING_BUFFER[5631:2816] := TMP_SIG_PADDING;
        TMP_SIG_PADDING_BUFFER[5759:5632] := 0
FI;

DS:RCX.ISVEXTPRODID := TMP_SIG.ISVEXTPRODID;
DS:RCX.ISVPRODID := TMP_SIG.ISVPRODID;
DS:RCX.ISVSVN := TMP_SIG.ISVSVN;
DS:RCX.ISVFAMILYID := TMP_SIG.ISVFAMILYID;
DS:RCX.PADDING := SHA384(TMP_SIG_PADDING_BUFFER);

(* Mark the SECS as initialized *)
Update DS:RCX to initialized;

(* Set RAX and ZF for success*)
```

```
    RFLAGS.ZF := 0;
    RAX := 0;
EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set and RAX contains the error code. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand is not properly aligned.
	If another instruction is modifying the SECS.
	If the enclave is already initialized.
	If the SECS.MRENCLAVE is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RCX does not resolve in an EPC page.
	If the memory address is not a valid, uninitialized SECS.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is not properly aligned.
	If another instruction is modifying the SECS.
	If the enclave is already initialized.
	If the SECS.MRENCLAVE is in use.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If RCX does not resolve in an EPC page.
	If the memory address is not a valid, uninitialized SECS.

ELDB/ELDU—Load an EPC Page and Mark its State

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 07H ENCLS[ELDB]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as blocked.
EAX = 08H ENCLS[ELDU]	IR	V/V	SGX1	This leaf function loads, verifies an EPC page and marks the page as unblocked.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	ELDB/ELDU (In)	Return error code (Out)	Address of the PAGEINFO (In)	Address of the EPC page (In)	Address of the version-array slot (In)

Description

This leaf function copies a page from regular main memory to the EPC. As part of the copying process, the page is cryptographically authenticated and decrypted. This instruction can only be executed when current privilege level is 0.

The ELDB leaf function sets the BLOCK bit in the EPCM entry for the destination page in the EPC after copying. The ELDU leaf function clears the BLOCK bit in the EPCM entry for the destination page in the EPC after copying.

RBX contains the effective address of a PAGEINFO structure; RCX contains the effective address of the destination EPC page; RDX holds the effective address of the version array slot that holds the version of the page.

The table below provides additional information on the memory parameter of ELDB/ELDU leaf functions.

ELDB/ELDU Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	PAGEINFO.SECS	EPCPAGE	Version-Array Slot
Non-enclave read access	Non-enclave read access	Non-enclave read access	Enclave read/write access	Read/Write access permitted by Enclave	Read/Write access permitted by Enclave

The error codes are:

Table 7-25. ELDB/ELDU Return Value in RAX

Error Code (see Table 7-12)	Description
No Error	ELDB/ELDU successful.
SGX_MAC_COMPARE_FAIL	If the MAC check fails.

Concurrency Restrictions

Table 7-26. Base Concurrency Restrictions of ELDB/ELDU

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
ELDB/ELDU	Target [DS:RCX]	Exclusive	#GP
	VA [DS:RDX]	Shared	#GP
	SECS [DS:RBX]PAGEINFO.SECS	Shared	#GP

Table 7-27. Additional Concurrency Restrictions of ELDB/ELDU/ELDBC/ELBUC

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
ELDB/ELDU	Target [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Concurrent	
	SECS [DS:RBX]PAGEINFO.SECS	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in ELDB/ELDU Operational Flow

Name	Type	Size (Bits)	Description
TMP_SRCPGE	Memory page	4KBytes	
TMP_SECS	Memory page	4KBytes	
TMP_PCMD	PCMD	128 Bytes	
TMP_HEADER	MACHEADER	128 Bytes	
TMP_VER	UINT64	64	
TMP_MAC	UINT128	128	
TMP_PG_KEY	UINT256	256	Page encryption/MAC key.
SCRATCH_PCMD	PCMD	128 Bytes	

```
(* Check PAGEINFO and EPCPAGE alignment *)
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

(* Check VASLOT alignment *)
IF (DS:RDX is not 8Byte aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
TMP_SECS := DS:RBX.SECS;
TMP_PCMD := DS:RBX.PCMD;

(* Check alignment of PAGEINFO (RBX) linked parameters. Note: PCMD pointer is overlaid on top of PAGEINFO.SECINFO field *)
IF ( (DS:TMP_PCMD is not 128Byte aligned) or (DS:TMP_SRCPGE is not 4KByte aligned) )
    THEN #GP(0); FI;

(* Check concurrency of EPC by other Intel SGX instructions *)
IF (other instructions accessing EPC)
```

```
    THEN
        #GP(0); FI;

(* Check concurrency of EPC and VASLOT by other Intel SGX instructions *)
IF (Other instructions modifying VA slot)
    THEN
        #GP(0); FI;

(* Verify EPCM attributes of EPC page, VA, and SECS *)
IF (EPCM(DS:RCX).VALID = 1)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~0FFFH).PT ≠ PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Copy PCMD into scratch buffer *)
SCRATCH_PCMD[1023: 0] := DS:TMP_PCMD[1023:0];

(* Zero out TMP_HEADER*)
TMP_HEADER[sizeof(TMP_HEADER)-1: 0] := 0;

TMP_HEADER.SECINFO := SCRATCH_PCMD.SECINFO;
TMP_HEADER.RSVD := SCRATCH_PCMD.RSVD;
TMP_HEADER.LINADDR := DS:RBX.LINADDR;

(* Verify various attributes of SECS parameter *)
IF ( (TMP_HEADER.SECINFO.FLAGS.PT = PT_REG) or (TMP_HEADER.SECINFO.FLAGS.PT = PT_TCS) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_TRIM) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_FIRST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1) or
     (TMP_HEADER.SECINFO.FLAGS.PT = PT_SS_REST and CPUID.(EAX=12H, ECX=1):EAX[6] = 1))
    THEN
        IF ( DS:TMP_SECS is not 4KByte aligned)
            THEN #GP(0) FI;
        IF (DS:TMP_SECS does not resolve within an EPC)
            THEN #PF(DS:TMP_SECS) FI;
        IF (Another instruction is currently modifying the SECS)
            THEN
                #GP(0); FI;
        TMP_HEADER.EID := DS:TMP_SECS.EID;
    ELSE
        (* TMP_HEADER.SECINFO.FLAGS.PT is PT_SECS or PT_VA which do not have a parent SECS, and hence no EID binding *)
        TMP_HEADER.EID := 0;
        IF (DS:TMP_SECS ≠ 0)
            THEN #GP(0) FI;
FI;

(* Copy 4KBytes SRCPGE to secure location *)
DS:RCX[32767: 0] := DS:TMP_SRCPGE[32767: 0];
TMP_VER := DS:RDX[63:0];

(* Derive version-unique paging key *)
TMP_PG_KEY := derivePK(TMP_VER);

(* Decrypt and MAC page. AES_GCM_DEC has 2 outputs, {plain text, MAC} *)
```

```
(* Parameters for AES_GCM_DEC {Key, Counter, ..} *)
{DS:RCX, TMP_MAC} := AES256_GCM_DEC(TMP_PG_KEY, 0, TMP_HEADER, 128, DS:RCX, 4096);

IF ( (TMP_MAC ≠ DS:TMP_PCMD.MAC) )
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_MAC_COMPARE_FAIL;
        GOTO ERROR_EXIT;
FI;

(* Clear VA Slot *)
DS:RDX := 0

(* Commit EPCM changes *)
EPCM(DS:RCX).PT := TMP_HEADER.SECINFO.FLAGS.PT;
EPCM(DS:RCX).RWX := TMP_HEADER.SECINFO.FLAGS.RWX;
EPCM(DS:RCX).PENDING := TMP_HEADER.SECINFO.FLAGS.PENDING;
EPCM(DS:RCX).MODIFIED := TMP_HEADER.SECINFO.FLAGS.MODIFIED;
EPCM(DS:RCX).PR := TMP_HEADER.SECINFO.FLAGS.PR;
EPCM(DS:RCX).ENCLAVEADDRESS := TMP_HEADER.LINADDR;

IF ( (EAX = 07H) and (TMP_HEADER.SECINFO.FLAGS.PT is NOT PT_SECS or PT_VA) )
    THEN
        EPCM(DS:RCX).BLOCKED := 1;
    ELSE
        EPCM(DS:RCX).BLOCKED := 0;
FI;

EPCM(DS:RCX). VALID := 1;

RAX := 0;
RFLAGS.ZF := 0;

ERROR_EXIT:
RFLAGS.CF,PF,AF,OF,SF := 0;
```

Flags Affected

Sets ZF if unsuccessful, otherwise cleared and RAX returns error code. Clears CF, PF, AF, OF, SF.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the instruction's EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand expected to be in EPC does not resolve to an EPC page.
	If one of the EPC memory operands has incorrect page type.
	If the destination EPC page is already valid.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the instruction’s EPC resource is in use by others.
	If the instruction fails to verify MAC.
	If the version-array slot is in use.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand expected to be in EPC does not resolve to an EPC page.
	If one of the EPC memory operands has incorrect page type.
	If the destination EPC page is already valid.

EUPDATESVN—Update CR_CPUSVN

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 18H ENCLS[EUPDATESVN]	None	V/V	Bit 10	This leaf function updates the CR_CPUSVN if microcode has been updated and EPC is ready.

Description

If EPC is ready for SVN update, this leaf function updates CR_CPUSVN to the currently loaded microcode update SVN and may generate new cryptographic assets for CR_BASE_PK and CR_REPORT_MAC_KEY. The EPC is ready when no page in the EPC is valid. EREMOVE should be used to mark all pages as unused.

It is the responsibility of system software to ensure that, while executing EUPDATESVN, no other logical processor is executing ENCLS or is executing SEAMOPS instruction that updates CR_REPORT_MAC_KEY (system software can ensure that the latter case does not occur by verifying no other logical processor is executing in SEAM). Concurrency violations between EUPDATESVN and some ENCLS leaves may cause the ENCLS leaf to generate #GP(0) in ways unexpected to legacy software. System software should also prevent unnecessary software from having access to EUPDATESVN. For example, enable ENCLS exiting should be used to prevent VMs that are not part of the management system software from using EUPDATESVN.

The EUPDATESVN leaf function fails if there is insufficient entropy in the random number generator or if any logical processor is executing ENCLS or is executing SEAMOPS so as to update CR_REPORT_MAC_KEY. The ZF flag will be set to indicate an error and a code returned in RAX. If EUPDATESVN was successful but CR_CPUSVN was already up to date, the CF flag will be set and RAX will indicate that no update occurred.

On some CPUs, EUPDATESVN will update only the attestation assets (CR_CPUSVN, CR_BASE_PK and CR_REPORT_MAC_KEY) on the processor package of the logical processor executing the instruction. System software should execute EUPDATESVN on at least one logical processor on each processor package before continuing use of SGX.

If insufficient entropy causes a failure, software should repeat the instruction.

The error codes are:

Table 7-28. EUPDATESVN Return Value in RAX

Error Code (see Table 7-12)	Value	Description
No Error	0	EUPDATESVN successful.
SGX_EPC_PAGE_CONFLICT	7	An instruction concurrency rule was violated.
SGX_INSUFFICIENT_ENTROPY	29	RNG contains insufficient entropy.
SGX_EPC_NOT_READY	30	EPC is not ready for SVN update.
SGX_NO_UPDATE	31	EUPDATESVN was successful, but CR_CPUSVN was not updated because the current SVN is older than CR_CPUSVN.

Concurrency Restrictions

Table 7-29. Base Concurrency Restrictions of EUPDATESVN

Leaf	Base Concurrency Restrictions	
	Access	On Conflict
EUPDATESVN	Exclusive	SGX_EPC_PAGE_CONFLICT
An execution of SEAMOPS that updates CR_REPORT_MAC_KEY	Exclusive	SGX_EPC_PAGE_CONFLICT

Table 7-30. Additional Concurrency Restrictions of EUPDATESVN

Leaf	Additional Concurrency Restrictions	
	vs. EADD, EAUG, ECREATE, ELDB, EPA, EREMOVE, EWB	
	Access	On Conflict
EUPDATESVN	Exclusive	SGX_EPC_PAGE_CONFLICT

Operation

Temp Variables in EUPDATESVN Operational Flow

Name	Type	Size (Bytes)	Description
TMP_CPUSVN	CPUSVN	16	Temporary copy of CR_CPUSVN prior to update.
TMP_KEY	Key	64	Temporary copy of new paging key.

```
RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

IF (Other logical processor is executing an SGX instruction that accesses EPC) THEN
    RFLAGS.ZF := 1
    RAX := SGX_EPC_PAGE_CONFLICT;
    GOTO ERROR_EXIT;
FI

IF (Other logical processor is executing a SEAMOPS instruction that updates CR_REPORT_MAC_KEY) THEN
  RFLAGS.ZF := 1
  RAX := SGX_EPC_PAGE_CONFLICT;
  GOTO ERROR_EXIT;
FI

(* Verify EPC is ready *)
IF (the EPC contains any valid pages) THEN
    RFLAGS.ZF := 1;
    RAX := SGX_EPC_NOT_READY;
    GOTO ERROR_EXIT;
FI

IF (A Microcode Update has been loaded since last execution of EUPDATESVN) THEN
    (* Refresh paging key *)
    TMP_KEY = (* Generate a 512-bit cryptographically random number *)
    IF (insufficient entropy) THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INSUFFICIENT_ENTROPY;
        GOTO ERROR_EXIT;
    FI

    CR_BASE_PK := TMP_KEY[255:0];
    CR_REPORT_MAC_KEY := TMP_KEY[511:256];
FI
```

```
(* Update CR_CPUSVN to reflect current microcode update SVN *)
TMP_CPUSVN := CR_CPUSVN;


(* Determine if info status is needed *)
IF (TMP_CPUSVN = CR_CPUSVN) THEN
    RFLAGS.CF := 1;
    RAX := SGX_NO_UPDATE;
FI
ERROR_EXIT:
```

Flags Affected

ZF is set if an error occurs; otherwise, cleared.

CF is set when the instruction is completed successfully and no SVN update was needed.

PF, AF, OF, and SF are cleared.

