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

Volume 3C: System Programming Guide, Part 3

NOTE: The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of seven volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-L*, Order Number 253666; *Instruction Set Reference M-Z*, Order Number 253667; *Instruction Set Reference*, Order Number 326018; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019. Refer to all seven volumes when evaluating your design needs.

Order Number: 326019-040US
October 2011

CHAPTER 23
INTRODUCTION TO VIRTUAL-MACHINE EXTENSIONS

23.1 OVERVIEW

This chapter describes the basics of virtual machine architecture and an overview of the virtual-machine extensions (VMX) that support virtualization of processor hardware for multiple software environments.

Information about VMX instructions is provided in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*. Other aspects of VMX and system programming considerations are described in chapters of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B*.

23.2 VIRTUAL MACHINE ARCHITECTURE

Virtual-machine extensions define processor-level support for virtual machines on IA-32 processors. Two principal classes of software are supported:

- **Virtual-machine monitors (VMM) —** A VMM acts as a host and has full control of the processor(s) and other platform hardware. A VMM presents guest software (see next paragraph) with an abstraction of a virtual processor and allows it to execute directly on a logical processor. A VMM is able to retain selective control of processor resources, physical memory, interrupt management, and I/O.
- **Guest software —** Each virtual machine (VM) is a guest software environment that supports a stack consisting of operating system (OS) and application software. Each operates independently of other virtual machines and uses on the same interface to processor(s), memory, storage, graphics, and I/O provided by a physical platform. The software stack acts as if it were running on a platform with no VMM. Software executing in a virtual machine must operate with reduced privilege so that the VMM can retain control of platform resources.

23.3 INTRODUCTION TO VMX OPERATION

Processor support for virtualization is provided by a form of processor operation called VMX operation. There are two kinds of VMX operation: VMX root operation and VMX non-root operation. In general, a VMM will run in VMX root operation and guest software will run in VMX non-root operation. Transitions between VMX root operation and VMX non-root operation are called VMX transitions. There are two kinds of VMX transitions. Transitions into VMX non-root operation are called VM entries. Transitions from VMX non-root operation to VMX root operation are called VM exits.

Processor behavior in VMX root operation is very much as it is outside VMX operation. The principal differences are that a set of new instructions (the VMX instructions) is available and that the values that can be loaded into certain control registers are limited (see Section 23.8).

Processor behavior in VMX non-root operation is restricted and modified to facilitate virtualization. Instead of their ordinary operation, certain instructions (including the new VMCALL instruction) and events cause VM exits to the VMM. Because these VM exits replace ordinary behavior, the functionality of software in VMX non-root operation is limited. It is this limitation that allows the VMM to retain control of processor resources.

There is no software-visible bit whose setting indicates whether a logical processor is in VMX non-root operation. This fact may allow a VMM to prevent guest software from determining that it is running in a virtual machine.

Because VMX operation places restrictions even on software running with current privilege level (CPL) 0, guest software can run at the privilege level for which it was originally designed. This capability may simplify the development of a VMM.

23.4 LIFE CYCLE OF VMM SOFTWARE

Figure 23-1 illustrates the life cycle of a VMM and its guest software as well as the interactions between them. The following items summarize that life cycle:

- Software enters VMX operation by executing a VMXON instruction.
- Using VM entries, a VMM can then enter guests into virtual machines (one at a time). The VMM effects a VM entry using instructions VMLAUNCH and VMRESUME; it regains control using VM exits.
- VM exits transfer control to an entry point specified by the VMM. The VMM can take action appropriate to the cause of the VM exit and can then return to the virtual machine using a VM entry.
- Eventually, the VMM may decide to shut itself down and leave VMX operation. It does so by executing the VMXOFF instruction.

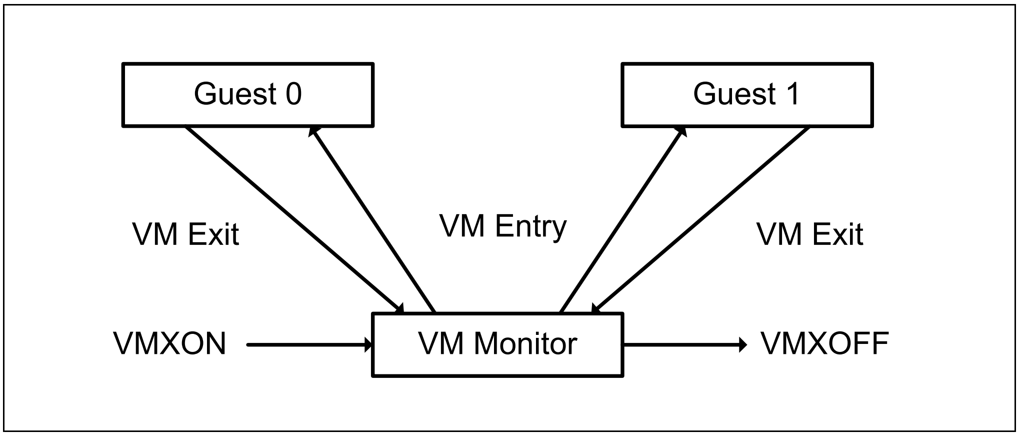

Figure 23-1. Interaction of a Virtual-Machine Monitor and Guests

23.5 VIRTUAL-MACHINE CONTROL STRUCTURE

VMX non-root operation and VMX transitions are controlled by a data structure called a virtual-machine control structure (VMCS).

