Flexible Return and Event Delivery (FRED)

Specification

June 2025

Revision 9.0

Document Number: 346446-009US

Notices & Disclaimers

This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

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

No product or component can be absolutely secure.

Your costs and results may vary.

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

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

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

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

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that you may publish an unmodified copy. You may create software implementations based on this document and in compliance with the foregoing that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Contents

Revision History

Revision Number	Description	Date
1.0	Initial Release of the document.	March 2021
2.0	Numerous updates across chapters include the following: • Event-related fields in the VMCS are renamed for consistency with FRED. • Changes are made to the event data saved for debug exceptions. • The treatment of event data by VMX transitions is updated and event-data fields are added to VMCS. • The treatment of RPL values by FRED event delivery and ERETS is updated. • FRED event delivery indicates whether event occurred in 64-bit mode. • FRED event delivery clears RFLAGS.TF. • The entry point for events incident to ring 0 is now 256 bytes after that for ring 3. • ERETU clears any earlier monitoring by UMONITOR. • ERETU provides the same instruction-ordering guarantees as SYSRET. • MOV to SS and POP SS do not block events when FRED transitions are enabled. • FRED event delivery of INT1, INT3, INT *n*, INTO, SYSCALL, SYSENTER save the instruction length on the stack. • VMX support for instruction length that already exists for INT1, INT3, INT *n*, and INTO is extended to SYSCALL and SYSENTER. • The addresses in the FRED RSP MSRs must be 64-byte aligned. WRMSR and VM entry will enforce this.	June 2021
3.0	Principal changes: • Previous versions of this specification augmented the existing token-based management of shadow-stack switching with an MSR-based verification mechanism. This version of the specification eliminates the token-based management (and the MSR-based verification) and replaces it with checks by ERETS and ERETU of a FRED SSP MSR. (In addition, ERETU can be executed only if CSL = 0 and FRED event delivery checks the canonicality of the new SSP value.) These changes are reflected in Section 3, Section 4.3, Section 5, Section 6, Section 9 (now Section 11), and Appendix A. This also resulted in deletion from Section 7 (now Section 8) of a subsection "WRMSR and XRSTORS" that specified details of the MSR-based verification mechanism. • When supervisor shadow stacks are enabled, FRED event delivery within ring 0 pushes the CSL on the shadow stack instead of the CS selector; ERETS compares the stack-level values popped from the regular and shadow stacks and causes #CP if they are not equal. • The format of the 64 bytes pushed on the regular stack by FRED event delivery have been reorganized. Corresponding changes have been made to ERETS and ERETU. FRED event delivery indicates whether the event was a nested exception. • Removed references to the IA32_FMASK MSR; FRED event delivery clears RFLAGS unconditionally. • Existing instructions cannot be used to enter compatibility mode in ring 0 (see Section 7.2, now Section 8.2). This limitation is also mentioned in Section 3 and resulted in corresponding updates to Section 9.5.2.3 and Section 10 (now Section 11.5.2.3 and Section 12). • Deleted a section on "Blocking Due to MOV to SS or POP SS." Enabling FRED transitions does not affect this blocking, which still occurs if those instructions are used. This change resulted in text being deleted from Section 9.5.3 and Section 9.6.1 (now Section 11.5.3 and Section 11.6.1). • LKGS can be executed only in 64-bit mode. With a null selector, the instruction zeroes the IA32_KERNEL_GS_BASE MSR. (Changes continue on next page)	May 2022

Revision Number	Description	Date
3.0	(Changes continued from previous page) Other changes: • Corrected an item in Section 4.3 that misidentified the bits in the IA32_FRED_CONFIG MSR that can be written to modify CSL. (The correct bits are 1:0.) • Clarified in Section 4.3 that the FRED SSP MSRs exist even on CPUs that do not support CET. • Added to Section 5.1.1 an indication that FRED event delivery will lead to a general-protection fault if the new RIP is value is not canonical. • Revised Section 5.1.2 and Section 5.1.3 with regard to the checking and loading of the new SSP value. • In Section 5.2.1, clarified the relations between #DB event data and DR6. • Added a sentence at the end of Section 5.3 to clarify that data breakpoints detected during FRED event delivery may be pending after completion of FRED event delivery. • Updated Section 6.2.1 to specify that ERETU always sets the RPL fields for CS and SS to 3. • Added to Section 8 (now Section 9) text clarifying why LKGS clears IA32_KERNEL_GS_BASE[63:32]. • Added to Section 10.3.4 (now Section 11.3.4) an explanation of why no new VMCS field is added for exiting-event data. • Corrected an item in Section 10.5 (now Section 11.5) that misidentified the bit in the CR4 field in the guest-state area of the VMCS that correspond to the bit in CR4 that enables FRED transitions. (The correct bit is 32.) • Added to an item in Section 10.5.1 (now Section 11.5.1) a sentence specifying that, if VM entry is injecting SYSCALL or SYSENTER, the VM-entry instruction-length field must be in the range 0–15. Added a footnote to that item, recalling that VM entry can use event type 7 for pending MTF VM exits (as well as SYSCALL and SYSENTER). • Updated the pseudocode in Appendix A.1 as follows: — To align with Section 5.2.1 and indicate that FRED event delivery saves the oldCSL as 0 when delivering an event that occurred in ring 3. — To order the check on the canonicality of the new RIP value to follow the determination of that value. — To clarify the determination of event stack level for privileged software exceptions (INT1). — To clarify that FRED event delivery establishes the pseudovariable newSSP only if supervisor shadow stacks are enabled. If newSSP is loaded from an MSR, it is immediately checked for canonicality and alignment. • In Appendix A.3, removed a check that SSP must be 8-byte aligned.	May 2022
4.0	• Added note at the end of Section 3 about shadow-stack token-management. • Updated Section 4.2 with details regarding the MOV to CR4 instruction. Called out that CR4.FRED can be 1 only in IA-32e mode and simplified some of the presentation as a result. • In the stack frame established by FRED event delivery (and used by ERETS and ERETU), moved the bits for blocking by STI, NMIs, and system calls from the augmented CS to the augmented SS. This is reflected in Section 5.2.1, Section 6.1.1, Section 6.1.3, Section 6.2.1, Section 6.2.3, Section 10.4.2 (now Section 11.4.2), Section 10.5.4 (now Section 11.5.4), and Appendix A. • Updated Section 5.1.2 and Appendix A.1 to remove canonicality checking on a value loaded from a FRED SSP MSR. • Updated Section 5.2.1 and Appendix A.1 to specify that FRED event delivery saves, as part of the augmented CS on the stack, the state of the supervisor indirect branch tracker. • Clarified in Section 5.2.1 that undefined event data are currently saved as zero but may be defined in the future. Added a footnote to clarify that STI blocking is saved as zero by event delivery resulting from execution of SYSCALL, SYSENTER, INT1, INT3, or INT *n*. • Updated Section 5.2.2 and Appendix A.1 to specify that FRED event delivery saves on the shadow stack the full 64-bit augmented CS (including additional bits in positions 63:16) that was saved on the regular stack. • Added to Section 6 a note specifying the error code used by #CP exceptions produced by ERETS and ERETU. • Revised the specification of ERETS and ERETU (in Section 6.1.1, Section 6.2.1, Appendix A.2, and Appendix A.3) to specify that the instructions add 8 to RSP before popping the return context from the stack. (Changes continue on next page)	May 2023

Revision Number	Description	Date
4.0	(Changes continued from previous page) • Revised the specification of ERETS and ERETU (in Section 6.1.1, Section 6.2.1, Appendix A.2, and Appendix A.3) to relax SSP canonicality checking by ERETS to be relative to the processor's maximum linear-address width and to eliminate the canonicality checking by ERETU on the value in IA32_PL3_SSP. • Updated Section 6.1.2 and Appendix A.2 to specify that, when supervisor shadow stacks are enabled, ERETS will fault if the 64-bit augmented CS popped from the shadow stack is not equal to the corresponding value popped from the regular stack. • Updated Section 6.1.1, Section 6.1.3, and Appendix A.2 to specify that ERETS may restore the indirect branch tracker from the augmented CS on the stack. • Revised the specification of ERETU (in Section 6.2.1 and Appendix A.3) to indicate that, when CS and SS match values derived from the IA32_STAR MSR, their selectors are loaded with the values popped from the stack. This change does not modify the operation of ERETU and was made only to simplify the presentation. • Corrected Section 6.2.1 on page 29 to indicate that case 2 of ERETU (return to a standard configuration for ring 3 in compatibility mode) sets CS.D to 1. • Updated Section 7.1 (now Section 8.1) to include CLRSSBSY and SETSSBSY. • Clarified in Section 8 (now Section 9) that LKGS can take an operand in a register or memory. • Clarified in Section 10.2 (now Section 11.2) that exceptions are reported as nested only if they occur during FRED event delivery (this reporting does not occur if FRED is not enabled). • Revised the treatment of the "deliver error code" bit in Section 10.5.1 (now Section 11.5.1). • Updated Section 10.5.2.1 (now Section 11.5.2.1) to specify that VM entry checks that the host CR4.FRED is 0 if "host address-space size" is 0. Similarly updated Section 10.5.2.2 (now Section 11.5.2.2) to specify that VM entry checks that the guest CR4.FRED is 0 if "IA-32e mode guest" is 0. • Updated Section 10.5.2.3 (now Section 11.5.2.3) to specify that VM entry will fail on an attempt to return to FRED-enabled ring 3 with STI blocking. • Updated Section 10.5.4 (now Section 11.5.4) to clarify the use of CPL and the saving of stack level, to provide details regarding the saving of the indirect branch tracker, and to clarify the use of an error code. • Updated Section 10.6.1 (now Section 11.6.1) to specify that a VM exit to legacy mode forces CR4.FRED to 0. • Updated Section 10.6.2 and Section 10.6.3 (now Section 11.6.2 and Section 11.6.3) to indicate that the treatment of the "error-code valid" bit is unchanged from what was done without FRED. • Added Section 10.7 (now Section 11.7). • Updated Section 11 (now Section 12) to specify that RSM will fail on an attempt to return such that CR4.FRED = 1 and IA32_EFER.LMA = 0, or to FRED-enabled ring 3 with STI blocking.	May 2023
5.0	• Updated the second footnote in Section 5.1.2 to remove the INT1, INT3, and INTO instructions that were previously listed.	May 2023
6.0	• Added documentation of NMI-source reporting (principally in Section 9, now Section 10), with some details in other sections). • Clarified in Section 4.2 the reasons that CR4.FRED cannot be set outside IA-32e mode. • Updated Section 5.2.1 to clarify how FRED event delivery determines the 8-bit vector that it saves on the stack. • Updated Section 5.2.1 (and Appendix A.1) to specify that FRED event delivery of INT1 or INT3 saves the state of the indirect branch tracker that existed prior to fetch of the instruction. • Added Section 5.4 to specify the error code and other details used by faults encountered during FRED event delivery. • Updated Section 6.1.1 to clarify that the #GP resulting from a non-canonical return RIP uses a zero error code. • Added to Section 6.1.1 (and Appendix A.2) specification of a check by ERETS to prevent establishment of STI blocking with RFLAGS.IF = 0. • Updated Section 6.1.2 to clarify that the #GP resulting from a non-canonical return SSP uses a zero error code. (Changes continue on next page)	December 2023

Revision Number	Description	Date
6.0	(Changes continued from previous page) • Updated Section 6.2.1 (and Appendix A.3) to clarify that ERETU to compatibility truncates the return RIP to a 32-bit value and checks that truncated RIP against the return CS limit. • Updated Section 7.4 (now Section 8.4) to indicate that SYSCALL is supported only in 64-bit mode (even when FRED transitions are enabled). Section 10.5.1 (now Section 11.5.1) notes that VM entry does not prevent injection of SYSCALL into a guest operating in compatibility mode. • Added Section 7.5 (now Section 8.5) to detail interactions with authenticated-code execution mode.	December 2023
6.1	• Updated Section 4.1 to clarify that any Intel processor that enumerates support for FRED transitions will also enumerate support for LKGS and NMI-source reporting. • Updated Section 9 (now Section 10) to clarify that processors that support FRED may also support a related feature called NMI-source reporting. • Change bars are left for revision 6.0 edits in this version due to the small number of changes.	December 2023
7.0	• Clarified in Section 5.1.3 and in Appendix A.1 that FRED event delivery makes both CS and SS usable regardless of the values being loaded into their selectors. • Updated Section 5.2.1 with details regarding the saving of bit 18 of the CS image on the stack (related to indirect-branch tracking). Deleted a related footnote in Section 10.5.4 (now Section 11.5.4) and updated the relevant portion of Appendix A.1 to refer to Section 5.2.1. • Removed from Section 6.1.1 and Appendix A.2 a check by ERETS related to blocking by STI and the value being loaded for RFLAGS.IF. • Updated Section 6.1.3 and Appendix A.2 to specify that ERETS will not establish blocking by STI if such blocking had been in effect prior to execution of ERETS or if RFLAGS.IF will be 0 following execution of ERETS. • Updated Section 6.2.1 and Appendix A.3 to indicate that ERETU does not fault based on bit 16 of the SS image on the stack. • Clarified in Section 6.2.1 and in Appendix A.3 that ERETU makes both CS and SS usable regardless of the values being loaded into their selectors. • Updated Section 6.2.3 and Appendix A.3 to indicate that ERETU to compatibility mode clears RSP[63:32]. • Clarified in Section 7.3 (now Section 8.3) that execution of INTO if RFLAGS.OF = 0 does not result in FRED event delivery. • Added to Section 10.5.4 (now Section 11.5.4) a paragraph emphasizing that FRED event delivery drops any pending debug exceptions for events injected with VM entry.	March 2024
8.0	There was no revision 8.0. Changes in revision 9.0 (below) are relative to revision 7.0.	
9.0	• Updated various sections to refer to “MSR-access instructions” and “MSR-write instructions” (or “a write to this MSR”) rather than specifically to RDMSR and WRMSR. This ensures that the relevant material applies also to RDMSRLIST, WRMSRLIST, and WRMSRNS. • Added footnote 1 on page 9. • Added to Section 2 text explaining notation such as CS, SS, DS, ES, FS, GS, DPL, and RPL. • Removed from Section 4.1 a statement that any processor enumerating support for FRED transitions would also enumerate support for NMI-source reporting. • Updated Section 5.3 and Appendix A.1 to indicate that, in the presence of supervisor indirect branch tracking, FRED event delivery does not enter the WAIT_FOR_ENDBRANCH state there is no need for event handlers to begin with ENDBR64. • Added new Section 7 to document interactions with Intel Processor Trace. • Added details to Section 8.2 (formerly Section 7.2) to clarify the priority of new exceptions on some existing instructions. • Updated Section 8.5 (formerly Section 7.5) to clarify that GETSEC[EXITAC] updates CR4 only when returning from GETSEC[ENTERACCS].	June 2025

1 Introduction

This document is a specification of the architecture of a new feature called **flexible return and event delivery** (**FRED**).

The FRED architecture defines simple new transitions that change privilege level (**ring transitions**). The FRED architecture was designed with the following goals:

- Improve overall performance and response time by replacing event delivery through the interrupt descriptor table (**IDT event delivery**) and event return by the IRET instruction with lower latency transitions.[1]
- Improve software robustness by ensuring that event delivery establishes the full supervisor context and that event return establishes the full user context.

The new transitions defined by the FRED architecture are **FRED event delivery** and, for returning from events, two **FRED return instructions**. FRED event delivery can effect a transition from ring 3 to ring 0, but it is used also to deliver events incident to ring 0. One FRED instruction (ERETU) effects a return from ring 0 to ring 3, while the other (ERETS) returns while remaining in ring 0. Collectively, FRED event delivery and the FRED return instructions are **FRED transitions**.

In addition to these transitions, the FRED architecture defines a new instruction (LKGS) for managing the state of the GS segment register. The LKGS instruction can be used by 64-bit operating systems that do not use the new FRED transitions.

The FRED architecture includes a feature called **NMI-source reporting**. FRED event delivery of an NMI reports to software information about the NMI being delivered.