EWB—Invalidate an EPC Page and Write out to Main Memory

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0BH ENCLS[EWB]	IR	V/V	SGX1	This leaf function invalidates an EPC page and writes it out to main memory.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX	RDX
IR	EWB (In)	Error code (Out)	Address of an PAGEINFO (In)	Address of the EPC page (In)	Address of a VA slot (In)

Description

This leaf function copies a page from the EPC to regular main memory. As part of the copying process, the page is cryptographically protected. This instruction can only be executed when current privilege level is 0.

The table below provides additional information on the memory parameter of EPA leaf function.

EWB Memory Parameter Semantics

PAGEINFO	PAGEINFO.SRCPGE	PAGEINFO.PCMD	EPCPAGE	VASLOT
Non-EPC R/W access	Non-EPC R/W access	Non-EPC R/W access	EPC R/W access	EPC R/W access

The error codes are:

Table 7-31. EWB Return Value in RAX

Error Code (see Table 7-12)	Description
No Error	EWB successful.
SGX_PAGE_NOT_BLOCKED	If page is not marked as blocked.
SGX_NOT_TRACKED	If EWB is racing with ETRACK instruction.
SGX_VA_SLOT_OCCUPIED	Version array slot contained valid entry.
SGX_CHILD_PRESENT	Child page present while attempting to page out enclave.

Concurrency Restrictions

Table 7-32. Base Concurrency Restrictions of EWB

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EWB	Source [DS:RCX]	Exclusive	#GP
	VA [DS:RDX]	Shared	#GP

Table 7-33. Additional Concurrency Restrictions of EWB

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EWB	Source [DS:RCX]	Concurrent		Concurrent		Concurrent	
	VA [DS:RDX]	Concurrent		Concurrent		Exclusive	

Operation

Temp Variables in EWB Operational Flow

Name	Type	Size (Bytes)	Description
TMP_SRCPGE	Memory page	4096	
TMP_PCMD	PCMD	128	
TMP_SECS	SECS	4096	
TMP_BPEPOCH	UINT64	8	
TMP_BPREFCOUNT	UINT64	8	
TMP_HEADER	MAC Header	128	
TMP_PCMD_ENCLAVEID	UINT64	8	
TMP_VER	UINT64	8	
TMP_PG_KEY	UINT256	32	

```
IF ( (DS:RBX is not 32Byte Aligned) or (DS:RCX is not 4KByte Aligned) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (DS:RDX is not 8Byte Aligned)
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

(* EPCPAGE and VASLOT should not resolve to the same EPC page*)
IF (DS:RCX and DS:RDX resolve to the same EPC page)
    THEN #GP(0); FI;

TMP_SRCPGE := DS:RBX.SRCPGE;
(* Note PAGEINFO.PCMD is overlaid on top of PAGEINFO.SECINFO *)
TMP_PCMD := DS:RBX.PCMD;

IF (DS:RBX.LINADDR ≠ 0) OR (DS:RBX.SECS ≠ 0)
    THEN #GP(0); FI;

IF ( (DS:TMP_PCMD is not 128Byte Aligned) or (DS:TMP_SRCPGE is not 4KByte Aligned) )
    THEN #GP(0); FI;

(* Check for concurrent Intel SGX instruction access to the page *)
IF (Other Intel SGX instruction is accessing page)
    THEN
        #GP(0); FI;

(*Check if the VA Page is being removed or changed*)
IF (VA Page is being modified)
    THEN #GP(0); FI;
```

```
(* Verify that EPCPAGE and VASLOT page are valid EPC pages and DS:RDX is VA *)
IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF ( (EPCM(DS:RDX & ~0FFFH).VALID = 0) or (EPCM(DS:RDX & ~FFFH).PT is not PT_VA) )
    THEN #PF(DS:RDX); FI;

(* Perform page-type-specific exception checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM ) or
 (EPCM(DS:RCX).PT is PT_SS_FIRST ) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        TMP_SECS = Obtain SECS through EPCM(DS:RCX)
    (* Check that EBLOCK has occurred correctly *)
    IF (EBLOCK is not correct)
        THEN #GP(0); FI;
FI;

RFLAGS.ZF,CF,PF,AF,OF,SF := 0;
RAX := 0;

(* Zero out TMP_HEADER*)
TMP_HEADER[ sizeof(TMP_HEADER) - 1 : 0] := 0;

(* Perform page-type-specific checks *)
IF ( (EPCM(DS:RCX).PT is PT_REG) or (EPCM(DS:RCX).PT is PT_TCS) or (EPCM(DS:RCX).PT is PT_TRIM )or
 (EPCM(DS:RCX).PT is PT_SS_FIRST ) or (EPCM(DS:RCX).PT is PT_SS_REST))
    THEN
        (* check to see if the page is evictable *)
        IF (EPCM(DS:RCX).BLOCKED = 0)
            THEN
                RAX := SGX_PAGE NOT_BLOCKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI;
        (* Check if tracking done correctly *)
        IF (Tracking not correct)
            THEN
                RAX := SGX_NOT_TRACKED;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
        FI;

        (* Obtain EID to establish cryptographic binding between the paged-out page and the enclave *)
        TMP_HEADER.EID := TMP_SECS.EID;

        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID := TMP_SECS.EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_SECS)
        (*check that there are no child pages inside the enclave *)
        IF (DS:RCX has an EPC page associated with it)
            THEN
                RAX := SGX_CHILD_PRESENT;
                RFLAGS.ZF := 1;
                GOTO ERROR_EXIT;
```

```
        FI:
        TMP_HEADER.EID := 0;
        (* Obtain EID as an enclave handle for software *)
        TMP_PCMD_ENCLAVEID := (DS:RCX).EID;
    ELSE IF (EPCM(DS:RCX).PT is PT_VA)
        TMP_HEADER.EID := 0; // Zero is not a special value
        (* No enclave handle for VA pages*)
        TMP_PCMD_ENCLAVEID := 0;
FI;

TMP_HEADER.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;
TMP_HEADER.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
TMP_HEADER.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
TMP_HEADER.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
TMP_HEADER.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
TMP_HEADER.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;

(* Derive paging key *)
TMP_PG_KEY = derivePK(TMP_VER);

(* Encrypt the page, DS:RCX could be encrypted in place. AES-GCM produces 2 values, {ciphertext, MAC}. *)
(* AES-GCM input parameters: key, GCM Counter, MAC_HDR, MAC_HDR_SIZE, SRC, SRC_SIZE)*)
{DS:TMP_SRCPGE, DS:TMP_PCMD.MAC} := AES256_GCM_ENC(TMP_PG_KEY, 0, TMP_HEADER, 128, DS:RCX, 4096);

(* Write the output *)
Zero out DS:TMP_PCMD.SECINFO
DS:TMP_PCMD.SECINFO.FLAGS.PT := EPCM(DS:RCX).PT;
DS:TMP_PCMD.SECINFO.FLAGS.RWX := EPCM(DS:RCX).RWX;
DS:TMP_PCMD.SECINFO.FLAGS.PENDING := EPCM(DS:RCX).PENDING;
DS:TMP_PCMD.SECINFO.FLAGS.MODIFIED := EPCM(DS:RCX).MODIFIED;
DS:TMP_PCMD.SECINFO.FLAGS.PR := EPCM(DS:RCX).PR;
DS:TMP_PCMD.RESERVED := 0;
DS:TMP_PCMD.ENCLAVEID := TMP_PCMD_ENCLAVEID;
DS:RBX.LINADDR := EPCM(DS:RCX).ENCLAVEADDRESS;

(*Check if version array slot was empty *)
IF ([DS.RDX])
    THEN
        RAX := SGX_VA_SLOT_OCCUPIED
        RFLAGS.CF := 1;
FI;

(* Write version to Version Array slot *)
[DS.RDX] := TMP_VER;

(* Free up EPCM Entry *)
EPCM.(DS:RCX).VALID := 0;
ERROR_EXIT:
```

Flags Affected

ZF is set if page is not blocked, not tracked, or a child is present. Otherwise cleared.

CF is set if VA slot is previously occupied, Otherwise cleared.

Protected Mode Exceptions

#GP(0)	If a memory operand effective address is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If the tracking resource is in use.
	If the EPC page or the version array page is invalid.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If one of the EPC memory operands has incorrect page type.

64-Bit Mode Exceptions

#GP(0)	If a memory operand is non-canonical form.
	If a memory operand is not properly aligned.
	If the EPC page and VASLOT resolve to the same EPC page.
	If another Intel SGX instruction is concurrently accessing either the target EPC, VA, or SECS pages.
	If the tracking resource is in use.
	If the EPC page or the version array page in invalid.
	If the parameters fail consistency checks.
#PF(error code)	If a page fault occurs in accessing memory operands.
	If a memory operand is not an EPC page.
	If one of the EPC memory operands has incorrect page type.

7.8 INTEL® SGX USER LEAF FUNCTION REFERENCE

Leaf functions available with the ENCLU instruction mnemonic are covered in this section. In general, each instruction leaf requires EAX to specify the leaf function index and/or additional registers specifying leaf-specific input parameters. An instruction operand encoding table provides details of the implicitly-encoded register usage and associated input/output semantics.

In many cases, an input parameter specifies an effective address associated with a memory object inside or outside the EPC, the memory addressing semantics of these memory objects are also summarized in a separate table.

EGETKEY—Retrieves a Cryptographic Key

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 01H ENCLU[EGETKEY]	IR	V/V	SGX1	This leaf function retrieves a 128-bit cryptographic key.
EAX = 0BH ENCLU[EGETKEY256]	IR	V/V	EAX[12]	This leaf function retrieves a 256-bit cryptographic key.

Instruction Operand Encoding

Op/En	EAX		RBX	RCX
IR	EGETKEY (In)	Return error code (Out)	Address to a KEYREQUEST (In)	Address of the OUTPUTDATA (In)

Description

The ENCLU[EGETKEY] and ENCLU[EGETKEY256] instructions return a secret key from the processor specific key hierarchy. EGETKEY returns a 128-bit key, and EGETKEY256 returns a 256-bit key. The register RBX contains the effective address of a KEYREQUEST structure, which the instruction interprets to determine the key being requested. The Requesting Keys section below provides a description of the keys that can be requested. The RCX register contains the effective address where the key will be returned. Both the addresses in RBX & RCX should be locations inside the enclave.

EGETKEY/EGETKEY256 derives keys using a processor unique value to create a specific key based on a number of possible inputs. This instruction leaf can only be executed inside an enclave.

EEGETKEY Memory Parameter Semantics

KEYREQUEST	OUTPUTDATA
Enclave read access	Enclave write access

After validating the operands, the instruction determines which key is to be produced and performs the following actions:

- The instruction assembles the derivation data for the key based on the Table 7-34.
- Computes derived key using the derivation data and package specific value.
- Outputs the calculated key to the address in RCX.

The instruction fails with #GP(0) if the operands are not properly aligned. Successful completion of the instruction will clear RFLAGS.{ZF, CF, AF, OF, SF, PF}. The instruction returns an error code if the user tries to request a key based on an invalid CPUSVN or ISVSVN (when the user request is accepted, see the table below), requests a key for which it has not been granted the attribute to request, or requests a key that is not supported by the hardware. These checks may be performed in any order. Thus, an indication by error number of one cause (for example, invalid attribute) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same Enclave. The correctness of software should not rely on the order resulting from the checks documented in this section. In such cases the ZF flag is set and the corresponding error bit (SGX_INVALID_SVN, SGX_INVALID_ATTRIBUTE, SGX_INVALID_KEYNAME) is set in RAX and the data at the address specified by RCX is unmodified.

Requesting Keys

The KEYREQUEST structure (see "KEY REQUEST KeyNames" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3) identifies the key to be provided. The Keyrequest.KeyName field identifies which type of key is requested. Note that Report Key and Provisioning Key are accessible by EGETKEY only.

Deriving Keys

Key derivation is based on a combination of the enclave specific values (see Table 7-34) and a processor key. Depending on the key being requested a field may either be included by definition or the value may be included from the KeyRequest. A "yes" in Table 7-34 indicates the value for the field is included from its default location, identified in the source row, and a "request" indicates the values for the field is included from its corresponding KeyRequest field.

Table 7-34. Key Derivation

	Key Name	Attributes	Owner Epoch	CPU SVN	ISV SVN	ISV PRODID	ISVEXT PRODID	ISVFAMILYID	MRENCLAVE	MRSIGNER	CONFIG ID	CONFIGSVN	RAND
Source	Key Dependent Constant	Y := SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES; R := AttribMask & SECS.ATTRIBUTES and SECS.MISCSELECT and SECS.CET_ATTRIBUTES;	CR_SGXOWNEREPOCH	Y := CPUSVN Register; R := Req.CPUSVN;	R := Req.ISVSVN;	SECS.ISVID	SECS.ISVEXTPRODID	SECS.ISVFAMILYID	SECS.MRENCLAVE	SECS.MRSIGNER	SECS.CONFIGID	SECS.CONFIGSVN	Req.KEYID
EINITTOKEN	Yes	Request	Yes	Request	Request	Yes	No	No	No	Yes	No	No	Request
Report	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	Yes	Yes	Request
Seal	Yes	Request	Yes	Request	Request	Request	Request	Request	Request	Request	Request	Request	Request
Provisioning	Yes	Request	No	Request	Request	Yes	No	No	No	Yes	No	No	Yes
Provisioning Seal	Yes	Request	No	Request	Request	Request	Request	Request	No	Yes	Request	Request	Yes

Keys that permit the specification of a CPU or ISV's code's, or enclave configuration's SVNs have additional requirements. The caller may not request a key for an SVN beyond the current CPU, ISV or enclave configuration's SVN, respectively.

Several keys are access controlled. Access to the Provisioning Key and Provisioning Seal key requires the enclave's ATTRIBUTES.PROVISIONKEY be set. The EINITTOKEN Key requires ATTRIBUTES.EINITTOKEN_KEY be set and SECS.MRSIGNER equal IA32_SGXLEPUBKEYHASH.