Access to the VMCS is managed through a component of processor state called the VMCS pointer (one per logical processor). The value of the VMCS pointer is the 64-bit address of the VMCS. The VMCS pointer is read and written using the instructions VMPTRST and VMPTRLD. The VMM configures a VMCS using the VMREAD, VMWRITE, and VMCLEAR instructions.

A VMM could use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM could use a different VMCS for each virtual processor.

23.6 DISCOVERING SUPPORT FOR VMX

Before system software enters into VMX operation, it must discover the presence of VMX support in the processor. System software can determine whether a processor supports VMX operation using CPUID. If CPUID.1:ECX.VMX[bit 5] = 1, then VMX operation is supported. See Chapter 3, "Instruction Set Reference, A-L" of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

The VMX architecture is designed to be extensible so that future processors in VMX operation can support additional features not present in first-generation implementations of the VMX architecture. The availability of extensible VMX features is reported to software using a set of VMX capability MSRs (see Appendix A, "VMX Capability Reporting Facility").

23.7 ENABLING AND ENTERING VMX OPERATION

Before system software can enter VMX operation, it enables VMX by setting CR4.VMXE[bit 13] = 1. VMX operation is then entered by executing the VMXON instruction. VMXON causes an invalid-opcode exception (#UD) if executed with CR4.VMXE = 0. Once in VMX operation, it is not possible to clear CR4.VMXE (see Section 23.8). System software leaves VMX operation by executing the VMXOFF instruction. CR4.VMXE can be cleared outside of VMX operation after executing of VMXOFF.

VMXON is also controlled by the IA32_FEATURE_CONTROL MSR (MSR address 3AH). This MSR is cleared to zero when a logical processor is reset. The relevant bits of the MSR are:

- **Bit 0 is the lock bit.** If this bit is clear, VMXON causes a general-protection exception. If the lock bit is set, WRMSR to this MSR causes a general-protection exception; the MSR cannot be modified until a power-up reset condition. System BIOS can use this bit to provide a setup option for BIOS to disable support for VMX. To enable VMX support in a platform, BIOS must set bit 1, bit 2, or both (see below), as well as the lock bit.
- **Bit 1 enables VMXON in SMX operation.** If this bit is clear, execution of VMXON in SMX operation causes a general-protection exception. Attempts to set this bit on logical processors that do not support both VMX operation (see Section 23.6) and SMX operation (see Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*) cause general-protection exceptions.
- **Bit 2 enables VMXON outside SMX operation.** If this bit is clear, execution of VMXON outside SMX operation causes a general-protection exception. Attempts to set this bit on logical processors that do not support VMX operation (see Section 23.6) cause general-protection exceptions.

NOTE

A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

Before executing VMXON, software should allocate a naturally aligned 4-KByte region of memory that a logical processor may use to support VMX operation.[1] This region is called the **VMXON region**. The address of the VMXON region (the VMXON pointer)

1. Future processors may require that a different amount of memory be reserved. If so, this fact is reported to software using the VMX capability-reporting mechanism.

is provided in an operand to VMXON. Section 24.10.5, “VMXON Region,” details how software should initialize and access the VMXON region.

23.8 RESTRICTIONS ON VMX OPERATION

VMX operation places restrictions on processor operation. These are detailed below:

- In VMX operation, processors may fix certain bits in CR0 and CR4 to specific values and not support other values. VMXON fails if any of these bits contains an unsupported value (see “VMXON—Enter VMX Operation” in Chapter 5 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*). Any attempt to set one of these bits to an unsupported value while in VMX operation (including VMX root operation) using any of the CLTS, LMSW, or MOV CR instructions causes a general-protection exception. VM entry or VM exit cannot set any of these bits to an unsupported value.[2]

NOTES

The first processors to support VMX operation require that the following bits be 1 in VMX operation: CR0.PE, CR0.NE, CR0.PG, and CR4.VMXE. The restrictions on CR0.PE and CR0.PG imply that VMX operation is supported only in paged protected mode (including IA-32e mode). Therefore, guest software cannot be run in unpaged protected mode or in real-address mode. See Section 30.2, “Supporting Processor Operating Modes in Guest Environments,” for a discussion of how a VMM might support guest software that expects to run in unpaged protected mode or in real-address mode.