The following is the organization of this specification:

- Section 2 describes the existing behavior of ring transitions in the Intel® 64 architecture.
- Section 3 gives an overview of the FRED architecture.
- Section 4 gives details on enumeration and configuration of the FRED architecture, including the new state defined for it.
- Section 5 explains FRED event delivery.
- Section 6 presents the FRED return instructions.
- Section 8 describes how enabling FRED transitions changes the operation of existing instructions (including those use for system call and return).
- Section 9 presents a new instruction (LKGS) for managing the state of the GS segment register.
- Section 10 describes NMI-source reporting, a feature that allows FRED event delivery to provide additional information about non-maskable interrupts.
- Section 11 provides details of virtualization support (VMX) for the FRED architecture.

1. "Event delivery" here refers to the changes in control and (in some cases) privilege that invoke software handlers for interrupts and exceptions. Prior to FRED transitions, event delivery was managed by a software-configured interrupt-descriptor table (IDT). In addition to interrupts and exceptions, FRED event delivery applies to executions of SYSCALL and SYSENTER.

- Section 12 identifies changes to the delivery of system-management interrupts and execution of the RSM instruction.
- Appendix A presents details of the operation of FRED event delivery and the FRED return instructions.
- Appendix B provides a supplementary discussion of event stack levels.

Note: This document uses the term **canonical** in two different ways. A linear address is **canonical relative to the processor's maximum linear-address width** if bits 63:M–1 of the address are identical, where M is processor's maximum linear-address width (as enumerated in CPUID.80000008H:EAX[bits 15:8]). A linear address is **canonical relative to the current paging mode** if bits 63:L–1 of the address are identical, where L is 48 if 4-level paging is enabled and 57 if 5-level paging is enabled.

2 Ring Transitions and the CS, SS, and GS Segments

The Intel® 64 architecture's protection architecture is based on use of a 2-bit privilege level (0–3). Operation at a particular privilege level is informally called a **ring**; thus, ring 0 (the most privileged ring) refers to operation while the current privilege level (CPL) is 0, while ring 3 (the least privileged ring) refers to operation while the CPL is 3.

The Intel 64 architecture defines a number of control-flow transitions that change the CPL. Known informally as **ring transitions**, there are two principal types:

- Transitions that increase privilege (by decreasing the CPL). These include transitions using interrupt and trap gates in the interrupt descriptor table (IDT), executions of the far CALL instruction that access call gates, and executions of the SYSCALL and SYSENTER instructions.
- Transitions that decrease privilege (by increasing the CPL). These include executions of the instructions IRET, far RET, SYSEXIT, and SYSRET.

In IA-32e mode, the CPL is visible to software in the RPL field (requested privilege level) of the code-segment (CS) selector (bits 1:0). In addition, the DPL field (descriptor privilege level) in the descriptor cache for the stack segment (SS) also contains the CPL, although this is not directly visible to software. Because the CPL is manifest in the CS and SS segment registers, ring transitions always modify the CS and SS segment registers.

Besides CS and SS, the Intel® 64 architecture use four data segments: DS, ES, FS, and GS. GS is another segment that software manages at the time of ring transitions. This is because 64-bit operating systems use the GS segment to support **thread-local storage** (TLS): the GS base address identifies the location of the TLS. User and supervisor software use the TLS at different addresses, so the base address of the GS segment will differ depending on the CPL.

Unlike CS and SS, GS is not modified by existing ring transitions. This means that, after a transition to ring 0, the GS base address will still reference the user TLS. For this reason, supervisor software must update the GS base address before it can access its own TLS. Similarly, it must switch the GS base address back to the user value before returning to user software. The SWAPGS instruction supports efficient updates of the GS base address.

3 Overview of the FRED Architecture

The following items summarize the principle elements of the FRED architecture:

- **FRED event delivery.** Any event that would normally cause IDT event delivery (e.g., an interrupt or exception) will instead establish the new context with a new process that does not access existing data structures such as the IDT. The SYSCALL and SYSENTER instructions will also use FRED event delivery in place of their existing operation.
- **FRED return instructions.** Two new return instructions, ERETS and ERETU, effect returns from event handling. ERETS (event return to supervisor) is designed to return from events incident to ring 0 and does not change CPL; ERETU (event return to user) returns to application software operating in ring 3.
- **GS segment management.** FRED transitions that change the CPL (FRED event delivery and ERETU) update the GS base address in a manner similar to the SWAPGS instruction. In addition, the LKGS instruction allows system software to manage the GS segment more flexibly.

When FRED transitions are enabled, they are the only ring transitions possible. Since FRED event delivery is always to ring 0 and ERETU returns only to ring 3, it is not possible to enter ring 1 or ring 2 while FRED transitions are enabled. Enabling FRED transitions also prevents entry to compatibility mode in ring 0.

The FRED architecture is defined for use by a 64-bit operating system. Except where noted, changes to processor behavior and new instructions apply only in IA-32e mode (IA32_EFER.LMA = 1) and not in legacy protected mode (or real-address mode). The new processor state defined by the FRED architecture is accessible with MSR-access instructions (e.g., RDMSR, WRMSR) regardless of mode.

The FRED architecture introduces the concept of a **stack level**. The **current stack level** (CSL) is a value in the range 0–3 that a logical processor tracks when CPL = 0. FRED event delivery determines the stack level associated with the event being delivered and, if it is greater than the CSL (or if CPL had been 3), loads the stack pointer (RSP) from a new **FRED RSP MSR** associated with the event's stack level. (If supervisor shadow stacks are enabled, the stack level applies also to the shadow-stack pointer, SSP, which would be loaded from a **FRED SSP MSR**.) The FRED return instruction ERETS restores the old stack level. (Details of the new MSRs are given in Section 4.3.)

The existing shadow-stack architecture includes a token-management mechanism to ensure shadow-stack integrity when switching shadow stacks. This mechanism uses locked read-modify-write operations that may affect worst-case performance adversely. FRED transitions do not use this token-management mechanism and instead protect shadow-stack integrity by having the FRED return instructions check values in the FRED SSP MSRs.

Note: Because FRED transitions do not use the shadow-stack token-management mechanism, FRED event delivery can never cause a fault or a VM exit after setting the busy bit in a shadow-stack token (unlike IDT event delivery). For that reason, an OS

that enables FRED transitions need not consult CPUID.(EAX=07H,ECX=1H):EDX[bit 18] before enabling supervisor shadow stacks.

4 Enumeration, Enabling, and Configuration

This section describes the enumeration, enabling, and configuration mechanisms for the FRED architecture.

4.1 Enumeration

The FRED architecture includes two related but independent elements: the new FRED transitions and the LKGS instruction for management of the GS segment register. Because 64-bit operating systems may benefit from the LKGS instruction without using FRED, the two elements are enumerated independently.

CPUID.(EAX=7,ECX=1):EAX.FRED[bit 17] is a new feature flag that enumerates support for the new FRED transitions. It also enumerates support for the new architectural state (MSRs) defined by FRED.

CPUID.(EAX=7,ECX=1):EAX.LKGS[bit 18] is another new CPUID feature flag that enumerates support for the LKGS instruction.

CPUID.(EAX=7,ECX=1):EAX.NMI_SRC[bit 20] is another new CPUID feature flag that enumerates support for NMI-source reporting.

Any Intel® processor that enumerates support for FRED transitions will also enumerate support for LKGS.

4.2 Enabling in CR4

Software enables FRED transitions by setting CR4[32] (henceforth CR4.FRED). Specifically, setting CR4.FRED enables FRED event delivery (Section 5) as well as the FRED return instructions (Section 6), although those instructions can be used only in 64-bit mode (when CS.L = 1).

Execution of the MOV to CR4 instruction outside 64-bit mode clears CR4[63:32]. For that reason, software must first enter 64-bit mode before using MOV to CR4 to set CR4.FRED. The architecture ensures that CR4.FRED can be 1 only if the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1):

- As just noted, MOV to CR4 can set CR4.FRED only in IA-32e mode.
- Software can leave IA-32e mode only by executing MOV to CR0 to clear CR0.PG, and this can be done only in compatibility mode while CPL = 0. (Attempts to clear CR0.PG in 64-bit mode will fault.)
- If CR4.FRED = 1, it is impossible to enter compatibility mode while CPL = 0 (see Section 8.2).
- RSM and VMX transitions cannot enter a state in which CR4.FRED = 1 outside IA-32e mode.

The value of CR4.FRED does **not** affect the accessibility of the new MSRs (Section 4.3) by MSR-access instructions (e.g., RDMSR, WRMSR), nor does it effect the operation of the new LKGS instruction (Section 9).

Enabling FRED transitions changes the behavior of various existing instructions. See Section 8.

4.3 New MSRs for Configuration of FRED Transitions

FRED transitions are controlled by the following new MSRs:

- IA32_FRED_CONFIG (MSR index 1D4H). This MSR is organized as follows:
  - Bits 1:0 of this MSR contain the **current stack level** (CSL). This 2-bit value is manipulated and used by FRED event delivery (Section 5) and the FRED return instructions (Section 6).

    Software can modify the CSL with MSR-write instructions (e.g., WRMSR), but this is not an expected usage. It is recommended that, when using such an instruction to update IA32_FRED_CONFIG, software always write the existing CSL into bits 1:0 of the MSR (unless it specifically desires to change the CSL).
  - Bit 2 is reserved.
  - Bit 3 indicates, if set, that, if FRED event delivery is not changing stacks, it should decrement the shadow-stack pointer (SSP) by 8.
  - Bits 5:4 are reserved.
  - Bits 8:6 identify the amount (measured in 64-byte cache lines) by which FRED event delivery decrements the regular stack pointer (RSP) when not changing stacks.
  - Bits 10:9 identify the stack level that is used for maskable interrupts that are delivered in ring 0. See Section 5.1.2.
  - Bit 11 is reserved.
  - Bits 63:12 contain the upper bits of the linear address of a page in memory containing event handlers. FRED event delivery will load RIP to refer to an entry point on this page. See Section 5.1.1.

  A write to this MSR (e.g., using WRMSR) causes a general-protection exception (#GP) if its source operand sets any reserved bits or if it is not canonical relative to the processor's maximum linear-address width.
- IA32_FRED_RSP0, IA32_FRED_RSP1, IA32_FRED_RSP2, and IA32_FRED_RSP3 (MSR indices 1CCH–1CFH). These are the **FRED RSP MSRs**.

  If FRED event delivery causes a transition from ring 3 or a change to the CSL, it loads RSP from the FRED RSP MSR corresponding to the new stack level. See Section 5.1.2.

  A write to any of these MSRs (e.g., using WRMSR) causes a general-protection exception (#GP) if its source operand is not 64-byte aligned (bits 5:0 are not all zero) or if it is not canonical relative to the processor's maximum linear-address width.

Note: The numbers 0–3 in the MSR names refer to the corresponding stack level **and not to privilege level**.

- IA32_FRED_STKLVLS (MSR index 1D0H). This MSR is interpreted as an array of 32 2-bit values, one for each vector in the range 0–31. For an exception with vector *v* (or for a non-maskable interrupt, which always has vector *v* = 2) that occurs in

ring 0, FRED event delivery ensures that the new stack level is at least the value of IA32_FRED_STKLVLS[2*v*+1:2*v*]. See Section 5.1.2.

- IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3 (MSR indices 1D1H–1D3H). Together with the existing MSR IA32_PL0_SSP (MSR index 6A4H), these are the **FRED SSP MSRs**. References in this document to "IA32_FRED_SSP0" are to the IA32_PL0_SSP MSR.

  If supervisor shadow stacks are enabled and FRED event delivery causes a transition from ring 3 or a change to the CSL, it loads SSP from the FRED SSP MSR corresponding to the new stack level. See Section 5.1.2.

  The address in each of IA32_FRED_SSP*i* ($1 \le i \le 3$) must be 8-byte aligned, so bits 2:0 are reserved. (For legacy reasons, IA32_PL0_SSP — also called IA32_FRED_SSP0 — is allowed to be 4-byte aligned, so only bits 1:0 are reserved.)

  A write to any of these MSRs (e.g., using WRMSR) will cause a general-protection exception (#GP) if its source operand sets any reserved bits or if it is not canonical relative to the processor's maximum linear-address width. (The same is already true for executions of XRSTORS and MSR-write instructions that would load IA32_PL0_SSP.)

Note: As with the FRED RSP MSRs, the numbers 0–3 in the MSR names refer to the corresponding stack level **and not to privilege level**.

Note: The FRED SSP MSRs are supported by any processor that enumerates CPUID.(EAX=7,ECX=1):EAX.FRED[bit 17] as 1. If such a processor does not support CET, FRED transitions will not use the MSRs (because shadow stacks are not enabled), but the MSRs would still be accessible using MSR-access instructions (e.g., RDMSR, WRMSR).

The RESET state of each of the new MSRs is zero. INIT does not change the value of the new MSRs.

4.4 FRED Use of Existing MSRs

In addition to the new MSRs identified in Section 4.3, FRED transitions use several existing MSRs as specified in the following items:

- IA32_STAR.
  - — Existing use: this MSR is used by the existing operation of SYSCALL and SYSRET.
  - — FRED use: FRED event delivery loads the CS and SS selectors with values derived from IA32_STAR[47:32] (see Section 5.3). ERETU uses the value of IA32_STAR[63:48] to determine how to load the CS and SS registers (see Section 6.2.1).

    It is expected that operating systems will set IA32_STAR[33:32] to 00B and IA32_STAR[49:48] to 11B.
- IA32_KERNEL_GS_BASE.
  - — Existing use: the SWAPGS instruction exchanges the value of this MSR with the base address of the GS segment register.
  - — FRED use: executions of ERETU perform this same swapping operation, as does FRED event delivery of events that arrive in ring 3.
- IA32_PL0_SSP.

— Existing use: the shadow-stack feature of control-flow enforcement technology (CET) typically loads SSP from this MSR when entering ring 0.

— FRED use: FRED transitions use this MSR as IA32_FRED_SSP0 (Section 4.3).

5 FRED Event Delivery

When FRED transitions are enabled (CR4.FRED = 1), IDT event delivery of exceptions and interrupts is replaced with **FRED event delivery**. In addition, the existing operation of SYSCALL and SYSENTER is also replaced with FRED event delivery.

These changes do not affect the processor's handling of exceptions and interrupts prior to event delivery. For example, any determination that an event causes a VM exit or is converted into a double fault occurs normally. Similarly, page faults and debug exceptions update CR2 and DR6, respectively, in the normal way.

The principal functionality of FRED event delivery is to establish a new context, that of the event handler in ring 0, while saving the old context for a subsequent return. Some parts of the new context have fixed values, while others depend on the old context, the nature of the event being delivered, and software configuration. Section 5.1 describes how FRED event delivery determines and establishes the new context. Section 5.2 specifies how FRED event delivery saves elements of the old context on the stack (and, when enabled, the shadow stack). Section 5.3 then describes how additional state is updated. Section 5.4 gives details about faults encountered during FRED event delivery.

Appendix A.1 presents the detailed flow of the operation of FRED event delivery.

5.1 Determining and Establishing the New Context

The context of an event handler invoked by FRED event delivery includes the CS and SS segment registers, the instruction pointer (RIP), the flags register (RFLAGS), the stack pointer (RSP), and the base address of the GS segment (GS.base). The context also includes the shadow-stack pointer (SSP) if supervisor shadow stacks are enabled.

FRED event delivery establishes this context by loading these registers appropriately. It determines the values to be loaded into RIP, RSP, GS.base, and SSP based on the old context, the nature of the event being delivered, and software configuration. RFLAGS, CS, and SS are loaded with fixed values.

5.1.1 Determining the New RIP Value

FRED event delivery uses two entry points, depending on the CPL at the time the event occurred. This allows an event handler to identify the appropriate return instruction (ERETU or ERETS).

Specifically, the new RIP value that FRED event delivery establishes is IA32_FRED_CONFIG & ~FFFH for events that occur in ring 3 and (IA32_FRED_CONFIG & ~FFFH) + 256 for events that occur in ring 0.

If the new RIP value is not canonical relative to the current paging mode, FRED event delivery causes a general-protection fault (#GP).[1]

1. If this #GP does not cause a VM exit, FRED event delivery of the #GP will cause a second #GP, which will be converted to a double fault (#DF). If the #DF does not cause a VM exit, FRED event delivery of the #DF will cause a third #GP, resulting in a triple fault, which will either cause a VM exit or take the logical processor to shutdown.