Some keys are derived based on a hardcode SHA384 hash of a PKCS padding constant:

HARDCODED_PKCS1_5_PADDING[15:0] := 0100H;

HARDCODED_PKCS1_5_PADDING[2655:16] := SignExtend330Byte(-1); // 330 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[2815:2656] := 2004000501020403650148866009060D30313000H;

HARDCODED_PKCS1_5_PADDING[2831:2816] := 0100H;

HARDCODED_PKCS1_5_PADDING[5471:2832] := SignExtend314Byte(-1); // 314 bytes of 0FFH

HARDCODED_PKCS1_5_PADDING[5631:5472] := 3004000501020403650148866009060D30413000H;

HARDCODED_PKCS1_5_PADDING[5759:5632] := 0

HARDCODED_PKCS1_5_PADDING_HASH = SHA384(HARDCODED_PKCS1_5_PADDING);

The error codes are:

Table 7-35. EGETKEY Return Value in RAX

Error Code (see Table 7-12)	Value	Description
No Error	0	EGETKEY successful.
SGX_INVALID_ATTRIBUTE		The KEYREQUEST contains a KEYNAME for which the enclave is not authorized.
SGX_INVALID_CPUSVN		If KEYREQUEST.CPUSVN is an unsupported platforms CPUSVN value.
SGX_INVALID_ISVSVN		If KEYREQUEST software SVN (ISVSVN or CONFIGSVN) is greater than the enclave's corresponding SVN.
SGX_INVALID_KEYNAME		If KEYREQUEST.KEYNAME is an unsupported value.

Concurrency Restrictions

Table 7-36. Base Concurrency Restrictions of EGETKEY

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent	
EGETKEY256	KEYREQUEST [DS:RBX]	Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent	

Table 7-37. Additional Concurrency Restrictions of EGETKEY

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EGETKEY	KEYREQUEST [DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	
EGETKEY256	KEYREQUEST [DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EGETKEY Operational Flow

Name	Type	Size (Bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYCONTEXT			Temporary space for key derivation context.
TMP_INCSEALFUSES	Bool		Temporary space indicating if Seal Fuses are included in key derivation.
TMP_INCOWNEREPOCH	Bool		Temporary space indicating if OwnerEpoch is included in key derivation.
TMP_KEYCPUSVN		16 bytes	
TMP_KEYDEPENDENCIES			Temp space for key derivation.
TMP_ATTRIBUTES		16 bytes	Temp Space for the calculation of the sealable Attributes.
TMP_ISVEXTPRODID		16 bytes	Temp Space for ISVEXTPRODID.
TMP_ISVPRODID		2 bytes	Temp Space for ISVPRODID.
TMP_ISVFAMILYID		16 bytes	Temp Space for ISVFAMILYID.
TMP_CONFIGID		64 bytes	Temp Space for CONFIGID.
TMP_CONFIGSVN		2 bytes	Temp Space for CONFIGSVN.
TMP_OUTPUTKEY		32 bytes	Temp Space for the calculation of the key.
		48 bytes	Temporary space for building MRENCLAVE.
		48 bytes	Temporary space for building MRSIGNER.

```
(* Make sure KEYREQUEST is properly aligned and inside the current enclave *)
IF ( (DS:RBX is not 512Byte aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

(* Make sure DS:RBX is an EPC address and the EPC page is valid *)
IF ( (DS:RBX does not resolve to an EPC address) or (EPCM(DS:RBX).VALID = 0) )
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Make sure OUTPUTDATA is properly aligned *)
IF ((RAX = 0x01) AND (DS:RCX is not 16 byte aligned)) OR
    (RAX = 0x0B) AND (DS:RCX is not 32 byte aligned)))
    THEN #GP(0); FI;

(* Make sure OUTPUTDATA is inside the current enclave *)
IF (DS:RCX is not within CR_ELRANGE)
    THEN #GP(0); FI;
```

```
(* Make sure DS:RCX is an EPC address and the EPC page is valid *)
IF ( (DS:RCX does not resolve to an EPC address) or (EPCM(DS:RCX).VALID = 0) )
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
    THEN #PF(DS:RCX);
FI;

(* Verify RESERVED spaces in KEYREQUEST are valid *)
IF ( (DS:RBX).RESERVED ≠ 0) or (DS:RBX.KEYPOLICY.RESERVED ≠ 0) )
    THEN #GP(0); FI;

TMP_CURRENTSECS := CR_ACTIVE_SECS;

(* Verify that CONFIGSVN & New Policy bits are not used if KSS is not enabled *)
IF ((TMP_CURRENTSECS.ATTRIBUTES.KSS == 0) AND ((DS:RBX.KEYPOLICY & 0x003C ≠ 0) OR (DS:RBX.CONFIGSVN > 0)))
    THEN #GP(0); FI;
(* Determine which enclave attributes that must be included in the key. Attributes that must always be include INIT & DEBUG *)
REQUIRED_SEALING_MASK[127:0] := 00000000 00000000 00000000 00000003H;
TMP_ATTRIBUTES := (DS:RBX.ATTRIBUTEMASK | REQUIRED_SEALING_MASK) & TMP_CURRENTSECS.ATTRIBUTES;

(* Compute MISCSELECT fields to be included *)
TMP_MISCSELECT := DS:RBX.MISCMASK & TMP_CURRENTSECS.MISCSELECT

(* Compute CET_ATTRIBUTES fields to be included *)
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES_ MASK & TMP_CURRENTSECS.CET_ATTRIBUTES; FI;

(* Select correct MRENCLAVE/MRSIGNER for the enclave type *)
(* EGETKEY uses legacy 256 bit values regardless of ATTRIBUTES.SH384. *)
IF ((RAX = 0x0B) AND
    (TMP_CURRENTSECS.ATTRIBUTES.SHA384 = 1)) {
    THEN
        TMP_MRENCLAVEBUF[3071:0] := TMP_CURRENTSECS.MRENCLAVE384;
        TMP_MRSIGNERBUF[3071:0] := TMP_CURRENTSECS.MRSIGNER384;
ELSE
    THEN
        TMP_MRENCLAVEBUF[2047:0] := TMP_CURRENTSECS.MRENCLAVE;
        TMP_MRENCLAVEBUF[3071:2048] := 0;
        TMP_MRSIGNERBUF[2047:0] := TMP_CURRENTSECS.MRSIGNER;
        TMP_MRSIGNERBUF[3071:2048]  := 0;
FI;

TMP_KEYDEPENDENCIES := 0;

CASE (DS:RBX.KEYNAME)
    SEAL_KEY:
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
```

```
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CONFIGSVN > TMP_CURRENTSECS.CONFIGSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;

        (*Include enclave identity?*)
        TMP_MRENCLAVE := 0;
        IF (DS:RBX.KEYPOLICY.MRENCLAVE = 1)
            THEN TMP_MRENCLAVE := TMP_MRENCLAVEBUF;
        FI;
        (*Include enclave author?*)
        TMP_MRSIGNER := 0;
        IF (DS:RBX.KEYPOLICY.MRSIGNER = 1)
            THEN TMP_MRSIGNER := TMP_MRSIGNERBUF;
        FI;
(* Include enclave product family ID? *)
    TMP_ISVFAMILYID := 0;
    IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
        THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
        FI;

    (* Include enclave product ID? *)
    TMP_ISVPRODID := 0;
    IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
        TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
        FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
        TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
        FI;

    (* Include enclave extended product ID? *)
    TMP_ISVEXTPRODID := 0;
    IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1 )
        TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
    FI;
```

```
        //Determine values key is based on
        TMP_KEYCONTEXT.KEYNAME := SEAL_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_INCOWNEREPOCH := TRUE;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE := TMP_MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_INCSEALKEYFUSES := TRUE;
        TMP_KEYCPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
        TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
        TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
        IF CPUID.(EAX=12H, ECX=1):EAX[6] = 1
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := DS:RBX.CET_ATTRIBUTES _MASK;
        FI;
        BREAK;

REPORT_KEY:
    (* Report Keys are only available from EGETKEY. *)
        IF (RAX ≠ 0x01)
        THEN
            RFLAGS.ZF := 1;
            RAX := SGX_INVALID_KEYNAME;
            Goto EXIT;
        FI;

        (* Determine values key is based on *)
        TMP_KEYCONTEXT.KEYNAME := REPORT_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVSVN := 0;
        TMP_INCOWNEREPOCH := TRUE;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE[2047:0] := TMP_CURRENTSECS.MRENCLAVE;
        TMP_KEYDEPENDENCIES.MRENCLAVE[3071:2048] := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := 0;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_INCSEALKEYFUSES := TRUE;
        TMP_KEYCPUSVN := CR_CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := HARDCODED_PKCS1_5_PADDING_HASH;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := 0;
```

```
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := TMP_CURRENTSECS.CONFIGID;
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES_MASK := 0;
        FI;
        BREAK;

    EINITTOKEN_KEY:
        (* Check ENCLAVE has EINITTOKEN Key capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.EINITTOKEN_KEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYCONTEXT.KEYNAME := EINITTOKEN_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_INCOWNEREPOCH := TRUE;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
        TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID := DS:RBX.KEYID;
        TMP_INCSEALKEYFUSES := TRUE;
        TMP_KEYCPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := 0;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := 0;
        TMP_KEYDEPENDENCIES.CONFIGSVN := 0;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
```

```
        FI;
        BREAK;

    PROVISION_KEY:
    (* Report Keys are only available from EGETKEY. *)
    IF (RAX ≠ 0x01)
    THEN
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_KEYNAME;
        Goto EXIT;
    FI;

    (* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
        (* Determine values key is based on *)
        TMP_KEYCONTEXT.KEYNAME := PROVISION_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_INCOWNEREPOCH := FALSE;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_MRSIGNERBUF;
        TMP_KEYDEPENDENCIES.KEYID := 0;
        TMP_INCSEALKEYFUSES := FALSE;
        TMP_KEYCPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
        TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
        TMP_KEYDEPENDENCIES.CONFIGID := 0;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
```

```
        FI;
        BREAK;

    PROVISION_SEAL_KEY:
        (* Check ENCLAVE has PROVISIONING capability *)
        IF (TMP_CURRENTSECS.ATTRIBUTES.PROVISIONKEY = 0)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ATTRIBUTE;
                GOTO EXIT;
        FI;
        IF (DS:RBX.CPUSVN is beyond current CPU configuration)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_CPUSVN;
                GOTO EXIT;
        FI;
        IF (DS:RBX.ISVSVN > TMP_CURRENTSECS.ISVSVN)
            THEN
                RFLAGS.ZF := 1;
                RAX := SGX_INVALID_ISVSVN;
                GOTO EXIT;
        FI;
(* Include enclave product family ID? *)
    TMP_ISVFAMILYID := 0;
    IF (DS:RBX.KEYPOLICY.ISVFAMILYID = 1)
    THEN TMP_ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
    FI;

    (* Include enclave product ID? *)
    TMP_ISVPRODID := 0;
    IF (DS:RBX.KEYPOLICY.NOISVPRODID = 0)
        TMP_ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
    FI;

    (* Include enclave Config ID? *)
    TMP_CONFIGID := 0;
    TMP_CONFIGSVN := 0;
    IF (DS:RBX.KEYPOLICY.CONFIGID = 1)
        TMP_CONFIGID := TMP_CURRENTSECS.CONFIGID;
        TMP_CONFIGSVN := DS:RBX.CONFIGSVN;
    FI;

    (* Include enclave extended product ID? *)
    TMP_ISVEXTPRODID := 0;
    IF (DS:RBX.KEYPOLICY.ISVEXTPRODID = 1)
        TMP_ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
    FI;

        (* Determine values key is based on *)
        TMP_KEYCONTEXT.KEYNAME := PROVISION_SEAL_KEY;
        TMP_KEYDEPENDENCIES.ISVFAMILYID := TMP_ISVFAMILYID;
        TMP_KEYDEPENDENCIES.ISVEXTPRODID := TMP_ISVEXTPRODID;
        TMP_KEYDEPENDENCIES.ISVPRODID := TMP_ISVPRODID;
```

```
        TMP_KEYDEPENDENCIES.ISVSVN := DS:RBX.ISVSVN;
        TMP_INCOWNEREPOCH := FALSE;
        TMP_KEYDEPENDENCIES.ATTRIBUTES := TMP_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := DS:RBX.ATTRIBUTEMASK;
        TMP_KEYDEPENDENCIES.MRENCLAVE := 0;
        TMP_KEYDEPENDENCIES.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
        TMP_KEYDEPENDENCIES.KEYID := 0;
        TMP_INCSEALKEYFUSES := TRUE;
        TMP_KEYCPUSVN := DS:RBX.CPUSVN;
        TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
        TMP_KEYDEPENDENCIES.MISCSELECT := TMP_MISCSELECT;
        TMP_KEYDEPENDENCIES.MISCMASK := ~DS:RBX.MISCMASK;
        TMP_KEYDEPENDENCIES.KEYPOLICY := DS:RBX.KEYPOLICY;
        TMP_KEYDEPENDENCIES.CONFIGID := TMP_CONFIGID;
        TMP_KEYDEPENDENCIES.CONFIGSVN := TMP_CONFIGSVN;
        IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
            THEN
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := TMP_CET_ATTRIBUTES;
                TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
        FI;
        BREAK;
    DEFAULT:
        (* The value of KEYNAME is invalid *)
        RFLAGS.ZF := 1;
        RAX := SGX_INVALID_KEYNAME;
        GOTO EXIT:
ESAC;

(* Calculate the final derived key and output to the address in RCX *)
IF (RAX == 0x01)
THEN
    TMP_OUTPUTKEY = derivekey(TMP_KEYCPUSVN,
                              TMP_INCSEALFUSES,
                              TMP_INCOWNEREPOCH,
                              SHA384(TMP_KEYDEPENDENCIES),
                              TMP_KEY_CONTEXT,
                              128);
    DS:RCX[127:0] = TMP_OUTPUTKEY;
ELSE
THEN
    TMP_OUTPUTKEY = derivekey(TMP_KEYCPUSVN,
                              TMP_INCSEALFUSES,
                              TMP_INCOWNEREPOCH,
                              SHA384(TMP_KEYDEPENDENCIES),
                              TMP_KEY_CONTEXT,
                              256);
    DS:RCX[255:0] = TMP_OUTPUTKEY;
}

RAX := 0;
RFLAGS.ZF := 0;

EXIT:
RFLAGS.CF := 0;
```

```
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF is cleared if successful, otherwise ZF is set. CF, PF, AF, OF, SF are cleared.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is outside the DS segment limit.
	If KEYREQUEST format is invalid.
#PF(error code)	If a page fault occurs in accessing memory.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the current enclave.
	If an effective address is not properly aligned.
	If an effective address is not canonical.
	If KEYREQUEST format is invalid.
#PF(error code)	If a page fault occurs in accessing memory operands.

EREPORT—Create a Cryptographic Report of the Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 00H ENCLU[EREPORT]	IR	V/V	SGX1	This leaf function creates a cryptographic report of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EREPORT (In)	Address of TARGETINFO (In)	Address of REPORTDATA (In)	Address where the REPORT is written to in an OUTPUTDATA (In)

Description

This leaf function creates a cryptographic REPORT that describes the contents of the enclave. This instruction leaf can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine that the enclave is running on the same platform.

RBX contains the effective address of the MRENCLAVE value of the enclave that will authenticate the REPORT output, using the REPORT key delivered by EGETKEY command for that enclave. RCX contains the effective address of a 64-byte REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RDX contains the address where the REPORT will be output by the instruction.

EREPORT Memory Parameter Semantics

TARGETINFO	REPORTDATA	OUTPUTDATA
Read access by Enclave	Read access by Enclave	Read/Write access by Enclave

This instruction leaf perform the following:

1. Validate the 3 operands (RBX, RCX, RDX) are inside the enclave.
2. Compute a report key for the target enclave, as indicated by the value located in RBX(TARGETINFO).
3. Assemble the enclave SECS data to complete the REPORT structure (including the data provided using the RCX (REPORTDATA) operand).
4. Computes a cryptographic hash over REPORT structure.
5. Add the computed hash to the REPORT structure.
6. Output the completed REPORT structure to the address in RDX (OUTPUTDATA).

The instruction fails if the operands are not properly aligned.

CR_REPORT_KEYID, used to provide key wearout protection, is populated with a statistically unique value on boot of the platform by a trusted entity within the SGX TCB.

The instruction faults if any of the following:

EREPORT Faulting Conditions

An effective address not properly aligned.	An memory address does not resolve in an EPC page.
If accessing an invalid EPC page.	If the EPC page is blocked.
May page fault.	

Concurrency Restrictions

Table 7-38. Base Concurrency Restrictions of EREPORT

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EREPORT	TARGETINFO [DS:RBX]	Concurrent	
	REPORTDATA [DS:RCX]	Concurrent	
	OUTPUTDATA [DS:RDX]	Concurrent	

Table 7-39. Additional Concurrency Restrictions of EREPORT

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREPORT	TARGETINFO [DS:RBX]	Concurrent		Concurrent		Concurrent	
	REPORTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RDX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREPORT Operational Flow

Name	Type	Size (bits)	Description
TMP_ATTRIBUTES		32	Physical address of SECS of the enclave to which source operand belongs.
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_KEYDEPENDENCIES			Temporary space for key derivation.
TMP_KEYCONTEXT			Temporary space for key derivation context.
TMP_REPORTKEY		128	REPORTKEY generated by the instruction.
TMP_REPORT		3712	