Later processors support a VM-execution control called “unrestricted guest” (see Section 24.6.2). If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation (even if the capability MSR IA32_VMX_CR0_FIXED0 reports otherwise).[3] Such processors allow guest software to run in unpaged protected mode or in real-address mode.

- VMXON fails if a logical processor is in A20M mode (see “VMXON—Enter VMX Operation” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 2B*). Once the processor is in VMX operation, A20M

2. Software should consult the VMX capability MSRs IA32_VMX_CR0_FIXED0 and IA32_VMX_CR0_FIXED1 to determine how bits in CR0 are set. (see Appendix A.7). For CR4, software should consult the VMX capability MSRs IA32_VMX_CR4_FIXED0 and IA32_VMX_CR4_FIXED1 (see Appendix A.8).

3. “Unrestricted guest” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “unrestricted guest” VM-execution control were 0. See Section 24.6.2.

interrupts are blocked. Thus, it is impossible to be in A20M mode in VMX operation.

- The INIT signal is blocked whenever a logical processor is in VMX root operation. It is not blocked in VMX non-root operation. Instead, INITs cause VM exits (see Section 25.3, “Other Causes of VM Exits”).

CHAPTER 24
VIRTUAL-MACHINE CONTROL STRUCTURES

24.1 OVERVIEW

A logical processor uses **virtual-machine control data structures** (**VMCSs**) while it is in VMX operation. These manage transitions into and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. This structure is manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

A VMM can use a different VMCS for each virtual machine that it supports. For a virtual machine with multiple logical processors (virtual processors), the VMM can use a different VMCS for each virtual processor.

A logical processor associates a region in memory with each VMCS. This region is called the **VMCS region**.[1] Software references a specific VMCS using the 64-bit physical address of the region (a **VMCS pointer**). VMCS pointers must be aligned on a 4-KByte boundary (bits 11:0 must be zero). These pointers must not set bits beyond the processor's physical-address width.[2,3]

A logical processor may maintain a number of VMCSs that are **active**. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. At any given time, at most one of the active VMCSs is the **current** VMCS. (This document frequently uses the term "the VMCS" to refer to the current VMCS.) The VMLAUNCH, VMREAD, VMRESUME, and VMWRITE instructions operate only on the current VMCS.

The following items describe how a logical processor determines which VMCSs are active and which is current:

- The memory operand of the VMPTRLD instruction is the address of a VMCS. After execution of the instruction, that VMCS is both active and current on the logical processor. Any other VMCS that had been active remains so, but no other VMCS is current.
- The memory operand of the VMCLEAR instruction is also the address of a VMCS. After execution of the instruction, that VMCS is neither active nor current on the

1. The amount of memory required for a VMCS region is at most 4 KBytes. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. If IA32_VMX_BASIC[48] is read as 1, these pointers must not set any bits in the range 63:32; see Appendix A.1.

logical processor. If the VMCS had been current on the logical processor, the logical processor no longer has a current VMCS.

The VMPTRST instruction stores the address of the logical processor's current VMCS into a specified memory location (it stores the value FFFFFFFF_FFFFFFFFH if there is no current VMCS).

The **launch state** of a VMCS determines which VM-entry instruction should be used with that VMCS: the VMLAUNCH instruction requires a VMCS whose launch state is "clear"; the VMRESUME instruction requires a VMCS whose launch state is "launched". A logical processor maintains a VMCS's launch state in the corresponding VMCS region. The following items describe how a logical processor manages the launch state of a VMCS:

- If the launch state of the current VMCS is "clear", successful execution of the VMLAUNCH instruction changes the launch state to "launched".
- The memory operand of the VMCLEAR instruction is the address of a VMCS. After execution of the instruction, the launch state of that VMCS is "clear".
- There are no other ways to modify the launch state of a VMCS (it cannot be modified using VMWRITE) and there is no direct way to discover it (it cannot be read using VMREAD).

Figure 24-1 illustrates the different states of a VMCS. It uses "X" to refer to the VMCS and "Y" to refer to any other VMCS. Thus: "VMPTRLD X" always makes X current and active; "VMPTRLD Y" always makes X not current (because it makes Y current); VMLAUNCH makes the launch state of X "launched" if X was current and its launch state was "clear"; and VMCLEAR X always makes X inactive and not current and makes its launch state "clear".

The figure does not illustrate operations that do not modify the VMCS state relative to these parameters (e.g., execution of VMPTRLD X when X is already current). Note that VMCLEAR X makes X "inactive, not current, and clear," even if X's current state is not defined (e.g., even if X has not yet been initialized). See Section 24.10.3.