5.1.2 Determining the New Values for Stack Level, RSP, and SSP

FRED transitions support four different stacks for use in ring 0. A logical processor identifies the stack currently in use with a 2-bit value called the **current stack level** (CSL).

FRED event delivery first determines the event's stack level and then uses that to determine whether the CSL should change. An event's stack level is based on the CPL, the nature and type of the event, the event's vector (for some event types), and MSRs configured by system software:

- If the event occurred in ring 3, was not a nested exception encountered during event delivery, and was not a double fault (#DF), the event's stack level is 0.
- If the event occurred in ring 0, was a nested exception encountered during event delivery, or was a #DF, the following items apply:
  - If the event is a maskable interrupt, the event's stack level is IA32_FRED_CONFIG[10:9].
  - If the event is an exception or a non-maskable interrupt (NMI), the event's stack level is IA32_FRED_STKLVLS[2*v*+1:2*v*], where *v* is the event's vector (in the range 0–31).[1]
  - The stack level of all other events is 0.[2]

If the event occurred in ring 3, the new stack level is the event's stack level; otherwise, the new stack level is the maximum of the CSL and the event's stack level. (This implies that the CSL is always 0 following FRED event delivery of an event that occurred in ring 3, unless the event was a nested exception or a #DF.)

After determining the new stack level, FRED event delivery identifies the new RSP value as follows:

- If either the CPL or the CSL is changing, the new RSP value will be that of the FRED RSP MSR corresponding to the new stack level.
- Otherwise, the new RSP value will be the current RSP value decremented by IA32_FRED_CONFIG & 1C0H (the multiple of 64 in bits 8:6 of that MSR) and then aligned to a 64-byte boundary (by clearing RSP[5:0]).

If supervisor shadow stacks are enabled, the following items explain how the new SSP value is determined:

- If either the CPL or the CSL is changing, the new SSP value will be that of the FRED SSP MSR corresponding to the new stack level. If that new SSP value is not 8-byte aligned, FRED event delivery causes a general-protection fault (#GP).[3]
- Otherwise, the new SSP value will be the current SSP value decremented by IA32_FRED_CONFIG & 8 (setting bit 3 of that MSR indicates that SSP should be decremented by 8).

1. Exceptions include the events generated by the INT1, INT3, and INTO instructions, but not those generated by the INT *n* instruction (for any value of *n*).
2. The other events are those generated by the instructions INT *n* (for any value of *n*), SYSCALL, and SYSENTER.
3. The value of each of IA32_FRED_SSP*i* ($1 \le i \le 3$) is always 8-byte aligned, so the check specified above is necessary only if SSP is being loaded from the IA32_PL0_SSP MSR (IA32_FRED_SSP0).

5.1.3 Establishing the New Context

After determining the details of the new context, FRED event delivery loads registers to establish that context.

For events that occur in ring 3, FRED event delivery updates the CS, SS, and GS segments as well as the IA32_KERNEL_GS_BASE MSR:

- CS:
  - The selector is set to IA32_STAR[47:32] AND FFFCH (this forces CS.RPL to 0).
  - The base address is set to 0. The limit is set to FFFFFH and the G bit is set to 1.
  - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
  - The DPL is set to 0, the P and L bits are each set to 1, and the D bit is set to 0.
  - CS is made usable regardless of the value of the selector.
- SS:
  - The selector is set to the new value of the CS selector (above) plus 8 (thus, SS.RPL is also 0).
  - The base address is set to 0. The limit is set to FFFFFH and the G bit is set to 1.
  - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
  - The DPL is set to 0, and the P and B bits are each set to 1.
  - SS is made usable regardless of the value of the selector.
- GS: FRED event delivery swaps the value of the GS base address and that of the IA32_KERNEL_GS_BASE MSR.

(The above updates occur without any access to the GDT or an LDT.)

For events that occur in ring 0, FRED event delivery does not modify CS, SS, or GS.

After updating the segment registers, FRED event delivery loads RIP, RSP, and CSL with the values determined in Section 5.1.1 and Section 5.1.2. If supervisor shadow stacks are enabled, SSP is loaded with the value determined in Section 5.1.2. RFLAGS is loaded with the value 2 (all bits clear except position 1, which has no function but is always set).

If FRED event delivery incurs a nested exception or VM exit after this point, the processor restores the values that were in these registers before FRED event delivery commenced and only then delivers the nested exception or VM exit.

5.2 Saving Information About the Event and the Old Context

Like IDT event delivery, FRED event delivery saves information about the old context on the stack of the event handler. This information includes the previous context saved in much the same format as that following IDT event delivery, additional information about the event being delivered, and auxiliary information that will guide a subsequent return instruction.

If supervisor shadow stacks are enabled, FRED event delivery also saves information on the event handler’s shadow stack.

The memory accesses used to store information on the stacks are performed with supervisor privilege. Any of these memory accesses may result in nested exception. The nested exception (or a double fault to which it may be converted) would then be delivered with FRED event delivery.

5.2.1 Saving Information on the Regular Stack

FRED event delivery saves 64 bytes of information on the new regular stack. The following items provide details:

- The first 8 bytes pushed (bytes 63:56 of the 64-byte stack frame) are not currently defined and will be zero until they are.
- The next 8 bytes pushed (bytes 55:48) contain **event data** and are defined as follows:
  - — If the event being delivered is a page fault (#PF), the event data is the faulting linear address (this is the same value that the #PF loads into CR2).[1]
  - — If the event being delivered is a debug exception (#DB), the event data identifies the nature of the debug exception:
    - Bits 3:0 are B3–B0. When set, each of these bits indicates that the corresponding breakpoint condition was met (and may be set even if its corresponding enabling bit in DR7 is not set).
    - Bits 10:4 are not currently defined and will be zero until they are.
    - Bit 11 is BLD. When set, this bit indicates that the cause of the debug exception was acquisition of a bus lock (because IA32_DEBUGCTL[2] = 1).
    - Bit 12 is not currently defined and will be zero until it is.
    - Bit 13 is BD. When set, this bit indicates that the cause of the debug exception was "debug register access detected."
    - Bit 14 is BS. When set, this bit indicates that the cause of the debug exception was the execution of a single instruction (because execution of that instruction commenced with RFLAGS.TF = 1).[2]
    - Bit 15 is not currently defined and will be zero until it is.
    - Bit 16 is RTM. When set, this bit indicates that a debug exception (#DB) or a breakpoint exception (#BP) occurred inside an RTM region while advanced debugging of RTM transactional regions was enabled.
    - Bits 63:17 are not currently defined and will be zero until they are.

    The event data is not exactly the same as that which will be in DR6 following delivery of the #DB. The polarity of bit 11 and bit 16 is inverted in DR6; in addition, delivery of #DB may leave unmodified in DR6 some bits in positions that are clear in the event data and that were already set in DR6.
  - — If the event being delivered is a device-not-available exception (#NM) caused by extended feature disable, the event data indicates the cause of the exception and is the same as that loaded into the IA32_XFD_ERR MSR. If the #NM was not caused by extended feature disable, the event data is zero.
  - — If the event being delivered is a non-maskable interrupt (NMI) and the processor supports NMI-source reporting, bits 15:0 of the event data contain the NMI-source bitmap associated with the NMI; bits 63:16 are zero. If the

1. If the #PF occurred during execution of an instruction in enclave mode (but not during delivery of an event incident to enclave mode), bits 11:0 of the event data are cleared.
2. If in addition IA32_DEBUGCTL.BTF was 1 when the instruction commenced, a debug exception occurs only if the instruction was a taken branch.

processor does not support NMI-source reporting, the event data is zero. See Section 10 for information about NMI-source reporting.

— For any other event, the event data are not currently defined and will be zero until they are.

— The event data is determined differently when FRED event delivery is used for an event injected by VM entry. See Section 11.5.4.

- The next 40 bytes pushed (bytes 47:8) are the **return state** and have generally the same format as that used by IDT event delivery. That format stores the 16-bit selectors for CS and SS in 64-bit fields. FRED event delivery augments those selector values with additional information saved in the upper 48 bits of those fields.

  The following items detail the format of the return state on the stack from bottom (highest address) to top:

  — The augmented SS of the previous context, 64 bits formatted as follows:

    - Bits 15:0 contain the SS selector.
    - Bit 16 is set to 1 for FRED event delivery of a hardware exception if interrupt blocking by STI was in effect at the time the exception occurred[1] and is otherwise cleared to 0.[2]
    - Bit 17 is set to 1 for FRED event delivery of SYSCALL, SYSENTER, or INT *n* (for any value of *n*) and is otherwise cleared to 0.
    - Bit 18 is set to 1 if the event being delivered is a non-maskable interrupt (NMI) and is otherwise cleared to 0.
    - Bits 31:19 are not currently defined and will be zero until they are.
    - Bits 63:32 contain **event information**. This provides software with additional information about the event. The FRED return instructions ignore these bits. The event information is formatted as follows:

      — Bits 39:32 contain the event's vector. The vector depends on the type of event:

        - For an external interrupt, the vector is provided by the local APIC.
        - For a non-maskable interrupt (NMI), the vector is 2.
        - For a hardware exception, the vector is determined by the exception.
        - For a software interrupt (INT *n*), the vector comes from the instruction's opcode (value *n*).
        - For a software exception, the vector is determined by the instruction opcode: INT1 has vector 1; INT3 has vector 3; INTO has vector 4.
        - For SYSCALL and SYSENTER (which use FRED event delivery but not IDT event delivery), vectors 1 and 2 are used, respectively.

      — Bits 47:40 are not currently defined and will be zero until they are.

      — Bit 51:48 encode the event type as follows: 0 = external interrupt; 2 = non-maskable interrupt; 3 = hardware exception (e.g., page fault); 4 = software interrupt (INT *n*); 5 = privileged software exception (INT1); 6 = software exception (INT3 or INTO); and 7 = other event (used for SYSCALL and SYSENTER). Other values are not used.[3]

1. Execution of STI with RFLAGS.IF = 0 blocks interrupts on the instruction boundary following its execution.
2. Delivery of SYSCALL, SYSENTER, INT1, INT3, or INT *n* (for any value of *n*) will always clear this bit, as will delivery of any exception encountered during event delivery for one of those instructions.

— Bits 55:52 are not currently defined and will be zero until they are.

— Bit 56 is set to 1 to indicate that the event was incident to enclave execution. Specifically, it is set in any of the following cases:

- The event occurred while the logical processor was in enclave mode.
- The event was injected by VM entry (see Section 11.5.4) and the guest interruptibility-state field in the VMCS indicates an "enclave interruption" (bit 4 of the field is 1).
- The event was a debug exception that was pending following a VM entry for which guest interruptibility indicates an "enclave interruption" (see above).
- The event was a debug exception that was pending following an execution of RSM for which SMRAM indicates an "enclave interruption."
- The event was an exception that was encountered during delivery of any of the events above.

Otherwise, the bit is cleared to 0.

— Bit 57 is set to 1 if the logical processor had been in 64-bit mode when the event occurred. (A value of 0 indicates an event incident to compatibility mode.)

— Bit 58 is set to 1 if the event is nested exception encountered during FRED event delivery of another event. This bit is not set if the event is double fault (#DF).

— Bit 59 is not currently defined and will be zero until it is.

— Bits 63:60 contain the length of the instruction causing the event if the event type is software interrupt (INT *n*), privileged software exception (INT1), software exception (INT3 or INTO), or other event (when used for SYSCALL or SYSENTER). (The length used is determined differently when FRED event delivery is used for an event of these types injected by VM entry. See Section 11.5.4.) For other event types, these bits are cleared to zero.

— RSP of the previous context (64 bits).

— RFLAGS of the previous context (64 bits).

Bit 16 of the RFLAGS field (corresponding to the RF bit) is saved as 1 when delivering events that do the same for IDT event delivery. These are faults other than instruction breakpoints as well as any traps or interrupts delivered following partial execution of an instruction (e.g., between iterations of a REP-prefixed string instruction). Delivery of other events save in bit 16 the value that RFLAGS.RF had at the time the event occurred.

— The augmented CS of the previous context. The FRED return instructions use the 64 bits of this field. It is formatted as follows:

- Bits 15:0 contain the CS selector.
- Bits 17:16:
  — For delivery of events that occur in ring 0, these bits report the current stack level (CSL) at the time the event occurred.
  — For delivery of events that occur in ring 3, these bits are cleared to 0.

3. The event types used by FRED event delivery are the same as those already defined for VMX transitions. For SYSCALL or SYSENTER, FRED event delivery reports event type 7 (other event). For VM entry, that event type is used with vector 0 to indicate a pending MTF VM exit. That is why FRED event delivery uses vectors 1 and 2 to indicate SYSCALL and SYSENTER, respectively.

- Bit 18 is set to 1 if the event occurred in ring 0 and the indirect branch tracker was in the WAIT_FOR_ENDBRANCH state. Specifically, the bit is set if and only if the following are all true: CPL = 0; CR4.CET = 1; IA32_S_CET.ENDBR_EN = 1; and IA32_S_CET.TRACKER = 1.

  The following items document the implications of how bit 18 is saved based on the existing operation of indirect branch track (providing details regarding delivery of specific events occurring in ring 0):

  — Delivery of an event occurring at an instruction boundary saves bit 18 with the setting of the tracker at that instruction boundary:

    - For an event that takes priority over a missing-ENDBRANCH #CP exceptions, this value may be 0 or 1, depending on the state of the tracker. These events include certain machine-check exceptions, trap-class debug exceptions, non-maskable interrupts, external interrupts, debug exceptions due to instruction breakpoints, and faults from fetching the next instruction. They may also include faults from decoding the next instruction.
    - For a missing-ENDBRANCH #CP exception, this value is 1.
    - For a fault from decoding the next instruction that is prioritized below missing-ENDBRANCH #CP exceptions, this value is 0.

  — Delivery of INT1 or INT3 saves bit 18 with the tracker's setting at the time INT1 or INT3 was fetched. (Decode of these instructions does not clear the tracker.)

  — Delivery of INT *n* (for any value of *n*), SYSCALL, or SYSENTER will save bit 18 with value 0. (Decode of these instructions clears the tracker.)

  — Delivery of an exception encountered during FRED event delivery (including that for INT1, INT3, INT *n*, SYSCALL, or SYSENTER) will save bit 18 with the value that would have been saved by delivery of the original event.

  — Delivery of an exception encountered during execution of any instruction other than INT1 or INT3 will save bit 18 with value 0. (Decode of any instruction other than INT1 or INT3 clears the tracker.)

- Bits 63:19 are not currently defined and will be zero until they are.

— RIP of the previous context (64 bits).

If the event type is software interrupt (INT *n*), privileged software exception (INT1), software exception (INT3 or INTO), or other event (when used for SYSCALL or SYSENTER); the RIP value saved references the instruction after the one that caused the event being delivered. (If delivery of such an event encounters an exception, the RIP value saved by delivery of the exception will instead reference the instruction that caused the original event.)

- The last 8 bytes pushed (bytes 7:0) contain error code (defined only for certain exceptions; zero if none is defined) in bits 15:0. Bits 63:16 are not currently defined and will be zero until they are.

5.2.2 Saving Information on the Shadow Stack

If supervisor shadow stacks are enabled and the event being delivered occurred in ring 0, FRED event delivery saves information on the new shadow stack. The following items provide details:

- FRED event delivery writes four bytes of zeroes to the linear address SSP – 4 and then clears SSP[2:0]. This has the following effect:
  - If the value of SSP had not been 8-byte aligned (it is necessarily 4-byte aligned), 4 bytes of zeroes are stored (as padding) and subsequent shadow-stack pushes will immediately follow those 4 bytes of zeroes.
  - If the value of SSP had been 8-byte aligned, 4 bytes of zeroes are stored but will be overwritten by subsequent shadow-stack pushes.
- FRED event delivery then pushes the following values on the new shadow stack:
  - The same 64-bit augmented CS that was pushed on the regular stack (see Section 5.2.1), including any bits defined in positions 63:16.
  - The old linear instruction pointer.
  - The old SSP on the shadow stack.

  Each of these values is pushed in a separate 8-byte field on the shadow stack.

The write of zeroes and the pushes are all performed as shadow-stack accesses with supervisor privilege.

5.3 Loading Additional State

After saving the old context and other information, FRED event delivery completes operation by loading additional registers and updating other processor state.