```
TMP_MODE64 := ((IA32_EFER.LMA = 1) && (CS.L = 1));

(* Address verification for TARGETINFO (RBX) *)
IF ( (DS:RBX is not 512Byte Aligned) or (DS:RBX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
```

```
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Verify RESERVED spaces in TARGETINFO are valid *)
IF (DS:RBX.RESERVED != 0)
    THEN #GP(0); FI;

(* Address verification for REPORTDATA (RCX) *)
IF ( (DS:RCX is not 128Byte Aligned) or (DS:RCX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).R = 0) )
    THEN #PF(DS:RCX);
FI;

(* Address verification for OUTPUTDATA (RDX) *)
IF ( (DS:RDX is not 512Byte Aligned) or (DS:RDX is not within CR_ELRANGE) )
    THEN #GP(0); FI;

IF (DS:RDX does not resolve within an EPC)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).VALID = 0)
    THEN #PF(DS:RDX); FI;

IF (EPCM(DS:RDX).BLOCKED = 1)
    THEN #PF(DS:RDX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RDX).PT ≠ PT_REG) or (EPCM(DS:RDX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RDX).ENCLAVEADDRESS ≠ (DS:RDX & ~0FFFH) ) or (EPCM(DS:RDX).W = 0) )
    THEN #PF(DS:RDX);
FI;

(* REPORT MAC needs to be computed over data which cannot be modified *)
TMP_REPORT.CPUSVN := CR_CPUSVN;
TMP_REPORT.ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
TMP_REPORT.ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
TMP_REPORT.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
TMP_REPORT.ISVSVN := TMP_CURRENTSECS.ISVSVN;
TMP_REPORT.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
```

```
TMP_REPORT.REPORTDATA := DS:RCX[511:0];
TMP_REPORT.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
TMP_REPORT.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT.MRRESERVED := 0;
TMP_REPORT.KEYID[255:0] := CR_REPORT_KEYID;
TMP_REPORT.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
TMP_REPORT.CONFIGID := TMP_CURRENTSECS.CONFIGID;
TMP_REPORT.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN TMP_REPORT.CET_ATTRIBUTES := TMP_CURRENTSECS.CET_ATTRIBUTES; FI;

(* Derive the report key *)
TMP_KEYCONTEXT.KEYNAME := REPORT_KEY;
TMP_KEYDEPENDENCIES.ISVFAMILYID := 0;
TMP_KEYDEPENDENCIES.ISVEXTPRODID := 0;
TMP_KEYDEPENDENCIES.ISVPRODID := 0;
TMP_KEYDEPENDENCIES.ISVSVN := 0;
TMP_INCOWNEREPOCH := TRUE;
TMP_KEYDEPENDENCIES.ATTRIBUTES := DS:RBX.ATTRIBUTES;
TMP_KEYDEPENDENCIES.ATTRIBUTESMASK := 0;
TMP_KEYDEPENDENCIES.MRENCLAVE[2047:0] := DS:RBX.MEASUREMENT;
TMP_KEYDEPENDENCIES.MRENCLAVE[3071:2048]: = 0;
TMP_KEYDEPENDENCIES.MRSIGNER := 0;
TMP_KEYDEPENDENCIES.KEYID := TMP_REPORT.KEYID;
TMP_INCSEALKEYFUSES := TRUE;
TMP_KEYCPUSVN := CR_CPUSVN;
TMP_KEYDEPENDENCIES.PADDING := TMP_CURRENTSECS.PADDING;
TMP_KEYDEPENDENCIES.MISCSELECT := DS:RBX.MISCSELECT;
TMP_KEYDEPENDENCIES.MISCMASK := 0;
TMP_KEYDEPENDENCIES.KEYPOLICY := 0;
TMP_KEYDEPENDENCIES.CONFIGID := DS:RBX.CONFIGID;
TMP_KEYDEPENDENCIES.CONFIGSVN := DS:RBX.CONFIGSVN;
IF (CPUID.(EAX=12H, ECX=1):EAX[6] = 1)
    THEN
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES := DS:RBX.CET_ATTRIBUTES;
        TMP_KEYDEPENDENCIES.CET_ATTRIBUTES _MASK := 0;
FI;

(* Calculate the derived key*)
TMP_REPORTKEY = derivekey(CR_CPUSVN,
                          TRUE, // Include Seal Fuses
                          TRUE, // Include Owner Epoch
                          SHA384(TMP_KEYDEPENDENCIES),
                          TMP_KEYCONTEXT,
                          128);

(* Call cryptographic CMAC function, CMAC data are not including MAC&KEYID *)
TMP_REPORT.MAC := cmac(TMP_REPORTKEY, TMP_REPORT[3071:0] );
DS:RDX[3455: 0] := TMP_REPORT;
```

Flags Affected

None

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If the address in RCS is outside the DS segment limit.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If RCX is non-canonical form.
	If a memory operand is not properly aligned.
	If a memory operand is not in the current enclave.
#PF(error code)	If a page fault occurs in accessing memory operands.

EREPORT2—Create a Cryptographic Report of the Enclave

Opcode/ Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
EAX = 0AH ENCLU[EREPORT2]	IR	V/V	EAX[12]	This leaf function creates a cryptographic REPORT2 of the enclave.

Instruction Operand Encoding

Op/En	EAX	RBX	RCX	RDX
IR	EREPORT2 (In)	Address of REPORTDATA (In)	Address where the REPORT2 is written to in an OUTPUTDATA (In)	REPORT2 Subtype

Description

This instruction creates a cryptographic REPORT2 structure that describes the contents of the enclave. The instruction can only be executed when inside the enclave. The cryptographic report can be used by other enclaves to determine whether the enclave is running on the same platform.

RBX contains the effective address of a REPORTDATA structure, which allows the caller of the instruction to associate data with the enclave from which the instruction is called. RCX contains the address where the REPORT2 structure will be output by the instruction.

Compared to a REPORT created by ENCLU[EREPORT], REPORT2 structures can be verified by any enclave using ENCLU[EVERIFYREPORT2] instruction, whereas REPORT structures can only be verified by the enclave whose TARGETINFO was provided at EREPORT invocation. REPORT2 can reflect SHA384 measurements, whereas REPORT only supports legacy SHA256 measurement.

EREPORT2 Memory Parameter Semantics

REPORTDATA	OUTPUTDATA
Read access by Enclave	Read/Write access by Enclave

This instruction leaf performs the following:

1. Validate the operands (RBX, RCX) are inside the enclave.
2. Assemble the enclave SECS values to complete the REPORT2 structure.
3. Compute hash SECS values.
4. Assemble REPORTMACSTRUCT fields (including the data provided using the RBX [REPORTDATA] operand).
5. Computes a 256-bit HMAC over REPORTMACSTRUCT structure using CR_REPORT_MAC_KEY.
6. Output the completed REPORT2 structure to the address in RCX (OUTPUTDATA).

The instruction fails if the operands are not properly aligned.

The instruction faults if any of the following occur:

EREPORT2 Faulting Conditions

An effective address not properly aligned.	A memory address does not resolve in an EPC page.
If accessing an invalid EPC page.	If the EPC page is blocked.
May page fault.	

Concurrency Restrictions

Table 7-40. Base Concurrency Restrictions of EREPORT2

Leaf	Parameter	Base Concurrency Restrictions	
		Access	On Conflict
EREPORT2	REPORTDATA [DS:RBX]	Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent	

Table 7-41. Additional Concurrency Restrictions of EREPORT2

Leaf	Parameter	Additional Concurrency Restrictions					
		vs. EACCEPT, EACCEPTCOPY, EMODPE, EMODPR, EMODT		vs. EADD, EEXTEND, EINIT		vs. ETRACK	
		Access	On Conflict	Access	On Conflict	Access	On Conflict
EREPORT2	REPORTDATA [DS:RBX]	Concurrent		Concurrent		Concurrent	
	OUTPUTDATA [DS:RCX]	Concurrent		Concurrent		Concurrent	

Operation

Temp Variables in EREPORT2 Operational Flow

Name	Type	Size (bits)	Description
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.
TMP_REPORTKEY		32 bytes	256-bit REPORT2 key generated by instruction.
TMP_REPORT2	REPORT2	768 bytes	
TMP_DERIVATIONSTRING		192 bytes	Derivation string for REPORT2 key.
TMP_CURRENTSECS			Address of the SECS for the currently executing enclave.

EREPORT2 Return Value in RAX

Error	Value	Description
No error	0	EREPORT2 successful.
SGX_INVALID_REPORT_SUBTYPE		Invalid REPORT subtype was specified.

```
(* Address verification for REPORTDATA (RBX) *)
IF((DS:RBX is not 64 Byte aligned) or (DS:RBX is not within CR_ELRANGE))
    THEN #GP(0); FI;

IF (DS:RBX does not resolve within an EPC)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).VALID = 0)
    THEN #PF(DS:RBX); FI;

IF (EPCM(DS:RBX).BLOCKED = 1)
```

```
    THEN #PF(DS:RBX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RBX).PT ≠ PT_REG) or (EPCM(DS:RBX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RBX).PENDING = 1) or
    (EPCM(DS:RBX).MODIFIED = 1) or (EPCM(DS:RBX).ENCLAVEADDRESS ≠ (DS:RBX & ~0FFFH) ) or (EPCM(DS:RBX).R = 0) )
    THEN #PF(DS:RBX);
FI;

(* Address verification for OUTPUTDATA (RCX) *)
IF((DS:RCX is not 1024 Byte aligned) or (DS:RCX is not within CR_ELRANGE))
    THEN #GP(0); FI;

IF (DS:RCX does not resolve within an EPC)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).VALID = 0)
    THEN #PF(DS:RCX); FI;

IF (EPCM(DS:RCX).BLOCKED = 1)
    THEN #PF(DS:RCX); FI;

(* Check page parameters for correctness *)
IF ( (EPCM(DS:RCX).PT ≠ PT_REG) or (EPCM(DS:RCX).ENCLAVESECS ≠ CR_ACTIVE_SECS) or (EPCM(DS:RCX).PENDING = 1) or
    (EPCM(DS:RCX).MODIFIED = 1) or (EPCM(DS:RCX).ENCLAVEADDRESS ≠ (DS:RCX & ~0FFFH) ) or (EPCM(DS:RCX).W = 0) )
    THEN #PF(DS:RCX);
FI;

(* Verify subtype is valid *)
IF(RDX != 0)
    THEN
   RAX := SGX_INVALID_REPORT_SUBTYPE;
   RFLAGS.ZF := 1;
   GOTO EXIT;
FI;

TMP_CURRENTSECS := CR_ACTIVE_SECS;

(* REPORT MAC needs to be computed over data which cannot be modified *)
TMP_REPORT2.TEEINFO.MISCSELECT := TMP_CURRENTSECS.MISCSELECT;
TMP_REPORT2.TEEINFO.ISVPRODID := TMP_CURRENTSECS.ISVPRODID;
TMP_REPORT2.TEEINFO.ISVSVN := TMP_CURRENTSECS.ISVSVN;
TMP_REPORT2.TEEINFO.CONFIGSVN := TMP_CURRENTSECS.CONFIGSVN;
TMP_REPORT2.TEEINFO.ISVEXTPRODID := TMP_CURRENTSECS.ISVEXTPRODID;
TMP_REPORT2.TEEINFO.ATTRIBUTES := TMP_CURRENTSECS.ATTRIBUTES;
TMP_REPORT2.TEEINFO.MRENCLAVE := TMP_CURRENTSECS.MRENCLAVE;
TMP_REPORT2.TEEINFO.MRSIGNER := TMP_CURRENTSECS.MRSIGNER;
TMP_REPORT2.TEEINFO.CONFIGID := TMP_CURRENTSECS.CONFIGID;
TMP_REPORT2.TEEINFO.ISVFAMILYID := TMP_CURRENTSECS.ISVFAMILYID;
IF(TMP_CURRENTSECS.ATTRIBUTES.SHA384 == 1)
    THEN
        TMP_REPORT2.TEEINFO.MRENCLAVE384:= TMP_CURRENTSECS.MRENCLAVE384;
        TMP_REPORT2.TEEINFO.MRSIGNER384 := TMP_CURRENTSECS.MRSIGNER384;
FI;
```

```
TMP_REPORT2.REPORTMACSTRUCT.TYPE := 0x00;
TMP_REPORT2.REPORTMACSTRUCT.SUBTYPE := 0x00;
TMP_REPORT2.REPORTMACSTRUCT.VERSION := 0x00;
TMP_REPORT2.REPORTMACSTRUCT.CPUSVN := CR_CPUSVN
TMP_REPORT2.REPORTMACSTRUCT.TEEINFOHASH := SHA384(TMP_REPORT2.TEEINFO);
TMP_REPORT2.REPORTMACSTRUCT.REPORTDATA := DS:RBX[511:0];

(* compute HMAC for REPORTMACSTRUCT *)
TMP_REPORT2.REPORTMACSTRUCT.MAC := HMAC-SHA256(CR_REPORT_MAC_KEY, TMP_REPORT2.REPORTMACSTRUCT[1535:0]);

DS:RCX[6143:0] := TMP_REPORT2;
RAX := 0;
RFLAGS.ZF := 0;