If the event occurred in ring 3 and user shadow stacks are enabled, the IA32_PL3_SSP MSR is loaded with the old value of SSP. The value loaded into the MSR is adjusted so that bits 63:N get the value of bit N–1, where N is the processor's maximum linear-address width.

If supervisor indirect branch tracking is enabled, the IA32_S_CET MSR is updated to set the TRACKER value to IDLE and to clear the SUPPRESS bit to 0. There is no reason for the instruction referenced by the new RIP value to be ENDBR64.

FRED event delivery of a non-maskable interrupt (NMI) blocks NMIs.[1]

A debug trap (single-step trap or data or I/O breakpoint) may be pending at the time another event is delivered. FRED event delivery drops any debug traps that were pending at the time the original event occurred, regardless of the event being delivered. In particular, any pending data or I/O breakpoints (or single-step traps) are no longer pending after INT *n*, INT3, INTO, SYSCALL, or SYSENTER is delivered using FRED event delivery.

Data breakpoints detected during FRED event delivery might not be dropped and may be pending after completion of FRED event delivery. A single-step trap will never be pending following FRED event delivery (even if INT *n*, INT1, INT3, INTO, SYSCALL, or SYSENTER is executed while RFLAGS.TF = 1).

1. For an NMI injected by VM entry, this blocking may apply instead to virtual NMIs. See Section 11.5.4.

5.4 Faults During FRED Event Delivery

The earlier sections identify situations in which faults may occur during FRED event delivery. In addition, a memory access will cause a fault if it uses a non-canonical address or encounters a LASS violation[1] (#SS for an access to the regular stack; #GP for the shadow stack); or if it causes a page fault (#PF).

An exception encountered during FRED event delivery will be converted to a double fault (#DF) following the same principles under which this is done for IDT event delivery. As with IDT event delivery, the error code for the #DF is always zero.

For a #GP or #SS encountered during FRED event delivery that is not converted to #DF, the error code will be 0 if the event that was being delivered had type software interrupt (INT n), software exception (INT3 or INTO), or other event (SYSCALL and SYSENTER); it will be 1 for event types external interrupt, non-maskable interrupt, hardware exception, or privileged software exception (INT1). (This aligns with the existing specification of the EXT bit in such error codes.)

For a #PF encountered during FRED event delivery that is not converted to #DF, the error code is defined exactly as it is for IDT event delivery.

If FRED event delivery causes a fault, any changes to register state are undone before delivering the fault.

1. Because FRED event delivery clears both CPL and RFLAGS.AC, a LASS violation will occur if CR4.LASS = CR4.SMAP = 1 and a memory access uses a linear address that clears bit 63.

6 FRED Return Instructions

FRED defines two new instructions for returning from events delivered by FRED event delivery:

- **ERETS** (Section 6.1) is used to return from events that occur in ring 0; it does not modify CS, SS, or GS. ERETS has the opcode F2 0F 01 CA.
- **ERETU** (Section 6.2) is used to return from events that occur in ring 3; it loads CS and SS based on the stack image and also updates the GS base address. ERETU has the opcode F3 0F 01 CA.

Neither instruction takes explicit operands. These instructions can be executed only if FRED transitions are enabled and only if CPL = 0; in addition, ERETU can be executed only if CSL = 0.

Note: The following sections indicate situations in which one of the instructions may cause a control-protection exception (#CP). These exceptions will use the value 2 for an error code, indicating that the #CP was caused by a return instruction other than near RET.

6.1 ERETS (Event Return to Supervisor)

ERETS returns from an event handler while staying in ring 0, establishing the **return state** that was in effect before FRED event delivery. Because it stays within the supervisor context, ERETS does not modify the segment registers CS, SS, or GS. Execution of ERETS causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0.

ERETS begins by reading and checking the return state from the stack (Section 6.1.1). If supervisor shadow stacks are enabled, it then checks the shadow stack to confirm the validity of this control-flow transfer (Section 6.1.2). Finally, ERETS establishes the return state by loading the appropriate registers (Section 6.1.3).

Appendix A.2 presents the detailed flow of the operation of ERETS.

6.1.1 Loading and Checking the Return State

ERETS operates on the stack frame created by FRED event delivery (Section 5.2.1), principally to restore the state saved in that stack frame (the return state).

When FRED event delivery completes, the return state is located at offset 8 from the RSP value. (RSP references an 8-byte field containing the event's error code.) For this reason, ERETS begins by adding 8 to RSP so that it can access the return state directly.

After adding 8 to RSP, ERETS pops from the regular stack (referenced by RSP) the return state that was saved by FRED event delivery. The state is checked and held by the processor to update register state when the instruction completes. The following items detail the state fields that are popped, from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The RIP of the return state (64 bits).

  This field must be canonical relative to the current paging mode; otherwise, a #GP with a zero error code occurs.

- The augmented CS for the return state (64 bits).

  ERETS does not use bits 15:0 to load CS, but it does ensure that the value of these bits equals that of the current CS selector; otherwise, a #GP occurs.

  Bit 18 of this field determines whether ERETS updates the state of the indirect branch tracker.

  ERETS will establish the new stack level as the minimum of the CSL and the value of bits 17:16 of this field.

  Bits 63:19 of this field must be zero; otherwise, a #GP occurs.

- The RFLAGS of the return state (64 bits).

  Bit 1 of this field must be 1; bit 3, bit 5, bit 15, bit 17 (VM), and bits 63:22 of the field must be zero; otherwise, a #GP occurs.

- The RSP of the return state (64 bits). This field is not checked.

- The augmented SS of the return state (64 bits).

  ERETS does not use bits 15:0 to load SS, but it does ensure that the value of these bits equals that of the current SS selector; otherwise, a #GP occurs.

  Bits 18:16 of this field determine how ERETS manages certain events. See Section 6.1.3.

  Bits 31:19 of this field must be zero; otherwise, a #GP occurs. ERETS ignores bits 63:32 of this field (the event information).

6.1.2 Checking the Shadow Stack

If supervisor shadow stacks are enabled, ERETS pops and checks values from the shadow stack (referenced by SSP) that were saved by FRED event delivery. The following items detail the state that is popped from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The SSP of the return state (64 bits).

  This value must be 4-byte aligned (bits 1:0 must be zero); otherwise, a control-protection exception (#CP) occurs.

  This value must be canonical relative to the processor's maximum linear-address width; otherwise, a #GP occurs with a zero error code. (This #GP has priority below the #CP exceptions specified in the following items.)

- The linear instruction pointer of the return state (64 bits).

  This value must equal the RIP of the return context that was popped from the regular stack; otherwise, a #CP occurs.

- The augmented CS for the return state (64 bits).

  This value must equal the code-segment value that was popped from the regular stack; otherwise, a #CP occurs.

If supervisor shadow stacks are enabled and the new stack level is less than the CSL, ERETS compares the value of SSP (after popping the above fields) and the value of the FRED SSP MSR for the CSL (not the new stack level). (For example, if ERETS is executing with CSL = 2 and is returning to stack level 1, the comparison is with IA32_FRED_SSP2.) If the values are not equal, a #CP occurs.

6.1.3 Establishing the Return State

After the stack operations described in Section 6.1.1 and Section 6.1.2, ERETS then loads RIP, RFLAGS, RSP, and CSL with the values that were popped earlier from the regular stack. If supervisor shadow stacks are enabled, SSP is loaded with the value that was popped earlier from the shadow stack.

Additional steps are performed based on the value of the popped fields for CS and SS:

- If bit 18 of the CS field is 1 and ERETS will complete with indirect branch tracking enabled and not suppressed, the indirect branch tracker will be in the WAIT_FOR_ENDBRANCH state upon completion of ERETS.[1]
- If bit 16 of the SS field is 1, bit 9 of the RFLAGS field (corresponding to RFLAGS.IF) is 1, and there was no blocking by STI prior to ERETS, blocking by STI is in effect upon completion of ERETS.
- If bit 17 of the SS field is 1 and ERETS will result in RFLAGS.TF = 1, a single-step trap will be pending upon completion of ERETS.[2]
- If bit 18 of the SS field is 1, ERETS unblocks NMIs.[3]

6.2 ERETU (Event Return to User)

ERETU returns from an event handler while making a transition to ring 3, establishing the **return context** that was in effect before FRED event delivery. The change of context includes updates to the segment registers CS, SS, or GS. Execution of ERETU causes an invalid-opcode exception (#UD) if FRED transitions are not enabled or if CPL > 0, and a general-protection exception (#GP) if CSL > 0.

ERETU begins by reading and checking the return state from the stack (Section 6.2.1). If supervisor shadow stacks are enabled, it then checks the validity of the old and new shadow-stack pointers (Section 6.2.2). Finally, ERETU establishes the return state by loading the appropriate registers (Section 6.2.3).

Appendix A.3 presents the detailed flow of the operation of ERETU.

6.2.1 Loading and Checking the Return State

ERETU operates on the stack frame created by FRED event delivery (Section 5.2.1), principally to restore the state saved in that stack frame (the return state).

When FRED event delivery completes, the return state is located at offset 8 from the RSP value. (RSP references an 8-byte field containing the event's error code.) For this reason, ERETU begins by adding 8 to RSP so that it can access the return state directly.

1. ERETS will complete with indirect branch tracking enabled and not suppressed if CR4.CET = 1, IA32_S_CET.ENDBR_EN = 1, and IA32_S_CET.SUPPRESS = 0. If ERETS puts the indirect branch tracker in the WAIT_FOR_ENDBRANCH state, it will result in IA32_S_CET.TRACKER = 1.
2. If ERETS begins execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETS, regardless of the values on the stack for RFLAGS.TF and CS.
3. If in VMX non-root operation with the 1-setting of the "virtual NMIs" VM-execution control, this step unblocks **virtual** NMIs. See Section 11.

After adding 8 to RSP, ERETU first pops from the regular stack (referenced by RSP) the return state that was saved by FRED event delivery. The state is checked and held by the processor to update register state when the instruction completes. The following items detail the state fields that are popped, from top (lowest address) to bottom (highest address), specifying the checks that are performed:

- The RIP of the return state (64 bits).

  This field is checked once the new CS configuration is determined (see below).

- The augmented CS of the return state (64 bits).

  Bits 15:0 contain the new CS selector itself. ERETU ensures that the value of bits 1:0 is 3; otherwise, a #GP occurs.

  Bits 63:16 of this field must be zero; otherwise, a #GP occurs.

- The RFLAGS of the return state (64 bits).

  Bit 1 of this field must be 1; bit 3, bit 5, bits 13:12 (IOPL), bit 15, bit 17 (VM), and bits 63:22 of the field must be zero; otherwise, a #GP occurs.

Note: Checking IOPL ensures that, when FRED transitions are enabled, IOPL is always 0 when CPL = 3.

- The RSP of the return state (64 bits). This field is not checked.

- The augmented SS of the return state (64 bits).

  Bits 15:0 contain the new SS selector itself. ERETU ensures that the value of bits 1:0 is 3; otherwise, a #GP occurs.

  Bit 17 and bit 18 of this field determine how ERETU manages certain events. See Section 6.2.3.

  Bits 31:19 of this field must be zero; otherwise, a #GP occurs. ERETU ignores bit 16 of this field, as well as bits 63:32 (the event information).

After popping these fields, ERETU determines the configuration of the CS and SS segment registers for the return state according to one of these three cases:

1. If the selector popped for CS is IA32_STAR[63:48] + 16 and the selector popped for SS is IA32_STAR[63:48] + 8, ERETU establishes CS and SS in a standard configuration for ring 3 in 64-bit mode:
   - CS:
     - The selector is set to the selector popped for CS.
     - The base address is set to 0. The limit is set to FFFFFH and the G bit is set to 1.
     - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
     - The DPL is set to 3, the P and L bits are each set to 1, and the D bit is set to 0.
     - CS is made usable regardless of the value of the selector.
   - SS:
     - The selector is set to the selector popped for SS.
     - The base address is set to 0. The limit is set to FFFFFH and the G bit is set to 1.
     - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
     - The DPL is set to 3, and the P and B bits are each set to 1.
     - SS is made usable regardless of the value of the selector.

2. If the selector popped for CS is IA32_STAR[63:48] and the selector popped for SS is IA32_STAR[63:48] + 8, ERETU will establish CS and SS in a standard configuration for ring 3 in compatibility mode:
   — CS:
      - The selector is set to the selector popped for CS.
      - The base address is set to 0. The limit is set to FFFFFH and the G bit is set to 1.
      - The type is set to 11 (execute/read accessed code) and the S bit is set to 1.
      - The DPL is set to 3, the P and D bits are set to 1, and the L bit is set to 0.
      - CS is made usable regardless of the value of the selector.
   — SS:
      - The selector is set to the selector popped for SS.
      - The base address is set to 0. The limit is set to FFFFFH and the G bit is set to 1.
      - The type is set to 3 (read/write accessed data) and the S bit is set to 1.
      - The DPL is set to 3, and the P and B bits are each set to 1.
      - SS is made usable regardless of the value of the selector.
3. Otherwise, the selectors popped for CS and SS are used to load descriptors from the GDT or an LDT as would be done by an execution of IRET. These descriptor loads may cause ERETU to fault as would occur with IRET.

The RIP of the return state is then checked, depending on the mode to which ERETU is returning:

- If ERETU is returning to 64-bit mode (either case #1 above, or case #3, where the descriptor loaded for CS sets the L bit), a #GP with a zero error code occurs if the return RIP is not canonical relative to the current paging mode.
- If ERETU is returning to compatibility mode (either case #2 above, or case #3, where the descriptor loaded for CS clears the L bit), the upper 32 bits of the return RIP are cleared. Following this, a #GP with a zero error code occurs if the truncated RIP value exceeds the new CS segment limit (taking into account that segment's G bit). (Such a #GP will never occur for case #•, as it establishes a limit of FFFFFFFFH.)

6.2.2 Checking the Shadow-Stack Pointers

If user shadow stacks are enabled, the SSP of the return state is the value of the IA32_PL3_SSP MSR. If the return is to compatibility mode, a #GP occurs if IA32_PL3_SSP[63:32] are not all zero.

If supervisor shadow stacks are enabled, ERETU compares the current (supervisor) SSP value and the value of the IA32_FRED_SSP0 MSR. If the values are not equal, a #CP occurs.

Note: Execution of ERETU does not access any shadow stack.

6.2.3 Establishing the Return State

After the stack operations described earlier, ERETU loads RIP, RFLAGS, RSP, CS, and SS with the values identified in Section 6.2.1. If the return is to compatibility mode, the upper 32 bits of each of RIP and RSP are cleared to zero. If user shadow stacks are enabled, SSP is loaded with the value for the return state as identified in Section 6.2.2 (from the IA32_PL3_SSP MSR).

ERETU swaps the value of the GS base address and that of the IA32_KERNEL_GS_BASE MSR.

Additional steps are performed based on the value of the popped SS field:

- If bit 17 of the field is 1 and ERETU would result in RFLAGS.TF = 1, a single-step trap will be pending upon completion of ERETU.[1]
- If bit 18 of the field (above the selector) is 1, ERETU unblocks NMIs.[2]

Note: Unlike IRET, ERETU does not make any of DS, ES, FS, or GS null if it is found to have DPL < 3. It is expected that a FRED-enabled operating system will return to ring 3 (in compatibility mode) only when those segments all have DPL = 3.

6.2.4 Interactions Between ERETU and Other Instructions

ERETU provides the same instruction-ordering guarantees as SYSRET. Specifically, instructions following an execution of ERETU may be fetched from memory before earlier instructions complete execution, but they will not execute (even speculatively) until all instructions prior to the ERETU have completed execution. (The later instructions may execute before data stored by the earlier instructions have become globally visible.)

Execution of the UMWAIT instruction will not wait (will not enter an implementation-dependent optimized state) if ERETU was executed before UMWAIT and after the most recent execution of the UMONITOR instruction.

1. If ERETU began execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETU, regardless of the values on the stack for RFLAGS.TF and CS.
2. If in VMX non-root operation with the 1-setting of the “virtual NMIs” VM-execution control, this step unblocks **virtual** NMIs. See Section 11.

7 Intel® Processor Trace

Intel® Processor Trace (Intel PT) is an extension of Intel® Architecture that captures information about software execution (including control-flow tracing) in data packets. The packets include timing, program flow information and program-induced mode related information.