EXIT:
RFLAGS.CF := 0;
RFLAGS.PF := 0;
RFLAGS.AF := 0;
RFLAGS.OF := 0;
RFLAGS.SF := 0;
```

Flags Affected

ZF	Cleared if instruction completed successfully.
	Set if error occurred. RAX is set to the error code.
CF, PF, AF, OF, SF	Cleared.

Protected Mode Exceptions

#GP(0)	If executed outside an enclave.
	If a memory operand effective address is outside the DS segment limit.
	If DS segment is unusable.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands, including EPCM-induced faults.

64-Bit Mode Exceptions

#GP(0)	If executed outside an enclave.
	If memory address is in a non-canonical form.
	If a memory operand is not properly aligned.
#PF(error code)	If a page fault occurs in accessing memory operands, including EPCM-induced faults.

CHAPTER 8
RDPMC USER DISABLE

RDPMC (Read Performance-Monitoring Counter) is an instruction that allows reading a performance-monitoring counter. Executing RDPMC when CPL > 0 is allowed only if the PCE flag (performance-monitoring counter enable) is set in CR4. (Execution of RDPMC causes a general-protection exception if CPL > 0 and CR4.PCE = 0.) The new **RDPMC User Disable** feature adds per-counter control of RDPMC when CPL > 0. This feature applies only when CR4.PCE = 1.

8.1 PER-COUNTER RDPMC CONTROL

Configuration of general-purpose performance-monitoring counter x (where x identifies the specific general-purpose counter) is controlled by the IA32_PERFEVTSELx MSR. RDPMC User Disable defines bit 37 in each such MSR as RDPMC_USR_DISABLE. When the RDPMC_USR_DISABLE bit is set in one of these MSRs, reading the corresponding counter using RDPMC when CPL > 0 returns 0 instead of the counter value.

Configuration of the fixed-function performance-monitoring counters is controlled by the IA32_FIXED_CTR_CTRL MSR. RDPMC User Disable defines bit 33+4*n in that MSR as RDPMC_USR_DISABLEn (where n identifies a specific fixed-function counter). For example, fixed-function counter 0 is controlled by bit 33, RDPMC_USR_DISABLE0. When the RDPMC_USR_DISABLEn bit is set in the IA32_FIXED_CTR_CTRL MSR, reading fixed-function performance-monitoring counter n using RDPMC when CPL > 0 returns 0 instead of the counter value.

If IA32_PERF_CAPABILITIES.PERF_METRICS_AVAILABLE[bit 15] = 1, execution of RDPMC with ECX[31:16] = 2000H reads performance metrics. This functionality uses fixed-function counter 3. If the RDPMC_USR_DISABLE3 bit is set in the IA32_FIXED_CTR_CTRL MSR, an attempt to read performance metrics with RDPMC returns 0 when CPL > 0.

8.2 ENUMERATION

The RDPMC User Disable feature is enumerated through CPUID.23H.00H:EBX[2].

CHAPTER 9
PERFORMANCE MONITORING UNIT ENHANCEMENTS

This chapter provides information for Performance Monitoring Unit (PMU) enhancements of Diamond Rapids based on Panther Cove P-core microarchitecture and Nova Lake based on Coyote Cove P-core microarchitecture and Arctic Wolf E-core microarchitecture.

9.1 OFF-MODULE RESPONSE (OMR) FACILITY

Diamond Rapids and Nova Lake feature an expanded facility called the Off-Module Response (OMR) facility, which replaces the Off-Core Response (OCR) Performance Monitoring of previous processors. Legacy microarchitectures used the OCR facility to evaluate off-core and multi-core off-module transactions. The properly renamed, OMR facility, improves the OCR capability for scalable coverage of new memory systems of multi-core module systems.

See Section 21.3.10.1.3, "P-Core Off-Core Response Facility," and Section 21.3.10.2.4, "E-Core Off-Core Response Facility," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

9.1.1 Panther Cove and Coyote Cove Microarchitectures OMR Performance Monitoring

The OMR facility in the Panther Cove or Coyote Cove P-cores for Diamond Rapids or Nova Lake, respectively, introduce a new OMR event code 2AH, UMask bits [3:0], and a predefined mask bit value in a dedicated MSR to specify attributes of the off-module transaction. Table 9-1 below lists the event-code, UMask value, and the additional off-module configuration MSR that must be programmed to count off-module response events using IA32_PMCx MSRs.

Table 9-1. OMR Events Supported by Panther Cove and Coyote Cove Microarchitectures

Counter	Event Name	Event Select	UMask	Required Off-Module Response MSR
PMC0-7	OMR	2AH	01H	OFFMODULE_RSP_0 MSR (Address 3E0H)
PMC0-7	OMR	2AH	02H	OFFMODULE_RSP_1 MSR (Address 3E1H)
PMC0-7	OMR	2AH	04H	OFFMODULE_RSP_2 MSR (Address 3E2H)
PMC0-7	OMR	2AH	08H	OFFMODULE_RSP_3 MSR (Address 3E3H)

The OFFMODULE_RSP_x MSR layout is organized to specify the off-module transactions on P-cores as follows:

- Bits [15:0], the request type of a transaction to the uncore
- Bits [47:32], the supplier of the transaction from the uncore
- Bits [49:48], the distance across the interconnect
- Bits [51:50], the line state found
- Bits [55:52], the snoop information / line traffic
- Bit [62], specifies a match on any response, equivalent to setting 55:32 at once

The OMR usage model requires at least one bit from each of these fields to be set. Therefore, programming of at least one Request Type bit, one Supplier bit, one Distance bit, one Line State bit, and one Snoop Information bit, is required for proper event collection. All other bits should be set to 0 by software.

9.1.2 Arctic Wolf Microarchitecture Off-Module Response

The Arctic Wolf E-cores on Nova Lake support the OMR events through programming of event code B7H. Unlike the legacy OCR facility which only supported two subevents, the improved OMR facility has four subevents. Additionally, a new programming restriction of this event requires a specific PMCx to be used per subevent as described in Table 9-2.

Table 9-2. OMR Events Supported by Arctic Wolf Microarchitecture[1]

Counter	Event Name	Event Select	UMask	UMask 2	Required Off-Module Response MSR
PMC0	OMR	B7H	01H	00H	OFFMODULE_RSP_0 MSR (Address 3E0H)
PMC1	OMR	B7H	02H	00H	OFFMODULE_RSP_1 MSR (Address 3E1H)
PMC2	OMR	B7H	04H	00H	OFFMODULE_RSP_2 MSR (Address 3E2H)
PMC3	OMR	B7H	08H	00H	OFFMODULE_RSP_3 MSR (Address 3E3H)

NOTES:

1. These events are tied to PMC0-3 based on the Umask.

The OMR events still require programming an associated OFFMODULE_RSP_x MSR for configuration of the monitored transactions, similar to the OCR usage model. However, the OFFMODULE_RSP_x MSR layout has been updated and organized to specify off-module transactions on E-cores as follows:

- Bits [30:0], the request type of a transaction to the uncore
- Bit [31] specifies a match on any request, equivalent to setting 30:0 at once.
- Bit [47:32], the supplier of the transaction from the uncore
- Bit [49:48], the distance across the interconnect
- Bit [51:50], the line state found
- Bit [55:52], the snoop information / line traffic
- Bit [62] specifies a match on any response, equivalent to setting 55:32 at once
- Bit [63] enables outstanding cycle counts of a specified request type while waiting for any response.

The OMR usage model requires at least one bit from each of these fields to be set. Therefore, programming of at least one Request Type bit, one Supplier bit, one Distance bit, one Line State bit, and one Snoop Information bit, is required for proper event collection. The Request Type bits can be substituted by setting the Any Request bit, and the Supplier, Distance, Line State and Snoop Information bits can all be substituted by setting the Any Response bit. Additionally, the Outstanding Bit, can only be programmed with Request Type bits or the Any Request bit. All other bits should be set to 0 by software.

9.2 PEBS LOAD LATENCY AND STORE LATENCY FACILITY

Diamond Rapids and Nova Lake include PEBS Load Latency and Store Latency support similar to that described in Section 21.9.7, “Load Latency Facility,” Section 21.9.8, “Store Latency Facility,” Section 21.3.10.2.1, “E-core PEBS Load Latency,” and Section 21.3.10.2.2, “E-core PEBS Store Latency,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B. This section describes the PEBS record format changes from previous generations.

The Memory Aux Info field of the PEBS record was updated to support expanded data sources, along with new fields to provide additional, latency impacting and pipeline stall information. The format of these fields is model-specific and described in the following subsections.

9.2.1 Panther Cove and Coyote Cove Microarchitectures Memory Auxiliary Field Layout

The Memory Aux Info fields of the PEBS AUX group for Panther Cove or Coyote Cove P-core microarchitecture is described in Table 9-3.

Table 9-3. PEBS Memory Aux Info Field Layout for Panther Cove and Coyote Cove Microarchitectures

Sub-Field Name	Bits	Description	Valid for Store Latency Facility
Data Source[1]	8:0	An encoded value indicating the memory hierarchy source that satisfied the access. These encodings are detailed in Table 9-4.	V
STLB Clean Hit	9	A value of 1 indicates the access has a clean[1] hit in second-level TLB (STLB).	V
STLB Any Hit	10	A value of 1 indicates the access has any hit in second-level TLB (STLB).	
STLB Miss	11	A value of 1 indicates the access has missed the second-level TLB (STLB).	V
Is Lock	12	A value of 1 indicates the access was part of a locked (atomic) memory transaction.	V
Data Block	13	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.	
Address Block	14	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.	
Fill-Buffers Full	15	A value of 1 indicates that the access was blocked due to FB shortage.	V

NOTES:

1. The Data Source field is split into 2 sub-fields: Data Source [8] indicates an L2 miss and Data Source [7:0] is the encoding for the L2 Miss bit. See Table 9-4.

The breakdown of the expanded Data Source Encodings is provided in Table 9-4. The OMR encoding for an L2 miss Data Source is further explained in Table 9-5.

Table 9-4. Data Source Field Encodings for Panther Cove and Coyote Cove Microarchitectures

Data Source		Description
L2 Miss [8]	Encoding [7:0]	
0	01H	L0 Hit - Minimal latency core cache hit. This request was satisfied by the L0 data cache.
	02H	L1 Hit - This data request was satisfied by the L1 data cache.
	03H	FB Hit - Pending core cache HIT. Outstanding core cache miss to same cache-line address was already underway.
	04H	L2 Hit Clean - This data request was satisfied by the L2 cache with no coherency actions required (snooping).[1]
	07H	L2 Hit Snoop Miss - This data request was satisfied by the L2 and was serviced by another processor core with a cross core snoop where no modified copies were found.
	08H	L2 Hit Snoop Hit - This data request was satisfied by the L2 and was serviced by another processor core with a cross core snoop where modified copies were found.
	09H	Prefetch Promotion - This data request missed L2 but merged into an outstanding hardware prefetch to the same cache-line address, which was initiated from the same processor core (shortened latency).
	0AH	Cross Core Prefetch Promotion - This data request missed L2 but merged into an outstanding hardware prefetch to the same cache-line address, which was initiated from another processor core (shortened latency).
	0EH	TSX Abort - Transactional Synchronization Extensions (TSX) aborted (server only) or reserved.

Table 9-4. Data Source Field Encodings for Panther Cove and Coyote Cove Microarchitectures

Data Source		Description
L2 Miss [8]	Encoding [7:0]	
	0FH	UC - The request was to uncacheable memory.
1	OMR Encoding[2]	This data request missed L2 cache and the information about the data source is encoded as an 8-bit OMR encoding. See Table 9-5.

NOTES:

1. Coherency Actions (e.g., snoops) are required at the L2 cache since in Dual Core Module (DCM) architecture every two cores share a L2 cache.
2. 8-bit OMR Encoding contains all the information used by the processor's OMR facility encoded in an 8-bit field. See Table 9-5.

Table 9-5. OMR Encoding for P-Core and E-Core Microarchitectures

Line Action/ Modifiers [7]	Snoop Information [6]	Line State [5]	Distance [4]	Supplier[3:0]	
Reserved	Reserved	Reserved	Reserved	0000b	Invalid/Catch All
Reserved	Reserved	Reserved	Reserved	0001b	Reserved
00: Not used 01: Snoop miss 10: Snoop hit no FWD 11: Snoop not needed		0: Hit E/S/F 1: Hit M	0: Local CA	0010b	Local CA Shared Cache
1	0: HLE Abort 1: Snoop FWD	0: Hit E/S/F 1: Hit M	0: Local CA 1: Other CA or remote socket (applicable only to HLE Abort)	0011b	Local CA Non-Shared Cache
0: Promoted LLC prefetch 1: Not promoted prefetch	0: Snoop FWD with SB 1: Snoop FWD	0: Hit E/S/F (not applicable for Snoop FWD with WB) 1: Hit M	0: Local socket 1: Remote socket	0100b	Other CA—IO Agent
				0101b	Other CA Shared Cache
				0110b	Other CA Non-Shared Cache
1	1	0: Hit	0: Local CA 1: Other CA or remote socket	0111b	MMIO
0: Promoted LLC prefetch 1: Not promoted prefetch	0: Snoop hit no FWD or miss 1: Snoop unknown	0: Hit	0: Local or remote socket	1000b	Memory Region 0
				1001b	Memory Region 1
				1010b	Memory Region 2
				1011b	Memory Region 3
				1100b	Memory Region 4
				1101b	Memory Region 5
				1110b	Memory Region 6
				1111b	Memory Region 7

9.2.2 Arctic Wolf Microarchitecture Memory Auxiliary Field Layout

The Memory Aux Info fields of the PEBS AUX group for Arctic Wolf E-core microarchitecture is described in Table 9-6.

Table 9-6. PEBS Memory Aux Info Field Layout for Arctic Wolf Microarchitecture

Sub-Field Name	Bits	Description	Valid for Store Latency Facility
Data Source[1]	8:0	An encoded value indicating the memory hierarchy source that satisfied the access. These encodings are detailed in Table 9-7.	V
Prefetch Promotion	9	A value of 1 indicates that the access with an L2 Miss was originally a prefetch that was promoted to a demand. The Data Src will be all zeros to indicate that the source was unable to be captured during promotion.	V
Reserved	10	Reserved.	
STLB Miss	11	A value of 1 indicates the access has missed the second-level TLB (STLB).	V
Is Lock	12	A value of 1 indicates the access was part of a locked (atomic) memory transaction.	V
Data Block	13	A value of 1 indicates the load was blocked since its data could not be forwarded from a preceding store.	
Address Block	14	A value of 1 indicates the load was blocked due to potential address conflict with a preceding store.	
WCB Full	15	A value of 1 indicates that the access was blocked due to Write Combining Buffer (WCB/Fill Buffer) shortage.	V

NOTES:

1. The Data Source field is split into 2 sub-fields: Data Source [8] indicates an L2 miss and Data Source [7:0] is the encoding for the L2 Miss bit. See Table 9-7.

The breakdown of the expanded Data Source Encodings is provided in Table 16-7. The OMR encoding for an L2 miss Data Source is the same as that for P-core, detailed in Table 16-5.

Table 9-7. Data Source Field Encodings for Arctic Wolf Microarchitecture

Data Source		Description
L2 Miss [8]	Encoding [7:0]	
0	00H	Invalid reset value.
	01H	L1 Hit - Minimal latency core cache hit. This request was satisfied by the L1 data cache.
	02H	WCB Hit - Pending core cache Hit. Outstanding core cache miss to same cache-line address was already underway.
	03H	L2 Hit Clean - This data request was satisfied by the L2 cache with no coherency actions required (snooping).[1]
	04H	L2 Hit Snoop Miss - This data request was satisfied by the L2 cache and was serviced by another core with a cross-core snoop where no modified copies were found.
	05H	L2 Hit Snoop Hit - This data request was satisfied by the L2 cache and was serviced by another core with a cross-core snoop where modified copies were found.
	06H	UC - The request was to uncacheable memory.

Table 9-7. Data Source Field Encodings for Arctic Wolf Microarchitecture

Data Source		Description
L2 Miss [8]	Encoding [7:0]	
	07H	I/O - Request of input/output operation.
1	OMR Encoding[2]	This data request missed L2 cache and the information about the data source is encoded as an 8-bit OMR encoding. See Table 9-5.

NOTES:

1. Coherency actions (i.e., snoops) are required at the L2 cache since multicore modules share an L2 cache.
2. 8-bit OMR Encoding contains all the information used by the processor's OMR facility encoded in an 8-bit field (see Table 9-5).

9.3 COUNTER RESTRICTIONS

9.3.1 Panther Cove and Coyote Cove Microarchitectures Counter Restrictions

PMU counter restrictions for the Panther Cove or Coyote Cove P-core microarchitecture based on Diamond Rapids or Nova Lake, respectively, are summarized in the table below.

Table 9-8. Event Counter Restrictions for Panther Cove and Coyote Cove Microarchitectures

Event Name	Event Select	UMask	Counter Restriction (Supported Counters)
OFFCORE_REQUESTS_OUTSTANDING.*	20H	* (any UMask)	0-3
INST_DECODED.DEC_BW_3MU	75H	02H	0-3
UOPS_DECODED.HEAVY_INST_UOPS	76H	01H	0-3
IDQ.*	79H	*	0-3
TOPDOWN.BAD_SPEC_SLOTS, TOPDOWN.BR_MISPREDICT_SLOTS	A4	04H, 08H	0
PORT_PUSHOUT.ANY	ABH	08H	3
MEM_TRANS_RETIRED.STORE_SAMPLE	CDH	02H	0-1
MEM_INST_RETIRED.*, MEM_LOAD_RETIRED.*, MEM_LOAD_L2_MISS_RETIRED.*, MEM_LOAD_MISC_RETIRED.*	D0H, D1H, D6H, D4H	*	0-3
EXE_RETIRED.*	DFH	*	0-3
AMX_OPS_RETIRED.*	CEH	*	0
UOPS_EXECUTED.THREAD	B1H	01H	3
UOPS_DISPATCHED.INT_EU_ALL, UOPS_DISPATCHED.ALU	B2H	01H, 02H	
MEMORY_ACTIVITY.L3M_PENDING	47H	08H	0-3
MEMORY_STALLS.L3, MEMORY_STALLS.DRAM	46H	04H, 08H	0-3
DCACHE_PENDING.LOAD	48H	01H	0-3

9.3.2 Arctic Wolf Microarchitecture Counter Restrictions

The PMU event counter restrictions for the Arctic Wolf E-core microarchitecture based on Nova Lake are summarized in Table 9-9.

Table 9-9. Event Counter Restrictions for Arctic Wolf Microarchitecture

Event Name	Event Select	UMask (Supported UMasks)	Counter Restriction (Supported Counters)
OMR.*.*	B7H	01H, 02H, 04H, 08H	0-3 direct mapped
OMR ,*REQUEST_TYPE*>.OUTSTANDING	75H	01H, 02H	0-1 direct mapped
MEM_LOAD_UOPS_L2_MISS RETIRED. L3_HIT_SNOOP_HITM_LOCAL_CBB[1]	D3H	21H	0
MEM_LOAD_UOPS_L2_MISS RETIRED. L3_HIT_SNOOP_HITM_NON_LOCAL_ CBB[1]	D3H	22H	0

NOTES:

1. MEM_LOAD_UOPS_L2_MISS_RETIRED.L3_HIT_SNOOP_HITM* events listed must be “taken alone,” more specifically no other events can be programmed with these events. They must truly be collected individually with PMC1-7 not programming any other events at that time.

CHAPTER 10
PERFMON VIRTUALIZATION WITH PERFMON MASKING

This chapter describes a VMX extension called **PerfMon masking**. It supports virtualization of the processor's performance-monitoring architecture (PerfMon).

The PerfMon architecture is based on a set of performance-monitoring counters. In general, each counter is associated with one or more MSRs. A virtual-machine monitor (VMM) may choose to virtualize the counters by assigning some to guest software in a virtual machine while retaining other counters for its own use.

The existing VMX architecture supports this counter partitioning with existing features that allow a VMM control over whether guest software has access to specified MSRs. The VMM can allow guest access to the MSRs associated with the guest-assigned counters while preventing access to the MSR associated with those counters it is retaining for itself.

Partitioning control of the controlling MSRs does not suffice for the counter-partitioning usage. This is for two reasons: (1) some MSRs are associated with all the counters (e.g., by supporting a bitmap with one bit per counter); and (2) the RDPMC (Read Performance-Monitoring Counter) instruction may read the current value of any of the performance-monitoring counters. The existing VMX architecture does not provide a mechanism to allow guest software access to only some bits in an MSR (e.g., those associated with counters assigned to the guest) or one to allow guest executions of RDPMC to access only some performance-monitoring counters.

A VMM could intercept all accesses to the affected MSRs and all executions of RDPMC, but doing so might affect performance adversely. To address performance concerns, PerfMon masking extends the existing VMX architecture by providing fine-grained control over what bits can be accessed in certain PerfMon-related MSRs as well as control over which performance-monitoring counters can be accessed with RDPMC.

Section 10.1 identifies the MSRs affected by PerfMon masking.

Section 10.2 identifies new VMCS support (a new control and a new field).

Section 10.3 details the changes to VMX non-root operation.

Section 10.4 identifies potential undefined behaviors when accessing PerfMon MSRs during VMX transitions.

Section 10.5 provides some guidelines for VMM software.

10.1 PERFMON MSRS AFFECTED BY PERFMON MASKING

Two of the principal MSRs used by PerfMon are IA32_PERF_GLOBAL_CTRL and IA32_PERF_GLOBAL_STATUS. The former gives software control over individual counters, and the latter provides status of those counters. The following items detail the status of these MSRs:

- Bits 31:0 of IA32_PERF_GLOBAL_CTRL are used to enable general-purpose performance-monitoring counters 0–31. Each corresponding bit in IA32_PERF_GLOBAL_STATUS indicates whether the corresponding counter has overflowed.
- Bits 47:32 of IA32_PERF_GLOBAL_CTRL are used to enable fixed-function performance-monitoring counters 0–15. Each corresponding bit in IA32_PERF_GLOBAL_STATUS indicates whether the corresponding counter has overflowed.
- Bit 48 of IA32_PERF_GLOBAL_CTRL enables performance metrics. Bit 48 of IA32_PERF_GLOBAL_STATUS indicates that a performance-metrics-related resource has overflowed.
- Bits 63:54 of IA32_PERF_GLOBAL_STATUS indicate a variety of PerfMon-related status information.

Certain other MSRs have formats derived from those of IA32_PERF_GLOBAL_CTRL and IA32_PERF_-GLOBAL_STATUS:

- IA32_PERF_GLOBAL_STATUS_SET and IA32_PERF_GLOBAL_STATUS_RESET are MSRs to which software can write to set or clear bits in IA32_PERF_GLOBAL_STATUS. They each have the same format as that MSR. Reads of these two MSRs always return zero.

- For each general-purpose performance-monitoring counter n, IA32_PMC_GPn_CFG_B indicates, in each of bits 47:0, whether the counter should be reloaded when the counter corresponding to the bit (see above) overflows. Similarly, bit 48 indicates whether performance metrics should be cleared on overflow.
- For each fixed-purpose performance-monitoring counter m, IA32_PMC_FXm_CFG_B indicates, in each of bits 47:0, whether the counter should be reloaded when the counter corresponding to the bit (see above) overflows. Again, bit 48 indicates whether performance metrics should be cleared on overflow.
- The IA32_FIXED_CTR_CTRL MSR has a different format. For each n, $0 \le n \le 7$, bits 4n+3:4n and bits 4n+35:4n+32 correspond to fixed-purpose performance-monitoring counter n.

(For more details, see Chapter "Performance Monitoring" in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B.)

PerfMon masking uses the fact that these MSRs have similar formats to control them all with a single mask, called the PerfMon mask.

As detailed in Section 10.3, PerfMon masking affects reads from IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, and IA32_FIXED_CTR_CTRL; it affects writes to IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS_SET, IA32_PERF_GLOBAL_STATUS_RESET, IA32_PMC_GPn_CFG_B, IA32_PMC_FXm_CFG_B, and IA32_FIXED_CTR_CTRL.

There is a bit in the PerfMon mask corresponding to each performance-monitoring counter. Since each execution of RDPMC reads from a single counter (or performance metrics), the PerfMon mask is used also to control executions of that instruction.

10.2 VMCS SUPPORT

This feature defines a new VM-execution control called "PerfMon masking." It is tertiary processor-based VM-execution control 17.

This feature also introduces a new VMCS field. **PerfMon mask** is a new 64-bit VM-execution control field. Software can access this field with VMREAD or VMWRITE using the encoding pair 2054H/2055H. This field is supported on processors that support the 1-setting of the "PerfMon masking" VM-execution control.

The format of the PerfMon mask corresponds to the MSR formats identified in Section 10.1:

- Each of bits 31:0 corresponds to a general-purpose performance-monitoring counter; bit n ($0 \le n \le 31$) corresponds to general-purpose counter n.
- Each of bits 47:32 corresponds to a fixed-function performance-monitoring counter; bit n ($32 \le n \le 47$) corresponds to fixed-function counter n–32.
- Bit 48 corresponds to performance metrics.
- Bits 63:54 correspond to the miscellaneous bits in IA32_PERF_GLOBAL_STATUS.

10.3 CHANGES TO VMX NON-ROOT OPERATION

This section describes changes in VMX non-root operation when the "PerfMon masking" VM-execution control is 1. The changes affect three operations: MSR reads (Section 10.3.1), MSR writes (Section 10.3.2), and executions of RDPMC (Section 10.3.3).

10.3.1 Changes to MSR Reads

PerfMon masking affects reads from three MSRs identified below. These includes reads using the RDMSR, RDMSRLIST, and URDMSR instructions. PerfMon masking applies only to executions of those instructions that do not cause a fault or a VM exit.

PerfMon masking affects reads from the IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS, and IA32_FIXED_CTR_CTRL MSRs. In each case, the value returned by a read is controlled by an effective mask determined as follows:

- IA32_PERF_GLOBAL_CTRL: Bits 51:0 are affected, so the effective mask is the logical OR of the PerfMon mask and FFF00000_00000000H.
- IA32_PERF_GLOBAL_STATUS: All bits are potentially masked, so the effective mask is the PerfMon mask.
- IA32_FIXED_CTR_CTRL: Because of the format of this MSR, the effective mask is defined so that bits 4n+3:4n and bits 4n+35:4n+32 are set if and only if bit 32+n of the PerfMon mask is set (for $0 \le n \le 7$).

The value returned by a read of any of these MSRs is the logical AND of the MSR value and the effective mask indicated above. Specifically, zero is returned for any bit cleared in the effective mask.

10.3.2 Changes to MSR Writes

PerfMon masking affects writes to the MSRs identified below. These includes writes using the WRMSR, WRMSRLIST, and WRMSRNS instructions.[1] PerfMon masking applies only to executions of those instructions that do not cause a fault or a VM exit.

As explained below, PerfMon masking may cause an MSR write to fault, or it may change the value written. A fault caused by PerfMon masking has priority below any fault-like VM exit to which the MSR write may be subject. For an MSR write that does not fault or cause a VM exit, PerfMon masking may change the value written to the MSR.

PerfMon masking affects writes to the IA32_PERF_GLOBAL_CTRL, IA32_PERF_GLOBAL_STATUS_SET, IA32_PERF_GLOBAL_STATUS_RESET, and IA32_FIXED_CTR_CTRL MSRs. It also affects writes to IA32_PMC_GPn_CFG_B (for each general-purpose performance-monitoring counter n) and to IA32_PMC_FXm_CFG_B (for each fixed-purpose performance-monitoring counter m).

In each case, the operation of a write is controlled by an effective mask determined as follows:

- IA32_PERF_GLOBAL_CTRL, IA32_PMC_GPn_CFG_B, IA32_PMC_FXm_CFG_B: Bits 51:0 are affected, so the effective mask is the logical OR of the PerfMon mask and FFF00000_00000000H.
- IA32_PERF_GLOBAL_STATUS_SET, IA32_PERF_GLOBAL_STATUS_RESET: All bits are potentially masked, so the effective mask is the PerfMon mask.
- IA32_FIXED_CTR_CTRL: Because of the format of this MSR, the effective mask is defined so that bits 4n+3:4n and bits 4n+35:4n+32 are set if and only if bit 32+n of the PerfMon mask is set (for $0 \le n \le 7$).

A write to any of these MSRs causes a general-protection exception (#GP(0)) if its source operand sets a bit for which the corresponding bit in the effective mask is clear.

If such a write does not cause a fault, the write does not modify in the MSR any bit for which the corresponding bit in the MSR is clear. Thus, if the MSR's current value is VAL, the value of the source operand is SRC, and the effective mask is EFFMASK, then the MSR is written with (VAL & ~EFFMASK) | (SRC & EFFMASK).[2]

10.3.3 Changes to RDPMC

Operation of the RDPMC instruction is based on the value of the ECX register. In particular, ECX[31:16] determines the type of the read and (when relevant), ECX[15:0] determines the index of the counter being read. The following items explain how different types of reads are affected by PerfMon masking:

- If the type is 0, the read is of a general-purpose performance-monitoring counter. In this case, execution of RDPMC causes a general-protection exception (#GP(0)) if the index is at most 31 and the bit in the PerfMon mask selected by the index is 0.
- If the type is 4000H, the read is of a fixed-function performance-monitoring counter. In this case, execution of RDPMC causes a #GP(0) if the index is at most 15 and the bit in the PerfMon mask selected by the sum of 32 and the index (a value in the range 32–47) is 0.

---

1. At the time of this writing, the UWRMSR instruction cannot be used to write to any of these MSRs. If a future processor allowed UWRMSR to write to one of these MSRs, such writes would be subject to PerfMon masking.
2. Reads of IA32_PERF_GLOBAL_STATUS_SET and IA32_PERF_GLOBAL_STATUS_RESET always return zero. For that reason, a masked write to either of those MSRs writes SRC & EFFMASK, where SRC is the value of the source operand and EFFMASK is the PerfMon mask.

- If the type is 2000H, the read is of performance metrics. In this case, execution of RDPMC causes a #GP(0) if bit 48 of the PerfMon mask is 0.

As with MSR writes (Section 10.3.2), the priority of the faults caused by PerfMon masking is below that of any fault-like VM exit the instruction may incur.

10.4 ACCESS TO PERFMON MSRS DURING VMX TRANSITIONS

Software can specify that certain MSRs are loaded or saved during VMX transitions as follows:

- VM entries load MSRs from the VM-entry MSR-load area.
- VM exits save MSRs to the VM-exit MSR-store area.
- VM exits load MSRs from the VM-exit MSR-load area.

If the "PerfMon masking" VM-execution control is 1, the following uses of these areas may result in undefined behavior:

- Use of any of the following MSRs in the VM-entry MSR-load area: IA32_PERF_GLOBAL_CTRL, IA32_PERF_-GLOBAL_STATUS_SET, IA32_PERF_GLOBAL_STATUS_RESET, IA32_FIXED_CTR_CTRL; IA32_PMC_G-Pn_CFG_B (for each general-purpose performance-monitoring counter n); and IA32_PMC_FXm_CFG_B (for each fixed-purpose performance-monitoring counter m).
- Use of any of the following MSRs in the VM-exit MSR-store area: IA32_PERF_GLOBAL_CTRL, IA32_PERF_-GLOBAL_STATUS, and IA32_FIXED_CTR_CTRL.

Software should avoid including any of the indicated MSRs when setting the "PerfMon masking" VM-execution control to 1.

The setting of "PerfMon masking" does not affect the behavior of MSR loading from the VM-exit MSR-load area or the behavior of loading and saving of IA32_PERF_GLOBAL_CTRL using the VMCS fields defined for that MSR.

10.5 SOFTWARE GUIDELINES

Any software that accesses PerfMon counters is expected to use only counters enumerated through CPUID.0AH:EAX.NUM_GP_CTRS[15:8], CPUID.0AH:ECX.FIXED_CTR_MASK[31:0], CPUID.0AH:EDX.NUM_-FIXED_CTR[4:0], CPUID.23H.01H:EAX.GP_COUNTERS[31:0], CPUID.23H.01H:EBX.FIXED_COUNTERS[31:0], and IA32_PERF_CAPABILITIES[15]. In virtualized environments, the host is expected to virtualize these CPUID values such that the guest will enumerate only guest-owned counters.

The host may use MSR bitmaps to intercept guest accesses to MSRs corresponding only to host-owned counters (i.e., counter related MSRs that are not affected by PerfMon masking). The counter related MSRs are listed in Table 10-1 below.

Table 10-1. Host-Owned PerfMon MSRs Intercepted Using MSR Bitmaps

Name	Range	Description	Enumeration
IA32_PMC_GPx_CFG_A	1901H + x*4	Generic counter x configuration A.	CPUID.0AH:EAX[7:0] ≥ 1 CPUID.0AH:EAX[15:8] CPUID.23H.1:EAX[31:0]
IA32_PMC_GPx_CFG_B	1902H + x*4	Generic counter x configuration B. Note that this is about intercepting guest accesses to CFG_B MSRs corresponding to host-owned counters; to prevent the guest from setting bits corresponding to host-owned counters in CFG_B MSRs corresponding to guest-owned counters, the host would use PERFMON_MASK.	Same as above.
IA32_PMC_GPx_CFG_C	1903H + x*4	Generic counter x configuration C.	Same as above.
IA32_PMC_GPx_CTR	1900H + x*4	Generic counter x.	Same as above.

Table 10-1. Host-Owned PerfMon MSRs Intercepted Using MSR Bitmaps

Name	Range	Description	Enumeration
IA32_PMC_FXx_CFG_C	1983H + x*4	Fixed counter x configuration C.	CPUID.0AH:EAX[7:0] ≥ 1 CPUID.0AH:ECX[31:0] CPUID.0AH:EDX[4:0] CPUID.0AH:EAX[15:8] CPUID.23H.1:EBX[31:0]
IA32_PMC_FXx_CFG_B	1982H + x*4	Fixed counter x configuration B. Note that this is about intercepting guest accesses to CFG_B MSRs corresponding to host-owned counters; to prevent the guest from setting bits corresponding to host-owned counters in CFG_B MSRs corresponding to guest-owned counters, the host would use PERFMON_MASK.	Same as above.
IA32_PMC_FXx_CTR	1980H + x*4	Fixed counter x.	Same as above.
IA32_LEGACY_PMCx	00C1H + x	Generic counter x, limited range.	CPUID.0AH:EAX[7:0] ≥ 1 CPUID.0AH:EAX[15:8] CPUID.23H.1:EAX[31:0]
IA32_PERFEVTSELx	0186H + x	Generic counter x configuration register.	Same as above.
IA32_FIXED_CTRx	0309H + x	Fixed counter x.	Same as above.
IA32_A_PMCx	04C1H + x	Generic counter x, full range.	CPUID.0AH:EAX[7:0] ≥ 1 IA32_PERF_CAPABILITIES [13] = 1 CPUID.0AH:EAX[15:8] CPUID.23H.1:EAX[31:0]