When appropriately enabled, these packets may be generated by IDT event delivery and executions of the IRET instruction. When FRED transitions are enabled, FRED event delivery and executions of ERETS and ERETU generate packets in a similar manner.

The following items provide details for FRED event delivery:

- FRED event delivery of an exception, external interrupt, or non-maskable interrupt generates the same packets that IDT event delivery would generate for the same event.
- FRED event delivery of a software interrupt (INT *n*, INT3, INTO, or INT1) generates the same packets that IDT event delivery would generate for the same event.

  The only exception is the Control Flow Event (CFE) packet that is generated if event trace is enabled (IA32_RTIT_CTL.EventEn[bit 31] = 1). With FRED event delivery, that packet will use event type 0EH (SWINTR) instead of 1 (INTR). There are no other differences in the CFE packet or in other packets produced.
- FRED event delivery of SYSCALL or SYSENTER generates the same Target IP packet generated by execution of the instruction without FRED. If event trace is enabled, a CFE packet and a Flow Update (FUP) packet are also generated. The CFE packet will use event type 0FH (SYSCALL) and will set the IP bit, indicating that the FUP packet follows. The vector field in the CFE packet is not valid. The IP value in the FUP packet will be the address of the SYSCALL or SYSENTER instruction (“CLIP”). No event-data (EVD) packet is produced.
- Executions of ERETS and ERETU produce the same packets produced by IRET.

8 FRED and Existing Instructions

The Intel® 64 architecture defines numerous instructions that can effect ring transitions. This section considers those instructions (and others) and describes how enabling FRED transitions affects either their operation or their usage. (Recall that FRED transitions are enabled if CR4.FRED = 1.)

- Section 8.1 identifies instructions that cannot be executed when FRED transitions are enabled.
- Section 8.2 considers far CALL, far JMP, far RET, and IRET. Enabling FRED transitions modifies the operation of these instructions. A FRED-enabled operating system cannot use them for ring transitions or to enter compatibility mode in ring 0.
- Section 8.3 discusses software interrupts and related instructions (INT *n*, INT3, INTO, and INT1). When FRED transitions are enabled, these instructions use FRED event delivery.
- Section 8.4 describes changes to the instructions SYSCALL and SYSENTER. When FRED transitions are enabled, these instructions also use FRED event delivery.
- Section 8.5 discuss interactions with the GETSEC instructions that enter or exit authenticated code execution mode.

8.1 Disallowed Instructions

When FRED transitions are enabled, execution of any of the following instructions causes an invalid-opcode exception (#UD): CLRSSBSY, SETSSBSY, SWAPGS, SYSEXIT, and SYSRET.

Note: CLRSSBSY, SETSSBSY, and SWAPGS were defined to account for anomalous situations that do not exist with FRED. In addition, CLRSSBSY and SETSSBSY operate on supervisor shadow-stack tokens, which are not used when FRED transitions are enabled. If new FRED-appropriate functionality is identified for these instructions, the instructions may be allowed when FRED transitions are enabled (with that new functionality).

8.2 Far CALL, IRET, Far JMP, and Far RET

Enabling FRED transitions disables the ability of certain existing instructions to effect ring transitions or certain other mode changes.

The far CALL instruction provides a mechanism by which user software can effect a ring transition to more privileged software. The IRET and far RET instructions allow returns to the calling context, and this can cause a ring transition. The operation of these instructions (as well as that of far JMP) is modified when FRED transitions are enabled.

Far CALL can effect a ring transition only if it references a call gate in the GDT or an LDT. (Far JMP can also reference a call gate, but an execution that does so cannot cause a ring transition.) When FRED transitions are enabled, any execution of far CALL or far JMP that references a call gate causes a general-protection exception (#GP). (The

call-gate descriptor will not be modified.) The exception's error code will include the selector that referenced the call gate. This exception has the same priority as that of existing checks on the type of the target descriptor.

An execution of IRET or far RET causes a ring transition if the RPL value of selector referencing the target code segment is greater than the CPL. When FRED transitions are enabled, such an execution causes #GP. (The code-segment descriptor will be accessed but not modified.) The exception's error code will include the selector that referenced the code segment. This exception has the same priority as that of the existing check that the RPL cannot be less than the CPL.

These restrictions imply that, when they are enabled, FRED transitions are the only ring transitions possible and that it is impossible to enter ring 1 or ring 2.

In addition, enabling FRED transitions disables the ability of software to enter compatibility mode in ring 0. An execution of far CALL, IRET, far JMP, or far RET causes a transition to compatibility mode if it loads CS from a code-segment descriptor in which the L bit is 0. (The L bit is bit 53 of the 64-bit descriptor.) When FRED transitions are enabled, such an execution in ring 0 causes #GP. The exception's error code will include the selector that referenced the code segment. This exception has the same priority as that of existing checks on the type of the target descriptor.

Note: The restrictions described in this section do not prevent use of far CALL, IRET, far JMP, or far RET to enter compatibility mode when the instruction is executed in ring 3 and does not reference a call gate. They can be used in ring 0 only if their execution stays in ring 0 and in 64-bit mode.

8.3 Software Interrupts and Related Instructions

The Intel 64 architecture supports the following instructions that software in ring 3 can use to invoke the operating system using the IDT:

- INT *n* (opcode CD followed by an immediate byte). There are 256 such software-interrupt instructions, one for each value *n* of the immediate byte (0–255).
- INT3 (opcode CC). This instruction generates a breakpoint exception (#BP) as a trap.
- INTO (opcode CE). If RFLAGS.OF = 1, this instruction generates an overflow exception (#OF) as a trap. If RFLAGS.OF = 0, this instruction does not generate an exception and control passes to the next instruction. Executing INTO in 64-bit mode causes an invalid-opcode exception (#UD), but the instruction can be used to generate #OF in compatibility mode.
- INT1 (opcode F1). This instruction generates a debug exception (#DB) as a trap. Hardware vendors may use INT1 for hardware debug.

When FRED transitions are enabled, an execution of any of these instructions (except for INTO if RFLAGS.OF = 0) results in FRED event delivery. Section 5.2.1 describes how the resulting FRED event delivery saves event information for these instructions (including the length of the instruction).

Note: IDT event delivery of INT *n*, INT3, and INTO checks the DPL field of the IDT gate and generates a general-protection exception (#GP) if it is less than the CPL. (Delivery of INT1 does not do this.) FRED event delivery does not use the IDT and thus does not perform this check. The event handlers of a FRED-enabled operating system should check the event type and vector to identify situations that would have resulted in a fault with IDT event delivery.

8.4 SYSCALL and SYSENTER

The Intel 64 architecture supports two instructions that software in ring 3 can use to invoke the operating system without using the IDT: SYSCALL and SYSENTER.

When FRED transitions are enabled, the operation of SYSCALL is modified to use FRED event delivery (Section 5) in place of its existing operation.

The following pseudocode describes the operation of SYSCALL on a processor that supports FRED transitions, detailing the order of certain fault checking:

```
IF IA32_EFER.LMA = 0 OR CS.L = 0          // SYSCALL can be used only in 64-bit mode
    THEN #UD;
ELSIF CR4.FRED = 0
    THEN
        IF IA32_EFER.SCE = 0
            THEN #UD;
            ELSE existing SYSCALL operation;
        FI;
    ELSE    // FRED does not require enabling in IA32_EFER
        FRED event delivery of SYSCALL;     // the instruction's length will be saved on the stack
FI;
```

A FRED-enabled operating system should use ERETU (Section 6.2) instead of SYSRET to return after handling a system call invoked by an execution of SYSCALL in ring 3. (Recall that any execution of SYSRET causes #UD when FRED transitions are enabled. A FRED-enabled operating system will normally use ERETU to return from any event that occurred when in ring 3.)

(Because the SYSRET instruction always returns to ring 3, use of SYSCALL has previously been effectively limited to ring 3. When SYSCALL uses FRED event delivery, that instruction can be used effectively in ring 0. If this is done, a FRED-enabled operating system would naturally return with ERETS, remaining in ring 0.)

The operation of SYSENTER is also modified to use FRED event delivery when FRED transitions are enabled. The following pseudocode describes the operation of SYSENTER on a processor that supports FRED transitions, detailing the order of certain fault checking:

```
IF CR0.PE = 0 OR (CR4.FRED = 0 AND IA32_SYSENTER_CS[15:2] = 0)
    THEN #GP;          // IA32_SYSENTER_CS applies only if FRED transitions are not enabled
ELSIF CR4.FRED = 0
    THEN existing SYSENTER operation;
    ELSE FRED event delivery of SYSENTER;    // the instruction's length will be saved on the stack
FI;
```

A FRED-enabled operating system should use ERETU (Section 6.2) instead of SYSEXIT to return after handling a system call invoked by an execution of SYSENTER in ring 3. (Recall that any execution of SYSEXIT causes #UD when FRED transitions are enabled.)

(Because the SYSEXIT instruction always returns to ring 3, use of SYSENTER has previously been effectively limited to ring 3. When SYSENTER uses FRED event delivery, that instruction can be used effectively in ring 0. If this is done, a FRED-enabled operating system would naturally return with ERETS, remaining in ring 0.)

8.5 GETSEC and Authenticated-Code Execution Mode

Software can invoke an authenticated-code module that operates in authenticated-code execution mode using the ENTERACCS and SENTER functions of the GETSEC instruction. The authenticated-code module leaves authenticated-code execution mode by executing the GETSEC function EXITAC.

After an authenticated-code module is invoked, events should not be delivered using FRED event delivery using a configuration that was established before entry to authenticated-code execution mode. Processors ensure that by disabling FRED transitions when entering authenticated-code execution mode. A subsequent execution of GETSEC[EXITAC] will re-enable FRED transitions if they had been enabled earlier.

To support this functionality, processors that support FRED transitions implement the features described in the remainder of this section.

A four-byte field called Cr4High is defined at offset 92 in the header of an authenticated-code module (ACM). This field is used by entry to and exit from authenticated-code execution mode.

Execution of either GETSEC[ENTERACCS] or GETSEC[SENTER] saves the current value of CR4[63:32] into the Cr4High field in the ACM header and then clears CR4.FRED to 0. This ensures that FRED event delivery will not be used in authenticated-code execution mode until the ACM has had the opportunity to configure and enable FRED.

If the logical processor entered authenticated-code execution mode using GETSEC[ENTERACCS], execution of GETSEC[EXITAC] reads the Cr4High field in the ACM header. If bit 0 of the Cr4High field in the ACM header (corresponding to CR4.FRED) is 1, a general-protection fault (#GP) occurs if GETSEC[EXITAC] is executed (and would thus end) outside IA-32e mode (IA32_EFER.LMA = 0). Otherwise, GETSEC[EXITAC] loads CR4.FRED with the value of that bit. (If the logical processor entered authenticated-code execution mode in some other manner, GETSEC[EXITAC] does not read the Cr4High field in the ACM header and does not modify CR4.)

9 LKGS: Support for Managing GS

64-bit operating systems and their applications use the GS segment for thread-local storage (TLS). Because the operating system and applications use the TLS at different addresses, they use different base addresses for that segment.

FRED transitions ensure that an operating system can always operate with its own GS base address:

- For events that occur in ring 3, FRED event delivery swaps the GS base address with the IA32_KERNEL_GS_BASE MSR.
- ERETU (the FRED transition that returns to ring 3) also swaps the GS base address with the IA32_KERNEL_GS_BASE MSR.

An operating system can modify the GS base address of a user thread (e.g., as part of a context switch) by updating the IA32_KERNEL_GS_BASE MSR. However, existing instructions do not allow an operating system to modify other attributes of the GS segment without compromising its ability always to operate with its own GS base address. This is because the instructions that update those attributes (by loading them from a descriptor table) also update the GS base address.

The FRED architecture addresses this deficiency with a new instruction **LKGS** (abbreviating "load into IA32_KERNEL_GS_BASE"). LKGS behaves like the MOV to GS instruction except that it loads the base address into the IA32_KERNEL_GS_BASE MSR instead of the GS segment's descriptor cache.

The following items provide details regarding the LKGS instruction:

- Support for LKGS is enumerated with the feature flag CPUID.(EAX=7,ECX=1):EAX.LKGS[bit 18].
- LKGS has the opcode F2 0F 00 /6.
- Execution of LKGS causes an invalid-opcode exception (#UD) if CPL > 0 or if the logical processor is not in 64-bit mode.
- LKGS takes a single 16-bit operand (in a register or memory) and uses it to load a descriptor from the GDT or an LDT. It is subject to the same faults as would be incurred by the MOV to GS instruction. If there is no fault, the operand is loaded into the GS selector.
- The descriptor read by LKGS is loaded into the GS segment's descriptor cache as is done by the MOV to GS instruction except that the base address in the descriptor is not loaded into the descriptor cache (the base address in the cache is unmodified) but is instead loaded into the IA32_KERNEL_GS_BASE MSR. Because the base address in the descriptor is only 32 bits, LKGS clears the upper 32 bits of the 64-bit IA32_KERNEL_GS_BASE MSR.
- If the 16-bit operand is a null selector (values 0–3), the GS segment is marked unusable outside 64-bit mode and zero is loaded into the IA32_KERNEL_GS_BASE MSR. (The base address in the GS descriptor cache is not modified.)

The LKGS instruction and the nature of FRED transitions (above) remove any need for an operating system to use the SWAPGS instruction. As noted in Section 8.1, an execution of SWAPGS causes an invalid-opcode exception (#UD) if FRED transitions are enabled.

10 NMI-Source Reporting

A non-maskable interrupt (NMI) is a special event that is distinguished from other (maskable) interrupts in a number of ways.

Each maskable interrupt is associated with an 8-bit vector that is included in platform interrupt messages (or which is programmed into various registers in the local APIC). IDT event delivery uses this vector to select from the IDT a descriptor that controls interrupt delivery.

NMIs also result from platform interrupt messages (or registers in the local APIC), and these also usually include an 8-bit vector.[1] However, IDT event delivery ignores this vector and instead always uses vector 2 to deliver an NMI, selecting descriptor 2 from the IDT.

As explained in Section 5.2.1, FRED event delivery also uses vector 2 for NMIs. (It also distinguishes NMIs from other events with event type 2.) Processors that support FRED may also support a related feature called **NMI-source reporting**. With NMI-source reporting, FRED event delivery reports information about the sources of NMIs based on the vectors in NMI interrupt messages (or in the local APIC). When NMI-source reporting is available, software can program different NMI sources to use different vectors, allowing software to identify the source of an NMI when it is delivered.

Support for NMI-source reporting is enumerated with the feature flag CPUID.(EAX=7,ECX=1):EAX.NMI_SRC[bit 20].

When NMI-source reporting is supported, each logical processor maintains a 16-bit **NMI-source bitmap**. The bitmap is zero on reset. When a logical processor receives an NMI, it updates the NMI-source bitmap as follows:

- If the NMI has a vector (as specified in an interrupt message or APIC register) in the range 0–15, the processor sets the bit corresponding to the vector.
- If the NMI has no vector, or if its vector is not in the range 0–15, the logical processor sets bit 0.

A logical processor may receive multiple NMIs before the first is delivered. When this happens, the NMIs are coalesced and only one is delivered to software. In this case, NMI-source reporting will set multiple bits in the NMI-source bitmap (assuming the NMIs did not all have the same vector).

When a logical processor delivers an NMI, as an event or VM exit, it notes the value of the NMI-source bitmap and then clears it.

- If the NMI is delivered with FRED event delivery, the noted value of the NMI-source bitmap is saved on the stack as event data (see Section 5.2.1).
- IDT event delivery does not report the NMI-source bitmap to software, even if the processor supports NMI-source reporting.
- If an NMI causes a VM exit, the noted value of the NMI-source bitmap is saved into the VMCS as the exit qualification. See Section 11.6.2.

1. Some platforms may have sources of NMIs that do not provide a vector.

11 VMX Interactions with FRED Transitions

This section describes interactions between FRED transitions and the VMX architecture.

Section 11.1 presents the renaming of certain existing fields in the VMCS. Section 11.2 introduces a new VMX feature necessary for proper virtualization of FRED event delivery. Section 11.3 details additions to the VMCS to support FRED transitions. Section 11.4 describes the operation of FRED transitions in VMX non-root operation. Section 11.5 and Section 11.6 discuss interactions with VM entries and VM exits, respectively.