RELOAD_FIXED_CTRx	1309H + x	PEBS reload value for fixed counter x.	Model specific. Not available if support for IA32_PMC_FXx_CFG_C.
RELOAD_PMCx	14C1H + x	PEBS reload value for generic counter x.	Model specific. Not available if support for IA32_PMC_GPx_CFG_C.
PERF_METRICS	0329H	Performance metrics ratios.	CPUID.0AH:EAX[7:0] ≥ 1 IA32_PERF_CAPABILITIES [13] = 1
IA32_PERF_GLOBAL_INUSE	0392H	In-use MSR. VMM should handle read/write exit and provide filtered view and access control for host counters. It can choose to clear the PMI bit [63] or virtualize by deriving the state through OR-ing all the enabled guest counter PMI bits together.	CPUID.0AH:EAX[7:0] ≥ 4

The following items are guidelines for VMMs using PerfMon masking and guest PerfMon drivers. Note that these guidelines are not enforced by the CPU.

- Performance metrics. PERF_METRICS is tightly coupled with fixed counter 3. VMMs should define PERF_METRICS and fixed counter 3 as both host-owned or both guest-owned (in other words, set PERFMON_MASK[48] = PERFMON_MASK[34]).
- Reserved bits. The VMM should clear reserved bits 51:49 in the PerfMon mask to zero. VMMs should set these bits to 0, even though they are ignored by the CPU, to ensure compatibility with future implementations.
- PEBS buffer overflow (setting PerfMon mask bit 54). When the guest is allowed to use PEBS, the VMM should set bit 54 of the PerfMon mask; otherwise, the bit should be cleared. A VMM that allows the VM to use PEBS and access physical PEBS resources should set PERFMON_MASK[54] to 1, to provide the VM with an access to the PEBS overflow bit in IA32_PERF_GLOBAL_STATUS MSRs. Otherwise, the VMM should set PERFMON_MASK[54] to 0, so that the VM will not be able to affect PEBS use by the VMM.
- ToPA Buffer Overflow (setting PerfMon mask bit 55). When the guest is allowed to use Processor Trace, the VMM should set bit 55 of the PerfMon mask(ToPA PMI); otherwise, the bit should be cleared.

- LBR freeze (setting PerfMon mask bit 58). When the guest is not allowed to use LBRs, the VMM should clear bit 58 of the PerfMon mask (LBRSFrozen) to prevent the guest freezing host LBRs. When the guest is allowed to use LBRs, it should be made guest owned.

  Host PMIs can temporarily stop the guest LBR when the guest enabled LBR is freezing. The assumption is that the guest will only enable that if it can tolerate any LBR freezing actions. When using LBR Call stack mode, the guest is expected to avoid setting IA32_DEBUGCTL.FreezeLBRsOnPMI=1, but instead rely on CPL filtering or PEBS LBR saving. Alternatively, the host could avoid triggering PMIs.
- Counter Freeze (setting PerfMon mask bit 59). When IA32_DEBUGCTL[12], FreezePerfmonOnPMI, is enabled in a guest, guest counter overflows that raise PMI freeze all counters, including the host counters until the host PMI can clear IA32_PERF_GLOBAL_STATUS[59], CountersFrozen. Also a guest setting IA32_PERF_-GLOBAL_STATUS bit 59, CountersFrozen, could stop the host counters. The VMM should ensure that bit 59 of the PerfMon mask (CountersFrozen) is always cleared to prevent the guest from freezing host counters. The VMM can forbid the guest from setting IA32_DEBUGCTL[12], FreezePerfmonPMI. Alternatively, the VMM can virtualize the FreezePerfmonOnPMI bit for the guest, and if enabled and host counters are active, virtualize counters freezing by managing the counter enables on entry/exit, and intercept IA32_PERF_GLOBAL_STATUS and the counter control registers to supply virtualized freezing state to the guest. The host should avoid using CounterFreezing for its own counters.
- SGX anti side-channel interference (PerfMon mask bit 60). When the guest is allowed to use an SGX enclave, an enabled host counter may result in the status bit for the ASCI (Anti Side Channel Interference) bit being set in IA32_PERF_GLOBAL_STATUS to indicate that the counter was stopped during the enclave execution. When the guest is not allowed to use SGX, the VMM should clear this bit in the PerfMon mask. Otherwise, it can either make the bit host- or guest-owned. If the bit is host-owned, it might be set when the host has an active counter and the guest does not.
- VM migration. VMMs can use PerfMon masking to aid the migration of VMs that use Perfmon to CPUs with a different number of counters. Specifically, the VMM would expose only counters that are common to all CPUs in the migration pool as guest-owned counters.

CHAPTER 11
MACHINE CHECK ERROR CODES

Encoding of model-specific and other information fields is different across processor families. This chapter documents the differences for Machine Check Architecture Error Code (MCACOD) and Model Specific Error Code (MSCOD) encodings for the next generation Intel® Xeon® Processor Family (Diamond Rapids). For MCACOD and MSCOD encodings of processors prior to Diamond Rapids, see the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3.

11.1 INCREMENTAL DECODING INFORMATION: NEXT GENERATION INTEL® XEON® PROCESSOR FAMILY

In the next generation Intel® Xeon® Processor Family with CPUID DisplayFamily_DisplaySignature 13_01H, incremental error codes for internal machine check errors from the CBB PCU controller are reported in the register bank IA32_MC4. Table 11-1 lists model-specific fields to interpret error codes applicable to IA32_MC4_STATUS.

11.1.1 Internal Machine Check Errors

Table 11-1. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes[1]	15:0	MCACOD	
MCACOD[2]	15:0	Internal Errors	040BH: Scan-at-Field Error
Model Specific Errors	31:16	Reserved, except for the following	Model specific error code bits 31:16. For MCACOD values other than 040BH, MSCOD encoding should be interpreted as: 0816H: MCA_DVFS_COUNTER_EXPIRED 0823H: MCA_UNSUPP_RESP 0825H: MCA_FUSA_MBIST 0826H: MCA_SA_PLL_LOCKED 0827H: MCA_XTAL_FREQ_MON_ERROR 0828H: MCA_PMSB_FUSE_PULLER_ERROR 0829H: MCA_GPSB_FUSE_PULLER_ERROR 0830H: MCA_PCODE_WATCHDOG 0831H: MCA_BGR_FUSA_ERR 0832H: MCA_CORE_FIVR_ERR 0833H: MCA_D2D_FIVR_ERR 0834H: MCA_MLC_FIVR_ERR 0835H: MCA_RING_FIVR_ERR 0836H: MCA_D2D_MONF_PLL_FUSA_0 0837H: MCA_D2D_MONF_PLL_FUSA_1 0838H: MCA_TOPO_FUSA_PLL_ERR 0839H: MCA_BASE_FUSA_PLL_ERR

Table 11-1. Machine Check Error Codes for IA32_MC4_STATUS

Type	Bit No.	Bit Function	Bit Description
			0840H: MCA_UCIE_RESET_BUS_STOP_ON_ERROR 0841H: MCA_RSRC_ADAPT_0_ERR 0842H: MCA_RSC_ADAPT_1_ERR 0845H: MCA_TOP1_FUSA_PLL_ERR 0846H: MCA_TOP2_FUSA_PLL_ERR 0847H: MCA_TOP3_FUSA_PLL_ERR 0848H: UC_PATCH_LOAD_ACK_ERROR 0C01H: MCA_UC_IRAM_DOUBLE_ERROR 0C02H: MCA_UC_DOUBLE_EXCEPTION_ERROR 0C03H: MCA_UC_TELEM_SRAM_ERROR 0C04H: MCA_UC_DRAM_DOUBLE_ERROR 0C05H: MCA_UC_TPMI_SRAM_ERROR 1401H: MCA_FIVR_HI_YELLOW 1402H: MCA_FIVR_HI_RED
Status Register Validity Indicators[1]	63:32		Section17.3.2.2. High bits are architectural. See figure 17-6.

NOTES:

1. These fields are architecturally defined. Refer to Chapter 18, "Machine-Check Architecture," in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3, for more information.
2. The internal error codes may be model-specific.

11.1.2 Integrated Memory and I/O Hub PCU Machine Check Errors

Machine check error codes associated with the integrated memory and I/O hub (IMH) PCU are reported in the IA32_MC11_STATUS MSRs.

Table 11-2. IMH PCU Machine Check Error Codes for IA32_MC11_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes[1]	15:0	MCACOD	
MCACOD[2]	15:0	Internal Errors	0402H: PCU internal errors
Model Specific Errors	31:16	Reserved, except for the following	Model specific error code bits 31:16. If MCACOD = 040CH, MSCOD encoding should be interpreted as: 0001H: CR4_MCE_CLEAR: MCE when CR4.MCE is clear 0002H: MCE_MCIP_SET: MCE when MCIP bit is set 0003H: MCE_UNDER_WFS: MCE under WPS 0004H: MCE_LT_HANDSHAKE: Unrecoverable error during security flow execution 0005H: TRIPLE_FAULT: SW triple fault shutdown 0006H: VMX_ABORT: VMX exit consistency check failures 0007H: RSM_CONSISTENCY_FAIL: RSM consistency check failures 0008H: SMM_PROTECTED_ENTRY_FAIL: Invalid conditions on protected mode SMM entry 0009H: UCODE_PATCH_LOAD_FAIL: Unrecoverable error during security flow execution If MCACOD != 040CH, MSCOD encoding should be interpreted as: 0002H: Power Management Unit microcontroller double-bit ECC error 0003H: Power Management Unit microcontroller double-bit ECC error 0008H: Power Management Unit microcontroller error 0009H: Power Management Unit microcontroller error 000AH: Power Management Unit microcontroller Patch Load error 000BH: Power Management Unit microcontroller POReqValid error 000CH: Power Management Unit microcontroller RAM address error 000DH: Power Management Unit microcontroller RAM access error 0010H: If MCI_MISC.MODEL_SPECIFIC_INFORMATION is set to 1, indicates Pcode Watchdog Timer expired. If MCI_MISC.MODEL_SPECIFIC_INFORMATION is set to 3, indicates Power Management Unit TeleSRAM doublebit ECC error detected. 0020H: Power Management Agent signaled Error 0030H: Power Management Unit Microcontroller Error 00A0H: Power Management Unit HPMSRAM double-bit ECC error detected 00B0H: Power Management Unit TPMISRAM double-bit ECC error detected 0900H: MCA_TSC_DOWNLOAD_TIMEOUT 0A00H: MCA_INVALID_XTALFREQ_RANGE 0B00H: MCA_GPSB_TIMEOUT 0C00H: MCA_PMSB_TIMEOUT 0D00H: MCA_CFG_ACK_TIMEOUT 2300H: MCA_PCU_SVID_ERROR 3500H: MCA_SVID_LOADLINE_INVALID 3600H: MCA_SVID_ICCMAX_INVALID 4A00H: MCA_FIVR_PD_HARDERR 4C00H: MCA_HPM_DOUBLE_BIT_ERROR_DETECTED

Table 11-2. IMH PCU Machine Check Error Codes for IA32_MC11_STATUS

Type	Bit No.	Bit Function	Bit Description
			5800H: MCA_INVALID_MEMORY_FREQUENCY 6300H: MCA_SVID_VCCIN_PROTOCOL_ERROR 6A00H: MCA_SPPR_TIMEOUT 6F00H: MCA_INVALID_SID_ERROR 7000H: MCA_INVALID_REMOTE_LEGACY_AGENT_ERROR 7100H: MCA_INVALID_REMOTE_LT_AGENT_ERROR 7B00H: MCA_THERMAL_SENSOR_INVALID 7C00H: MCA_CXL_DEVICE_NO_CREDIT 7E00H: MCA_RECOVERABLE_DIE_THERMAL_TOO_HOT 8300H: MCA_PKGS_RECOVERABLE_RESET_PREP_ACK_TIMEOUT
Status Register Validity Indicators[1]	63:32		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 18, “Machine-Check Architecture,” in the Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3, for more information.
2. The internal error codes may be model-specific.

11.1.3 MCCHAN Machine Check Errors

Machine check codes associated with MCCHAN for the next generation Intel Xeon Processor Family with CPUID DisplayFamily_DisplaySignature of 13_01H are reported in the MSRs IA32_MC19_STATUS—IA32_MC26_STATUS.

Table 11-3. Machine Check Error Codes for IA32_MCi_STATUS (i=19-26)

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes[1]	15:0	MCACOD	Memory Controller error format: 000F 0000 1MMM CCCC
Model Specific Errors	31:16	Reserved, except for the following	0001H: CMI Request Address Parity Error (APPP) 0002H: CMI Wr data parity error 0004H: CMI Wr BE parity error 0008H: Transient or Correctable Error for Patrol Reads 0010H: UnCorr Patrol Scrub Error 0020H: Nontransient or Transient Correctable Error for Spare Reads 0040H: UnCorr Spare Error 0080H: Transient or Correctable Error for Demand or Underfill Reads 00A0H: Uncorrectable Error for Demand or Underfill Reads 00B0H: Poisoned Read Data when Poison Disabled 00C0H: Read 2LM MetaData Error for Demand, Underfill, or Patrol/Spare 0102H: WDB Read ECC Error 0104H: WDB BE Read Parity Error 0106H: WDB Read Persistent ECC Error 0108H: DDR Link Fail 0109H: Illegal incoming opcode 0200H: DDR CAParity or WrCRC Error 0240H: Decoder structure error

Table 11-3. Machine Check Error Codes for IA32_MCi_STATUS (Contd.)(i=19-26)

Type	Bit No.	Bit Function	Bit Description
			0400H: MC internal address parity error 0813H: CMI Credit Oversubscription Error 0814H: CMI Total Credit Count Error 0815H: CMI Reserved Credit Pool Error 0832H: MC Internal Errors 0834H: MC Tracker RDCMP RF parity error 0836H: MC Tracker WRCMP RF parity error
Status Register Validity Indicators[1]	63:32		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 18, "Machine-Check Architecture," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for more information.

Additional information is reported in MSRs IA32_MC19_MISC—IA32_MC26_MISC.

Table 11-4. MCCHAN Machine Check Error Codes for IA32_MCi_MISC (i=19-26)

Type	Bit No.	Bit Function	Bit Description
Model Specific Errors[1]	5:0	RECOVERABLE_ADDRESS_LSB	See Figure 17-8 in Chapter 17 Volume 3B of the Intel® 64 and IA-32 Architectures Software Developer's Manual.
	8:6	ADDRESS_MODE	See Table 17-3 in Chapter 17 Volume 3B of the Intel® 64 and IA-32 Architectures Software Developer's Manual.
	18:9	Column Address	Logged value is column address [11:2] or shift this field left by 2 to get real column address.
	16:9	WDB Parity or ECC Error (when MSCOD = 102H, 104H, or 106H)	This field indicates the WDB ID.
	31:9	Proprietary Error (when MSCOD ≥ 0800H)	This is valid for MSCOD 0800H or above.
	36:19	Row Address	
	38:37	Bank Address	
	41:39	Bank Group	
	42	Failed Device	Set to 1 if failed device number is valid.
	43		Failed x8 half device valid (not valid if bit[63] = 1 or bit[42] = 0): Set to 1 if failure identified is in one half device only for 5x8 DIMM. Valid for 5x8 DIMM only.
	48:44	Failed Device Number	Holds the failed device number (not valid if bit[63] = 1 or bit[42] = 0). For non-ADDDC, the device number is between 0-9. For ADDDC, the device number is between 0-8 and 10-19 (9 is the spare device).
	49	Sub-channel ID	In DDR mode, this is the sub-channel ID.
	50	Reserved	Reserved
	54:51	CBit	This field is for CBit with bit 54 as spare to accommodate possible future growth in size. Logged value: {1'b0,sub_rank[2:0]}
	56:55	Chip Select	Rank in the format as defined in RETRY_RD_ERR_lOG_ADDRESS1.CHIP_SELECT.
	57	Reserved	Reserved

Table 11-4. MCCHAN Machine Check Error Codes for IA32_MCi_MISC (Contd.) (i=19-26)

Type	Bit No.	Bit Function	Bit Description
	61:58	DDR ECC Mode	ECC mode is not valid if bit[63] = 1. 0001b: SDDC 128b 1LM 0010b: SDDC 125b 1LM 0011b: SDDC 125b 2LM 0100b: ADDDC 80b 1LM 0101b: ADDDC 80b 2LM 1001b: 5x8 128b 1LM 1010b: 5x8 125b 1LM 1011b: 5x8 125b 2LM 0000b: Invalid ECC Mode Other values: Reserved
	62		This is valid for most error types. It indicates if corresponding imc0_mc_addr has a valid channel address.
	63	Transient	Indicates if the error was transient error; transient error is indicated for demand reads, underfill reads for partial writes, patrol and sparing reads.

NOTES:

1. Which of these fields are valid depends on the error type. Bits 63:9 are only valid if IA32_MCi_STATUS.OTHER_INFO[1] = 1.

11.1.4 Intel® UPI Machine Check Errors

Table 11-5 provides information for interpreting model-specific fields for Intel® Xeon® Processor Family supporting Ultra Path Interconnect (UPI). The incremental machine check error codes related to UPI are reported in register bank IA32_MC18.

Table 11-5. Machine Check Error Codes for IA32_MC18_STATUS

Type	Bit No.	Bit Function	Bit Description
MCA Error Codes[1]	15:0	MCACOD	
Model Specific Errors	31:16	Reserved, except for the following	0000H: Internal Parity Error 0001H: Internal Parity Error 0002H: Internal Parity Error 0005H: Internal Parity Error 0008H: Internal Queue Overflow Error 0009H: UPI link layer buffer overflow 000AH: Internal Queue Underflow Error 000BH: UPI link layer buffer underflow 000CH: Link layer or internal credit overflow 0011H: Internal Queue Overflow Error 0012H: Internal Queue Overflow Error 0013H: Internal Queue Underflow Error 0014H: Internal Queue Underflow Error

Table 11-5. Machine Check Error Codes for IA32_MC18_STATUS (Contd.)

Type	Bit No.	Bit Function	Bit Description
			0021H: Internal Interface Error 0023H: Invalid Message 0024H: UPI Protocol Error 0025H: UPI Interleave Error 0028H: Internal Interface Error 0029H: Link integrity or initialization error 0029H: Link integrity or initialization error 002AH: Power Management Transition Error - Timeout 002BH: Power Management Transition Error - Timeout 002CH: Power Management Transition Error - Timeout 002DH: Power Management Transition Error - Timeout 0030H: Received poison when poison is disabled 0031H: Internal Interface Error 0200H: Link integrity or initialization error 0203H: Link degraded 0204H: Power Management Transition Error - Timeout 0205H: Power Management Transition Error - Timeout 0206H: Power Management Transition Error - Timeout 0207H: Power Management Transition Error - Timeout
Status Register Validity Indicators[1]	63:32		

NOTES:

1. These fields are architecturally defined. Refer to Chapter 18, "Machine-Check Architecture," of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B, for more information.