11.1 Renaming of Existing VMCS Fields

The VMCS contains certain fields related to event delivery. These fields are being renamed for clarity and consistency with FRED event delivery. The following items provide details:

- For events causing VM exits:
  - Basic event identification (VMCS encoding 4404H):
    - Old name: VM-exit interruption information
    - New name: **Exiting-event identification**
  - Error code (VMCS encoding 4406H):
    - Old name: VM-exit interruption error code
    - New name: **Exiting-event error code**
- For VM exits occurring during event delivery:
  - Basic event identification (VMCS encoding 4408H):
    - Old name: IDT-vectoring information
    - New name: **Original-event identification**
  - Error code (VMCS encoding 440AH):
    - Old name: IDT-vectoring error code
    - New name: **Original-event error code**
- For events injected by VM entry:
  - Basic event identification (VMCS encoding 4016H):
    - Old name: VM-entry interruption information
    - New name: **Injected-event identification**
  - Error code (VMCS encoding 4018H):
    - Old name: VM-entry exception error code
    - New name: **Injected-event error code**

For the basic event-identification fields, the sub-field at bits 10:8 is being renamed from "interruption type" to "**event type**."

11.2 New VMX Feature: VMX Nested-Exception Support

As noted in Section 5.1.2, the event stack level used by FRED event delivery depends on whether the event was a nested exception encountered during delivery of another event. In addition, FRED event delivery sets a bit on the stack when delivering a nested exception. For proper virtualization of these details, processors that support FRED will also support a new VMX feature called **VMX nested-exception support**. A processor enumerates VMX nested-exception support by setting bit 58 in the VMX capability MSR IA32_VMX_BASIC (index 480H). Any processor that enumerates support for FRED transitions (see Section 4.1) will also enumerate VMX nested-exception support.

VMX nested-exception support changes the way in which VM exits establish certain VM-exit information fields and the way in which VM entries use a related VM-entry control field:

- Exiting-event identification. This VM-exit information field is valid for VM exits due to events that would have been delivered to guest software (with IDT event delivery or FRED event delivery) if they had not caused a VM exit. The field provides details about the nature of the event causing the VM exit.

  VMX nested-exception support defines bit 13 of this field, which is always saved as 0 by processors without VMX nested-exception support.

  With VMX nested-exception support, a VM exit saves bit 13 of this field as 1 if the VM exit is due to a nested exception encountered during delivery of an earlier event using FRED event delivery; it is not set for VM exits due to #DF or to those encountered during IDT event delivery.

  Other VM exits for which the field is valid (including VM exit due to #DF) save bit 13 as 0.

  (The value of this bit is always identical to that of the valid bit of the original-event identification field.)

- Original-event identification. This VM-exit information field is valid for VM exits due to events encountered during delivery of an earlier event being delivered to guest software with IDT event delivery or FRED event delivery (including SYSCALL and SYSENTER with FRED event delivery). The field provides details about the nature of that earlier event.

  VMX nested-exception support defines bit 13 of this field, which is always saved as 0 by processors without VMX nested-exception support.

  With VMX nested-exception support, a VM exit saves bit 13 of this field as 1 if the earlier event was itself a nested exception encountered during delivery of an even earlier event using FRED event delivery; it is not set for VM exits due to #DF or due to exceptions encountered during IDT event delivery.

  Other VM exits for which the field is valid (including VM exits due to events encountered during delivery of #DF) save bit 13 as 0.

- Injected-event identification. Software establishes a valid value in this VM-entry control field to specify an event to be injected at the end of the next VM entry.

  VMX nested-exception support defines bit 13 of this field. (For processors without VMX nested-exception support, VM entry fails if this bit is 1.)

  With VMX nested-exception support, VM entry allows bit 13 to be 1 if the field indicates injection of a hardware exception (bits 10:8, the event type, should have value 3). If FRED transitions will be enabled in the guest and thus the injected exception will be delivered using FRED event delivery, the event information saved

on the stack will set a bit to indicate that the event was a nested exception (see Section 5.2.1). In addition, the event's stack level is determined as if the event had been a nested exception encountered during delivery of another event (see Section 5.1.2). If FRED transitions will not be enabled in the guest, bit 13 of the field is ignored.

VM entry fails if bit 13 of this field is 1 and the field indicates injection of an event other than an exception (bits 10:8 of the field have value other than 3).

11.3 VMCS Changes for FRED Transitions

A VMM (or its hosting operating system) should be able to use FRED transitions and to allow guest software to do so. For that reason, VMX transitions (VM entries and VM exits) must establish context sufficient to support FRED event delivery immediately after the transition. In addition, VM exits should be able to save the corresponding guest context before loading that for the VMM.

To support this context management, new fields are added to the VMCS that correspond to some of the following configuration MSRs identified in Section 4.3:

- MSRs added for FRED transitions: IA32_FRED_CONFIG, IA32_FRED_RSP0, IA32_FRED_RSP1, IA32_FRED_RSP2, IA32_FRED_RSP3, IA32_FRED_STKLVLS, IA32_FRED_SSP1, IA32_FRED_SSP2, and IA32_FRED_SSP3.
- Existing MSRs used by FRED transitions: IA32_STAR, IA32_KERNEL_GS_BASE, and IA32_PL0_SSP (also known as IA32_FRED_SSP0).

Fields for some of these MSRs are added to both the host-state and guest-state areas of the VMCS. Details are provided in Section 11.3.1 and Section 11.3.2. Section 11.3.3 enumerates the new VMX controls that manage the new MSRs.

As will be explained, VMCS fields are not needed for all of these MSRs. In particular, fields are not added for IA32_FRED_RSP0, IA32_STAR, IA32_KERNEL_GS_BASE, and IA32_PL0_SSP. Before VM entry, a virtual-machine monitor should ensure that these MSRs contain the values expected by guest software in the virtual machine being entered (e.g., with the WRMSR instruction).

As explained in Section 5.2.1, FRED event delivery saves **event data** that provide additional information about certain events (e.g., page faults). To support proper handling of event data across VMX transitions, new fields are added to the VMCS. These are described in Section 11.3.4.

11.3.1 Host-State Area

As noted earlier, each VM exit must establish the configuration required for FRED transitions that might occur immediately after the VM exit.

The CPL is always 0 after any VM exit. For that reason, delivery of an event that arrives immediately after a VM exit cannot cause a ring transition; the return from such an event will use ERETS, not ERETU. As a result, the following MSRs will not be needed for delivery of and return from such an event:

- IA32_FRED_RSP0 and IA32_PL0_SSP (also known as IA32_FRED_SSP0). If CPL = 0, FRED event delivery loads RSP from a FRED RSP MSR only if the stack level is numerically increasing; consequently, such FRED event delivery would not use IA32_FRED_RSP0 or IA32_PL0_SSP. Similarly, ERETS checks IA32_FRED_SSP*i* only when returning from stack level *i* to a numerically lower stack level; as a result,

ERETS would never refer to IA32_PL0_SSP. (ERETS does not use the FRED RSP MSRs at all.)

- IA32_STAR. FRED event delivery uses this MSR only when loading CS and SS when delivering an event that arrives in ring 3. ERETS does not use this MSR.
- IA32_KERNEL_GS_BASE. FRED event delivery swaps this MSR with the GS base address only when delivering an event that arrives in ring 3. ERETS does not do this swapping.

Thus, 64-bit fields are added to host-state area of the VMCS for the following MSRs (the encoding pair for each field is shown parenthetically):

- IA32_FRED_CONFIG (2C08H/2C09H)
- IA32_FRED_RSP1 (2C0AH/2C0BH)
- IA32_FRED_RSP2 (2C0CH/2C0DH)
- IA32_FRED_RSP3 (2C0EH/2C0FH)
- IA32_FRED_STKLVLS (2C10H/2C11H)
- IA32_FRED_SSP1 (2C12H/2C13H)
- IA32_FRED_SSP2 (2C14H/2C15H)
- IA32_FRED_SSP3 (2C16H/2C17H)

11.3.2 Guest-State Area

Section 11.3.1 identified fields for MSRs added to the host-state area of the VMCS. Since these MSRs will be loaded by VM exits, it must be possible for their guest values to be saved earlier by those VM exits. For that reason, corresponding 64-bit fields are added to the guest-state area of the VMCS (the encoding pair for each field is shown parenthetically):

- IA32_FRED_CONFIG (281AH/281BH)
- IA32_FRED_RSP1 (281CH/281DH)
- IA32_FRED_RSP2 (281EH/281FH)
- IA32_FRED_RSP3 (2820H/2821H)
- IA32_FRED_STKLVLS (2822H/2823H)
- IA32_FRED_SSP1 (2824H/2825H)
- IA32_FRED_SSP2 (2826H/2827H)
- IA32_FRED_SSP3 (2828H/2829H)

11.3.3 VMX Controls

The following VMX controls are added to support management of FRED context:

- VM-entry control 23 is “load FRED.” If this control is set, VM entries load the guest FRED state identified in Section 11.3.2.
- Secondary VM-exit control 0 is “save FRED.” If this control is set, VM exits save the guest FRED state identified in Section 11.3.2.
- Secondary VM-exit control 1 is “load FRED.” If this control is set, VM exits load the host FRED state identified in Section 11.3.1.

11.3.4 Event-Data Fields

New VMCS fields are defined for the event data saved by FRED event delivery

The first new field is a 64-bit VMX control field called **injected-event data**. If VM-entry injection of an event uses FRED event delivery, the event data saved on the stack is the value of this field (see Section 11.5.4). This field has uses the encoding pair 2052H/2053H.

The second new field is a 64-bit exit-information field called **original-event data**. If a VM exit occurs during FRED event delivery, the event data that would have saved on the stack is instead stored into this field (see Section 11.6.3). This field uses the encoding pair 2404H/2405H.

Note: No new exit-information field is added for exiting-event data (event data for event causing a VM exit). As will be explained in Section 11.6.2, such data may be saved in the existing exit-qualification field.

11.4 FRED Transitions and VMX Non-Root Operation

If FRED transitions are enabled in VMX non-root operation, the architectural changes identified in this specification apply to guest execution. Interactions between the features are described in this section.

11.4.1 VM Exits Due to Events

A VMM may use existing VMX features to specify that VM exits should occur on the occurrence of events that would be delivered with FRED event delivery. These events include external interrupts, non-maskable interrupts (NMIs), and exceptions (including those generated by INT1, INT3, and INTO). If such an event occurs, any specified VM exit occurs regardless of whether FRED transitions are enabled. FRED event delivery does not occur. See Section 11.6 for more details.

11.4.2 NMI Blocking

As specified in Section 5.3, FRED event delivery of a non-maskable interrupt (NMI) blocks NMIs. That does not change in VMX non-root operation. Note, however, that an NMI can be delivered in VMX non-root operation only if the “NMI exiting” VM-execution control is 0. (The treatment of NMI blocking by FRED event delivery of an NMI injected by VM entry is given in Section 11.5.4.)

As specified in Section 6.1.3 and Section 6.2.3, ERETS and ERETU each unblocks NMIs if bit 18 is set in the augmented SS in the return state. The following items detail how this behavior may be changed in VMX non-root operation, depending on the settings of certain VM-execution controls:

- If the “NMI exiting” VM-execution control is 0, this behavior of ERETS and ERETU is not modified (they unblock NMIs as indicated above).
- If the “NMI exiting” VM-execution control is 1, ERETS and ERETU do not unblock physical NMIs.
- If the “virtual NMIs” VM-execution control is 1 (which implies that the “NMI exiting” VM-execution control is also 1), the logical processor tracks virtual-NMI blocking. In this case, ERETS and ERETU each unblocks virtual NMIs if bit 18 is set in the augmented SS in the return state.

(If the "NMI exiting" VM-execution control is 1 and the "virtual NMIs" VM-execution control is 0, ERETS and ERETU ignore bit 18 of the augmented SS.)

11.5 FRED Transitions and VM Entries

This section describes the interactions between VM entries and various aspects of FRED transitions.

Some aspects of VM entry are changed if FRED transitions will be enabled following VM entry. FRED transitions are enabled following VM entry if bit 32 of the CR4 field (FRED) in the guest-state area is 1.

11.5.1 Checks on VMX Controls

VM entry performs checks on the various VMX controls, including those related to event injection.

If FRED transitions would be enabled following VM entry (see above), the following relaxations apply to the checks that are performed on the injected-event identification field when the valid bit (bit 31) in that field is set:

- If the field's "event type" (bits 10:8) is 7 (other event), the field's vector (bits 7:0) may have value 1 (indicating SYSCALL) or value 2 (indicating SYSENTER).[1,2] If it does, the value of the VM-entry instruction-length field must be in the range 0–15.[3] Although execution of SYSCALL is limited to 64-bit mode (see Section 8.4), VM entry to compatibility mode (with FRED transitions enabled) allows injection of SYSCALL.
- If the deliver-error-code bit (bit 11) is 1, the field's "event type" must be 3 (hardware exception) and bits 31:16 of the injected-event error-code field must be 0.[4]

Regardless of whether FRED transitions would be enabled following VM entry, processors with VMX nested-exception support (Section 11.2) apply the following relaxation to checks on the injected-event field when the valid bit (bit 31) in that field is set: if the field's "event type" (bits 10:8) is 3 (hardware exception), bit 13 of the field may have value 1 (indicating a nested exception).

11.5.2 State Checking by VM Entries

Support for FRED transitions impacts VM-entry state checking in the following ways:

- The ways in which host FRED state is checked (Section 11.5.2.1).
- The ways in which guest FRED state is checked (Section 11.5.2.2).

1. Prior to the introduction of FRED transitions, the vector value had to be zero, indicating "pending MTF VM exit."
2. Processors that support neither FRED transitions nor 1-setting of the "monitor trap flag" VM-execution control treat event type 7 as reserved. A processor that supports FRED transitions but not the 1-setting of "monitor trap flag" does not reserve event type 7, but VM entry on such a processor will fail when injecting an event with type 7 and vector 0.
3. A VM-entry instruction length of 0 is allowed only if IA32_VMX_MISC[30] is read as 1. It is expected that any processor that supports FRED transitions will enumerate that bit as 1.
4. This is the same as existing behavior when IA32_VMX_BASIC[56] is read as 1. All processors that support FRED transitions will enumerate that bit as 1.

- New checks on existing guest state if FRED transitions would be enabled after VM entry (Section 11.5.2.3).

11.5.2.1 State Checking of Host FRED State

If the "load FRED" VM-exit control is 1, VM entries check the host FRED state in the VMCS that was identified in Section 11.3.1. If the field for any MSR contains a value that is not valid for that MSR (see Section 4.3), VM entry fails as is normally the case when checking host state. The following items provide specifics of the properties that must hold:

- IA32_FRED_CONFIG: Bit 2, bits 5:4, and bit 11 of the field must be zero. The upper bits of the field must be such that the field's value is canonical relative to the processor's maximum linear-address width.
- IA32_FRED_RSP1–IA32_FRED_RSP3: The value of each of these fields must be canonical relative to the processor's maximum linear-address width, and bits 5:0 of each field must be zero.
- IA32_FRED_SSP1–IA32_FRED_SSP3: The value of each of these fields must be canonical relative to the processor's maximum linear-address width, and bits 2:0 of each field must be zero.

If the "host address-space size" VM-exit control is 0, bit 32 of the host CR4 field (corresponding to CR4.FRED) must be 0.

11.5.2.2 State Checking of Guest FRED State

If the "load FRED" VM-entry control is 1, VM entries check the guest FRED state in the VMCS that was identified in Section 11.3.2. If the field for any MSR contains a value that is not valid for that MSR, VM entry fails as is normally the case when checking guest state. The following items provide specifics of the properties that must hold:

- IA32_FRED_CONFIG: Bit 2, bits 5:4, and bit 11 of the field must be zero. The upper bits of the field must be such that its value is canonical relative to the processor's maximum linear-address width.
- IA32_FRED_RSP1–IA32_FRED_RSP3: The value of each of these fields must be canonical relative to the processor's maximum linear-address width, and bits 5:0 of each field must be zero.
- IA32_FRED_SSP1–IA32_FRED_SSP3: The value of each of these fields must be canonical relative to the processor's maximum linear-address width, and bits 2:0 of each field must be zero.

If the "IA-32e mode guest" VM-entry control is 0, bit 32 of the guest CR4 field (corresponding to CR4.FRED) must be 0.

11.5.2.3 State Checking If FRED Transitions Would Be Enabled After VM Entry

As noted elsewhere, execution is constrained when FRED transitions are enabled: software cannot enter ring 1 or ring 2, nor can it enter ring 0 in compatibility mode; in addition, IOPL must be 0 when CPL is 3. Checks are added to VM entry to enforce these limitations.

If FRED transitions would be enabled following VM entry (see above), VM entry performs the following checks on guest state in the VMCS:

- The DPL value in the SS attributes field (bits 6:5) must be 0 or 3.[1]
- If the DPL value in the SS attributes field is 0, the L bit in the CS attributes field (bit 13) must be 1.
- If the DPL value in the SS attributes field is 3, the IOPL value in the RFLAGS field (bits 13:12) and the "blocking by STI" bit in the interruptibility-state field (bit 0) must both be 0.

If any of these checks fail, VM entry fails as is normally the case when checking guest state.

11.5.3 State Loading by VM Entries

If the "load FRED" VM-entry control is 1, VM entries load the guest FRED state identified in Section 11.3.2 from the VMCS.

11.5.4 VM-Entry Event Injection

If the valid bit of the injected-event identification field (bit 31) is 1, VM entry injects an event.

For IDT event delivery, VM entry can be used to inject an external interrupt, non-maskable interrupt (NMI), exception (including those caused by INT1, INT3, and INTO), or software interrupt. If FRED transitions will be enabled after the VM entry (see above), injection of any of these events uses FRED event delivery (instead of IDT event delivery).

In addition, as noted in Section 11.5.1, VM entry can inject an event for SYSCALL or SYSENTER if FRED transitions will be enabled after the VM entry.

The remainder of this section provides details of injection of events using FRED event delivery.

Parts of FRED event delivery depend on the CPL at the time the event occurred. For an event injected by VM entry, the value used for CPL is that which VM entry loaded for DPL in the access-rights field for SS in the guest-state area of the VMCS.

If bit 13 of the injected-event identification field is 1 (implying that the event is an exception; see Section 11.5.1), the event's stack level is determined as if the event had been encountered during delivery of another event (see Section 5.1.2).

Section 5.2.1 specifies the event data that FRED event delivery of certain events saves on the stack. For an event injected by VM entry, the event data saved is the value of the injected-event-data field in the VMCS. This value is used instead of what is specified in Section 5.2.1 and is done for all injected events.

Section 5.2.1 specifies that FRED event delivery of exceptions saves as part of event information a bit indicating whether the exception had been encountered during delivery of another event. For such an event injected by VM entry, the bit saved is the value of bit 13 of the injected-event identification field.

1. SS.DPL corresponds to the logical processor's current privilege level (CPL).

Section 5.2.1 specifies an instruction length that FRED event delivery of certain events saves as part of event information. For such an event injected by VM entry, the instruction length saved is the value of the VM-entry instruction-length field in the VMCS.

The following items describe the existing treatment of RIP by VM-entry event injection:

1. If VM entry successfully injects (with no nested exception) an event with type external interrupt, NMI, or hardware exception, the guest RIP (as loaded from the VMCS) is pushed on the stack.
2. If VM entry successfully injects (with no nested exception) an event with type software interrupt, privileged software exception, or software exception, the guest RIP is incremented by the VM-entry instruction length before being pushed on the stack.
3. If VM entry encounters an exception while injecting an event and delivery of that exception does not cause a VM exit, the guest RIP (as loaded from the VMCS) is pushed on the stack regardless of event type or VM-entry instruction length.
4. If VM entry encounters a VM exit while injecting an event (perhaps due to an exception), the RIP value saved by the VM exit is the guest RIP (as loaded from the VMCS). If the injected event had type software interrupt, privileged software exception, or software exception, the value saved for the VM-exit instruction length is the VM-entry instruction length.

Item #2 of this existing treatment will apply also if VM entry is injecting SYSCALL or SYSENTER. For item #4, the treatment of the instruction length is extended to apply also to the injection of SYSCALL and SYSENTER.

If VM entry is injecting an NMI, physical-NMI blocking is not changed, but virtual NMIs are blocked if the "virtual NMIs" VM-execution control is 1.

The following items provide details of how FRED event delivery of an event injected by VM entry sets bits in the augmented CS and the augmented SS saved on the stack (see Section 5.2.1):[1]

- Bits 17:16 of the augmented CS are set to a value that depends on the CPL from which the event is being delivered (see above):
  - If the CPL is 0, bits 17:16 are set to the value that would have been in IA32_FRED_CONFIG[1:0] had VM entry completed without injecting an event.[2]
  - If the CPL is 3, bits 17:16 are zero.
- Bit 18 of the augmented CS is set to a value that depends on the CPL from which the event is being delivered and the value of bit 23 (CET) is 1 in the guest CR4 in the VMCS:
  - If the CPL is 3 or the guest CR4.CET is 0, bit 18 is 0.
  - If the CPL is 0 and the guest CR4.CET is 1, the bit is set to the value of IA32_S_CET.ENDBR AND IA32_S_CET.TRACKER, referring to the value that the IA32_S_CET MSR will have following VM entry.

1. If supervisor shadow stacks will be enabled, the same augmented CS is pushed on the shadow stack; see Section 5.2.2.
2. If the "load FRED" VM-entry control is 1, this will be bits 1:0 of the IA32_FRED_CONFIG field in the guest-state area of the VMCS, unless the IA32_FRED_CONFIG MSR was subsequently loaded from the VM-entry MSR-load area (in which case it will be bits 1:0 of the value loaded from memory).

- Bit 16 of the augmented SS is set to the value of bit 0 of the interruptibility-state field in the guest-state area of the VMCS (that bit indicates blocking by STI). (There will be no blocking by STI after FRED event delivery.)
- Bit 17 of the augmented SS is set if and only if the value of bits 10:8 in the injected-event identification field in the VMCS is 4 (indicating event type software interrupt) or 7 (indicating event type "other event," used for SYSCALL and SYSENTER).[1]
- Bit 18 of the augmented SS is set if and only if the value of bits 10:8 in the injected-event identification field in the VMCS is 2 (indicating event type NMI).

Section 5.2.1 specifies that an error code is pushed in the last 8 bytes pushed (bytes 7:0). (The error code is defined only for certain exceptions and is otherwise zero.) If bit 11 of the injected-event identification field is 1, indicating that the event is an exception with an error code, the value pushed for the error code is that of the injected-event error code field (zero extended); otherwise, the value pushed is zero.

Section 5.3 specifies that FRED event delivery drops any debug traps that were pending at the time the original event occurred. This means that VM entry will ignore the contents of the pending debug exceptions field in the guest-state area of the VMCS when injecting an event to be delivered with FRED event delivery. This differs from IDT delivery of software interrupts or software exceptions injected by VM entry, which may retain such exceptions as pending after event delivery.

11.6 FRED Transitions and VM Exits

This section describes the interactions between VM exits and various aspects of FRED transitions.

11.6.1 State Management by VM Exits

If the "save FRED" VM-exit control is 1, VM exits save the guest FRED state identified in Section 11.3.2 into the VMCS.

If the "load FRED" VM-exit control is 1, VM exits load the host FRED state identified in Section 11.3.1 from the VMCS.

If the "host address-space size" VM-exit control is 0, VM exits force CR4.FRED to 0 regardless of the value of the host CR4 field in the VMCS.

11.6.2 VM Exits Caused by Events That Would be Delivered by FRED

If an event that would use FRED event delivery instead causes VM exit, information about the event is saved into the exiting-event identification and error-code fields of the VMCS as would be done if FRED transitions were not enabled, with the following exceptions:

- Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack (even without FRED transitions enabled). If bit 11 is set to 1, the error code is placed in the exiting-event error-code field. If the bit is cleared to 0, the value of the exiting-event error code is undefined.[2]

1. Event type 7 may also be used for MTF VM exits, but these are not delivered using FRED event delivery.

- Bit 13 of the exiting-event identification field is set if the VM exit is due to a nested exception encountered during delivery of an earlier event. Other VM exits (including VM exit due to #DF) clear the bit.[1]

For some events for which event data is defined (see Section 5.2.1), the event data is saved in the exit-qualification field. (This is done for #DB, #PF, and NMI.)

Note: VM exits due to #NM do not save the event data into the VMCS. Any event data is recorded in the IA32_XFD_ERR MSR.

11.6.3 VM Exits During FRED Event Delivery

A VM exit may occur during FRED event delivery, due either to a nested exception (configured to cause a VM exit) or to some VMX-specific occurrence (e.g., an EPT violation). The treatment in this section applies to any event delivered using FRED event delivery, including executions of SYSCALL and SYSENTER.

The VMX architecture treats such a case in the same way that it would treat a VM exit incident to IDT event delivery. Specifically, no processor state is updated by the FRED event delivery that encountered the VM exit. (The VM exit may occur after there have been writes to memory, e.g., to push data on the stack.)

In these cases, information about the event is saved into the original-event identification, error-code, and event-data fields of the VMCS. If the event had been injected as part of VM entry, these fields receive the values of the corresponding injected-event fields. Otherwise, they are established as would be done if FRED transitions were not enabled, with the following exceptions:

- Bit 11 of the original-event identification field is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack (even without FRED transitions enabled). If bit 11 is set to 1, the error code is placed in the original-event error-code field. If the bit is cleared to 0, the value of the original-event error code is undefined.[2]
- Bit 13 of the original-event identification field is set if the original event was a nested exception encountered during delivery of another event. Other VM exits (including VM exit due to events encountered during delivery of #DF) clear the bit.[3]

The event data that FRED event delivery would save on the stack (see Section 5.2.1) is saved into the original-event-data field. This is done for all events (if FRED event delivery would have saved zero for event data, that value is saved into the VMCS).[4]

The existing treatment of VM exits encountered during IDT event delivery of events with type software interrupt (INT *n*), privileged software exception (INT1), or software exception (INT3 or INTO) is that the length of the instruction is saved into the VM-exit

2. This paragraph is not a change from existing (non-FRED) behavior. It is documented here because it is a change from earlier revisions of this document.

1. Bit 13 of the exiting-event identification field is always cleared by VM exits caused by events that would be delivered by IDT event delivery (if FRED transitions are not enabled).
2. This is the same as existing (non-FRED) behavior. It is documented here because it is a change from earlier revisions of this document.
3. Bit 13 of the original-event identification field is always cleared by VM exits during IDT event delivery (if FRED transitions are not enabled).
4. The value of the original-event-data field is undefined following all other VM exits, including those that occur during IDT event delivery.

instruction-length field in the VMCS. This treatment will apply also to FRED event delivery, including FRED event delivery of SYSCALL or SYSENTER. (See Section 11.5.4 for the case in which the FRED event delivery resulted from VM-entry event injection.)

In general, for VMX features that have special treatment during IDT event delivery (e.g., conversion of EPT violations to virtualization exceptions), that special treatment applies as well to FRED event delivery.

11.6.4 VM Exits During FRED Return Instructions

A VM exit may occur during execution of ERETS or ERETU due either to an exception (if configured to cause a VM exit) or to some VMX-specific occurrence (e.g., an EPT violation).

The VMX architecture treats this case in the same way that it generally treats VM exits incident to other instructions: for fault-like VM exits, no register state is updated.

In particular, an execution of ERETS and ERETU that causes a VM exit does not unblock NMIs (or virtual NMIs). Because of this, such a VM exit that results from a fault, EPT violation, page-modification log-full event, SPPT misconfiguration, or SPPT miss encountered by ERETS or ERETU never sets bit 12 of the exit qualification. (The processor sets this bit only for VM exits encountered by an execution of IRET that unblocks NMIs.)

11.7 Interactions with SMM-Transfer Monitors

An SMM-transfer monitor (STM) is a special-purpose VMM that operates inside SMM to support system-management functions.

An STM is activated by an execution of the VMCALL instruction in VMX root operation. STM activation updates the values of numerous processor registers, including CR4.

STM activation always clears CR4.FRED.

12 Changes to SMIs and RSM

Delivery of a system-management interrupt (SMI) saves the logical processor's pre-SMI context to a specific region within system-management RAM (SMRAM).

The SMRAM state-save region allocates 64 bits for CR4 at offset 7E40H. Prior to the introduction of FRED transitions, CR4[63:32] was not used, and SMI delivery saved only CR4[31:0]. On processors that support FRED transitions (which are enabled in CR4[32]; see Section 4.2), SMI delivery saves all 64 bits of CR4.

The RSM instruction (Resume from System Management Mode), which can be executed only in system-management mode (SMM), effects a return from SMM by restoring register state that was saved by delivery of the most recent SMI.

Prior to the introduction of FRED transitions, RSM read from SMRAM only the value of CR4[31:0] and did not restore CR4[63:32]. On processors that support FRED transitions, RSM restores all 64 bits of CR4.

If execution of RSM detects that it would restore invalid state, the logical processor enters the shutdown state and generates a special bus cycle to indicate this fact.

On processors that support FRED transitions, execution of RSM leads to shutdown if FRED transitions would be enabled after RSM (CR4.FRED would be 1) and any of the following apply:

- IA32_EFER.LMA would be 0.
- CPL would be 1 or 2.
- CPL would be 0, and the logical processor would be in compatibility mode.
- CPL would be 3, and either IOPL would be non-zero or STI blocking would be in effect.

Note: An explicit check for STI blocking might not be required for processors for which STI blocking inhibits delivery of SMIs.

A Detailed Pseudocode

The appendix gives detailed pseudocode for FRED event delivery (Appendix A.1), ERETS (Appendix A.2), and ERETU (Appendix A.3). Unless otherwise noted, the pseudocode in this appendix should be considered definitive for the FRED transitions. (This appendix does not provide all details of interactions with the VMX architecture.)

A.1 Detailed Operation of FRED Event Delivery

The appendix presents detailed pseudocode for the operation of FRED event delivery.

It does not include details of how event information is formatted or how event data is determined. These details are in Section 5.2.1.

```
// HOLD OLD STATE IN TEMPORARIES
oldRIP := RIP;
oldCS := CS;                                    // oldCS is 8 bytes; the upper 6 bytes are defined below
oldCPL := CPL;                                  // the same as oldCS[1:0]
oldRFLAGS := RFLAGS;
oldSS := SS;                                    // oldSS is 8 bytes; upper 6 bytes are zero
oldRSP := RSP;
IF CPL = 3
      THEN oldCSL := 0;
      ELSE oldCSL := CSL;
FI;
oldGSB := GS.base;
oldSSP := SSP;                                  // used only when shadow stacks are enabled
// ERETS and ERETU will restore the oldRFLAGS to RFLAGS
// Before saving, event delivery updates oldRFLAGS to set RF when appropriate
IF the event being delivered is a fault other than an instruction breakpoint OR
      the event being delivered is a trap or interrupt following partial execution of an instruction
      (e.g., between iterations of a REP-prefixed string instruction)
      THEN oldRFLAGS[16] := 1;
FI;
// Update bits above CS selector to hold additional information
// ERETS and ERETU will use these
oldCS[17:16] := oldCSL;
IF CPL = 3
      THEN oldCS[18] := 0;
      ELSE      // See Section 5.2.1 for details of how oldCS[18] is determined
            oldCS[18] = CR4.CET AND IA32_S_CET.ENDBR_EN AND IA32_S_CET.TRACKER;
FI;
// Update bits above SS selector to hold additional information
IF STI blocking was in effect when the event occurred
      THEN oldSS[16] := 1;
FI;
IF event being delivered is either SYSCALL, SYSENTER, or INT n
      THEN oldSS[17] := 1;
```

```
FI;
IF event being delivered is an NMI                    // includes VM-entry injection of NMI
      THEN oldSS[18] := 1;
FI;
// ERETS and ERETU will ignore the event information
oldSS[63:32] := event information as defined in Section 5.2.1;

// DETERMINE NEW CONTEXT
// determine new RIP
IF oldCPL = 3
      THEN newRIP := IA32_FRED_CONFIG & ~FFFH;
      ELSE newRIP := IA32_FRED_CONFIG & ~FFFH + 256;
FI;
IF newRIP is not canonical relative to the current paging mode
      THEN #GP(0);                                    // will lead eventually to shutdown or VM exit
FI;

// determine event's stack level and the new stack level for event handler
IF oldCPL = 3 AND event is not an exception nested on event delivery AND event is not #DF
      THEN eventSL := 0;
ELSE
      IF event type is external interrupt                    // oldCPL must be 0
            THEN eventSL := IA32_FRED_CONFIG[10:9];
      ELSIF event type is hardware exception, software exception, privileged software exception, or NMI
            // v = event vector (0–31); includes INT1, INT3, INTO
            THEN eventSL := IA32_FRED_STKLVLS[2v+1:2v];
            ELSE          // SYSCALL, SYSENTER, INT n with CPL = 0; do not change CSL
                  eventSL := 0;
      FI;
FI;
newCSL := MAX{eventSL, oldCSL};
// determine new RSP
IF oldCPL = 3 OR newCSL > oldCSL
      THEN newRSP := IA32_FRED_RSPi, where i = newCSL;
      ELSE      // decrement RSP as specified and align to 64 bytes
            newRSP := (RSP – (IA32_FRED_CONFIG & 1C0H)) & ~3FH;
FI;
// determine new RFLAGS; clear all bits except position 1
newRFLAGS := 2;
// if needed, determine new SSP
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
      THEN
            IF oldCPL = 3 OR newCSL > oldCSL
                  THEN
                        newSSP := IA32_FRED_SSPi, where i = newCSL;
                        IF newSSP[2] = 1        // check for 8-byte alignment
                              THEN #GP(0);
                        FI;
                  ELSE      // decrement SSP if specified
                        newSSP := SSP – (IA32_FRED_CONFIG & 8);
            FI;
```

```
FI;

// ESTABLISH NEW CONTEXT — OLD STATE WILL BE RESTORED IF THERE IS A SUBSEQUENT EXCEPTION
// update segment registers if event occurred in ring 3
IF oldCPL = 3
      THEN
            // set CS to standard values used by a 64-bit operating system
            CS.selector := IA32_STAR[47:32] & FFFCH;
            CS.base := 0;
            CS.limit := FFFFFH;
            CS.type := 11;
            CS.S := 1;
            CS.DPL := 0;
            CS.P := 1;
            CS.L := 1;
            CS.D := 0;
            CS.G := 1;
            CS.unusable := 0;
            // set SS to standard values used by a 64-bit operating system
            SS.selector := CS.selector + 8;
            SS.base := 0;
            SS.limit := FFFFFH;
            SS.type := 3;
            SS.S := 1;
            SS.DPL := 0;
            SS.P := 1;
            SS.B := 1;
            SS.G := 1;
            SS.unusable := 0;
            // swap in supervisor GS base address
            GS.base := IA32_KERNEL_GS_BASE;
            IA32_KERNEL_GS_BASE := oldGSB;
FI;
// update registers defining context
RIP := newRIP;
RFLAGS := newRFLAGS;
RSP := newRSP;
CSL := newCSL;                                          // reflected in IA32_FRED_CONFIG[1:0]
// Update SSP if supervisor shadow stacks enabled
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
      THEN SSP := newSSP;
FI;

// SAVE STATE ON STACKS
// Save return state on new regular stack; memory accesses here have supervisor privilege
push8B 00000000_00000000H;                  // First 8 bytes pushed are all zero
push8B event data as defined in Section 5.2.1;
push8B oldSS;
push8B oldRSP;
push8B oldRFLAGS;
push8B oldCS;
```

```
push8B oldRIP;
push8B errorcode;

// If supervisor shadow stacks enabled and old CPL was 0, save data on new shadow stack
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1 AND oldCPL = 0
      THEN
            store 4 bytes of zeroes to address SSP – 4;// shadow-stack access
            SSP := SSP & ~7;                          // aligns SSP to 8B boundary
            pushSS_8B oldCS;                          // full 64-bit field
            pushSS_8B oldRIP;
            pushSS_8B oldSSP;
FI;

// UPDATE ADDITIONAL STATE
// update additional CET state as needed (SSP was updated earlier)
IF CR4.CET = 1
      THEN
            IF oldCPL = 3 AND IA32_U_CET.SH_STK_EN = 1
                  THEN     // adjust so bits 63:N get the value of bit N–1
                           // N = processor's maximum linear-address width
                        IA32_PL3_SSP := LA_adjust(oldSSP);
            FI;
            IF IA32_S_CET.ENDBR_EN = 1
                  THEN IA32_S_CET[11:10] := 00b;          // IDLE with SUPPRESS = 0
            FI;
FI;
// update event-related state
clear any debug traps that were pending prior to FRED event delivery;
IF event being delivered is an NMI
      THEN block NMIs;[1]
FI;
CPL := 0;
```

1. If the NMI was being injected by VM entry, the existing treatment applies: physical-NMI blocking is not changed, but virtual NMIs are blocked if the "virtual NMIs" VM-execution control is 1.

A.2 Detailed Operation of ERETS

The appendix presents detailed pseudocode for the operation of ERETS:

```
IF CR4.FRED = 0 OR CS.L = 0 OR CPL > 0
      THEN #UD;
FI;
// pop old state from regular stack and check it
RSP := RSP + 8;                  // skip error code so that RSP references the return state
pop8B newRIP;
pop8B tempCS;                    // not used to load CS
pop8B newRFLAGS;
pop8B newRSP;
pop8B tempSS;                    // not used to load SS
IF newRIP is not canonical relative to the current paging mode OR
      tempCS & FFFFFFFF_FFF8FFFFH ≠ current CS selector OR
      newRFLAGS & FFFFFFFF_FFC2802AH ≠ 2 OR   // enforce bit 1 set; VM, reserved bits clear
      tempSS & FFF8FFFFH ≠ current SS selector OR // do not check bits 63:32
            THEN #GP(0);
FI;
// ERETS will not numerically increase stack level
newCSL := min{CSL,tempCS[17:16]};
IBT_restore := tempCS[18];
STI_block := tempSS[16];
pend_DB := tempSS[17];
NMI_unblock := tempSS[18];
// If supervisor shadow stacks are enabled, pop and check values from the shadow stack
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
      THEN
            IF SSP & 7 ≠ 0                        // require 8-byte alignment
                  THEN #CP(FAR-RET/IRET);
            FI;
            popSS_8B newSSP;
            popSS_8B checkSSLIP;
            popSS_8B checkSSCS;
            IF checkSSCS ≠ tempCS                 // 64-bit compare
                  OR checkSSLIP ≠ newRIP
                  OR newSSP & 3H ≠ 0
                  THEN #CP(FAR-RET/IRET);
            FI;
            IF newSSP not canonical relative to the processor's maximum linear-address width
                  THEN #GP(0);
            FI;
            // If the stack level is changing, compare SSP to the FRED SSP MSR for the old stack level
            IF newCSL < CSL AND IA32_FRED_SSPi ≠ SSP   // i = CSL
                  THEN #CP(FAR-RET/IRET);
            FI;
FI;

// update registers for return state
RIP := newRIP;
```

```
RFLAGS := newRFLAGS;                                    // ERETS can set RFLAGS.RF to 1
RSP := newRSP;
CSL := newCSL;                                          // reflect in IA32_FRED_CONFIG[1:0]
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1
    THEN SSP := newSSP;
FI;
IF CR4.CET = 1 AND IA32_S_CET.ENDBR_EN = 1 AND IA32_S_CET.SUPPRESS = 0 AND IBT_restore = 1
    THEN IA32_S_CET.TRACKER := 1;
FI;
// update event-related state
IF STI_block = 1 AND RFLAGS.IF = 1 AND STI blocking was not in effect prior to ERETS
    THEN establish STI blocking after ERETS;
FI;
IF pend_DB = 1 AND RFLAGS.TF =1
    THEN pend a single-step debug exception (#DB) to be delivered after ERETS;[1]
FI;
IF NMI_unblock = 1
    THEN unblock NMIs;[2]
FI;
```

1. If ERETS began execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETS, regardless of the values on the stack for RFLAGS.TF and CS.
2. If in VMX non-root operation with the 1-setting of the "virtual NMIs" VM-execution control, this step unblocks **virtual** NMIs. See Section 11.

A.3 Detailed Operation of ERETU

The appendix presents detailed pseudocode for the operation of ERETU:

```
IF CR4.FRED = 0 OR CS.L = 0 OR CPL > 0
      THEN #UD;
FI;
IF CSL > 0
      THEN #GP(0);
FI;
// pop old state from regular stack and check it
RSP := RSP + 8;                  // skip error code so that RSP references the return state
pop8B newRIP;
pop8B tempCS;
pop8B newRFLAGS;
pop8B newRSP;
pop8B tempSS;
IF tempCS & FFFFFFFF_FFFF0003H ≠ 3 OR            // enforce return to ring 3
      newRFLAGS & FFFFFFFF_FFC2B02AH ≠ 2 OR  // enforce bit 1 set; IOPL, VM, reserved bits clear
      tempSS & FFF80003H ≠ 3                         // do not check bits 63:32
            THEN #GP(0);
FI;
pend_DB := tempSS[17];
NMI_unblock := tempSS[18];
IF tempCS[15:0] = IA32_STAR[63:48] + 16 AND tempSS[15:0] = IA32_STAR[63:48] + 8
      THEN                                          // Return to ring 3 in standard 64-bit configuration
            // set newCS to standard values used ring 3 in 64-bit mode
                  newCS.selector := tempCS[15:0];
                  newCS.base := 0;
                  newCS.limit := FFFFFH;
                  newCS.type := 11;
                  newCS.S := 1;
                  newCS.DPL := 3;
                  newCS.P := 1;
                  newCS.L := 1;
                  newCS.D := 0;
                  newCS.G := 1;
                  newCS.unusable := 0;
            // set newSS to standard values for ring 3
                  newSS.selector := tempSS[15:0];
                  newSS.base := 0;
                  newSS.limit := FFFFFH;
                  newSS.type := 3;
                  newSS.S := 1;
                  newSS.DPL := 3;
                  newSS.P := 1;
                  newSS.B := 1;
                  newSS.G := 1;
                  newSS.unusable := 0;
ELSIF tempCS[15:0] = IA32_STAR[63:48] AND tempSS[15:0] = IA32_STAR[63:48] + 8
      THEN
```

```
                // set newCS to standard values used ring 3 in compatibility mode
                    newCS.selector := tempCS[15:0];
                    newCS.base := 0;
                    newCS.limit := FFFFFH;
                    newCS.type := 11;
                    newCS.S := 1;
                    newCS.DPL := 3;
                    newCS.P := 1;
                    newCS.L := 0;
                    newCS.D := 1;
                    newCS.G := 1;
                    newCS.unusable := 0;
                // set newSS to standard values for ring 3
                    newSS.selector := tempSS[15:0];
                    newSS.base := 0;
                    newSS.limit := FFFFFH;
                    newSS.type := 3;
                    newSS.S := 1;
                    newSS.DPL := 3;
                    newSS.P := 1;
                    newSS.B := 1;
                    newSS.G := 1;
                    newSS.unusable := 0;
            ELSE
                load newCS using tempCS[15:0];          // load each as is done by IRET, including
                load newSS using tempSS[15:0];          // checks that may lead to a fault
FI;
IF newCS.L = 1
    THEN    // return to 64-bit mode
        IF newRIP is not canonical relative to current paging mode
            THEN #GP(0);
        FI;
    ELSE    // return to compatibility mode
        newRIP[63:32] := 0;
        IF newRIP is not within newCS's limit (based on limit field and G bit)
            // newRIP is always within the limit with standard values for ring 3 in compatibility mode
            THEN #GP(0);
        FI;
        newRSP[63:32] := 0;
FI;
// If user shadow stacks are enabled, check new SSP value on return to compatibility mode
IF CR4.CET = 1 AND IA32_U_CET.SH_STK_EN = 1 AND newCS.L = 0 AND IA32_PL3_SSP[63:32] ≠ 0
    THEN #GP(0);
FI;
// If supervisor shadow stacks are enabled, compare SSP to the FRED SSP MSR for stack level 0
IF CR4.CET = 1 AND IA32_S_CET.SH_STK_EN = 1 AND IA32_FRED_SSP0 ≠ SSP
    THEN #CP(FAR-RET/IRET);
FI;

// update registers for return state
RIP := newRIP;
```

```
RFLAGS := newRFLAGS;                          // ERETU can set RFLAGS.RF to 1
RSP := newRSP;                                // load all 64 bits regardless of new mode
CS := newCS;                                  // selector and descriptor
SS := newSS;                                  // selector and descriptor
CPL := 3;
// swap GS.base and IA32_KERNEL_GS_BASE
tempGSB := GS.base;
GS.base := IA32_KERNEL_GS_BASE;
IA32_KERNEL_GS_BASE := tempGSB;
IF CR4.CET = 1 AND IA32_U_CET.SH_STK_EN = 1
     THEN SSP := IA32_PL3_SSP;
FI;

// update event-related state
IF NMI_unblock = 1
     THEN unblock NMIs;[1]
FI;
IF pend_DB = 1 AND RFLAGS.TF =1
     THEN pend a single-step debug exception (#DB) to be delivered after ERETU;[2]
FI;
```

1. If in VMX non-root operation with the 1-setting of the "virtual NMIs" VM-execution control, this step unblocks **virtual** NMIs. See Section 11.
2. If ERETU began execution with RFLAGS.TF = 1, there will always be a single-step debug exception pending after ERETU, regardless of the values on the stack for RFLAGS.TF and CS.

B Event Stack Levels

Section 5.1.2 explains that FRED event delivery determines the new stack level. This determination is based on the CPL, the CSL, the event type, the event vector, and whether the event was encountered during FRED event delivery.

The principal determination is based on CPL. Delivery of events occurring when CPL = 3 always establish stack level 0 unless an exception has been encountered during FRED event delivery.[1] In the latter case, delivery of the nested exception (or the double fault, if one was generated) determines the new stack level as is done for events occurring when CPL = 0 and is explained in the following paragraphs.

The remainder of this appendix explains the determination of the new stack level for events that occur when CPL = 0 or when there has been a nested exception. It also provides details of the event type and vector reported for all cases (e.g., even for CPL = 3).

For these remaining situations, FRED event delivery establishes the new stack level as the maximum of the CSL and **event stack level**. The event stack level is determined as follows:

- If the event is an execution of INT *n* (opcode CD *n* for 8-bit value *n*), the event stack level is 0. The event type is 4 (software interrupt) and the vector is *n*.
- If the event is an execution of SYSCALL (opcode 0F 05),or SYSENTER (opcode 0F 34), the event stack level is 0. The event type is 7 (other event) and the vector is either 1 (SYSCALL) or 2 (SYSENTER).
- If the event is an execution of INT1 (opcode F1), INT3 (opcode CC), or INTO (opcode CE), the event stack level is selected using the IA32_FRED_STKLVLS MSR:
  - For INT1, the event stack level is IA32_FRED_STKLVLS[3:2]. The event type is 5 (privileged software exception) and the vector is 1.
  - For INT3, the event stack level is IA32_FRED_STKLVLS[7:6]. The event type is 6 (software exception) and the vector is 3.
  - For INTO, the event stack level is IA32_FRED_STKLVLS[9:8]. The event type is 6 (software exception) and the vector is 4.

Note: Execution of INTO results in FRED event delivery (of exception #OF) only if RFLAGS.OF = 1. Note also that INTO cannot be executed in ring 0 when FRED transitions are enabled because INTO is not valid in 64-bit mode and enabling FRED transitions prevents entry to compatibility mode in ring 0.

- If the event is a non-maskable interrupt (NMI), the event stack level is IA32_FRED_STKLVLS[5:4]. The event type is 2 (non-maskable interrupt) and the vector is 2.
- If the event is a maskable interrupt, the event stack level is IA32_FRED_CONFIG[10:9]. The event type is 0 (external interrupt) and the vector is that of the interrupt.
- If the event is an exception with vector *n* (other than those generated by INT1, INT3, or INTO), the event stack level is IA32_FRED_STKLVLS[2*n*+1,2*n*]. The event type is 3 (hardware exception) and the vector is *n*.

1. The following faults may be encountered during FRED event delivery: #GP, #PF, #SS, or #VE. Occurrence of any of these faults may generate a #DF. In addition, FRED event delivery may encounter #MC.