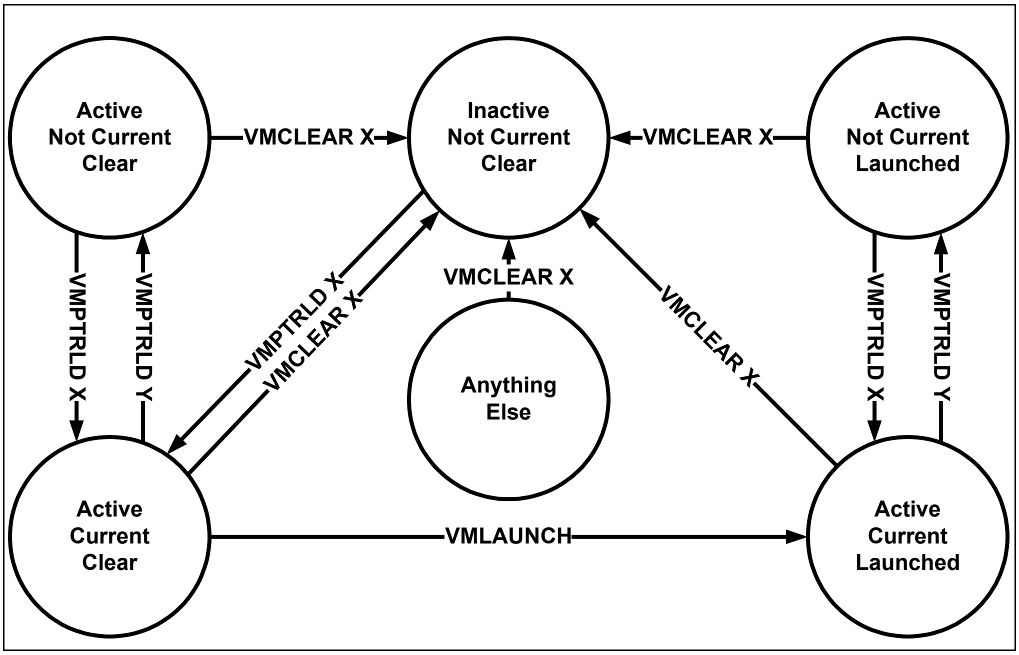

Figure 24-1. States of VMCS X

24.2 FORMAT OF THE VMCS REGION

A VMCS region comprises up to 4-KBytes.[1] The format of a VMCS region is given in Table 24-1.

Table 24-1. Format of the VMCS Region

Byte Offset	Contents
0	VMCS revision identifier
4	VMX-abort indicator
8	VMCS data (implementation-specific format)

The first 32 bits of the VMCS region contain the **VMCS revision identifier**. Processors that maintain VMCS data in different formats (see below) use different VMCS

1. The exact size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC to determine the size of the VMCS region (see Appendix A.1).

revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.[1]

Software should write the VMCS revision identifier to the VMCS region before using that region for a VMCS. The VMCS revision identifier is never written by the processor; VMPTRLD may fail if its operand references a VMCS region whose VMCS revision identifier differs from that used by the processor. Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC (see Appendix A, "VMX Capability Reporting Facility").

The next 32 bits of the VMCS region are used for the **VMX-abort indicator**. The contents of these bits do not control processor operation in any way. A logical processor writes a non-zero value into these bits if a VMX abort occurs (see Section 27.7). Software may also write into this field.

The remainder of the VMCS region is used for **VMCS data** (those parts of the VMCS that control VMX non-root operation and the VMX transitions). The format of these data is implementation-specific. VMCS data are discussed in Section 24.3 through Section 24.9. To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 24.10.4) in writeback cacheable memory. Future implementations may allow or require a different memory type[2]. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

24.3 ORGANIZATION OF VMCS DATA

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. They are read-only.

1. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions.
2. Alternatively, software may map any of these regions or structures with the UC memory type. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

The VM-execution control fields, the VM-exit control fields, and the VM-entry control fields are sometimes referred to collectively as VMX controls.

24.4 GUEST-STATE AREA

This section describes fields contained in the guest-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM entry (see Section 26.3.2) and stored into these fields on every VM exit (see Section 27.3).

24.4.1 Guest Register State

The following fields in the guest-state area correspond to processor registers:

- Control registers CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Debug register DR7 (64 bits; 32 bits on processors that do not support Intel 64 architecture).
- RSP, RIP, and RFLAGS (64 bits each; 32 bits on processors that do not support Intel 64 architecture).[1]
- The following fields for each of the registers CS, SS, DS, ES, FS, GS, LDTR, and TR:
  — Selector (16 bits).
  — Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits on processors that support Intel 64 architecture.
  — Segment limit (32 bits). The limit field is always a measure in bytes.
  — Access rights (32 bits). The format of this field is given in Table 24-2 and detailed as follows:
    - The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

- Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.[1]
- Bits 31:17 are reserved.

Table 24-2. Format of Access Rights

Bit Position(s)	Field
3:0	Segment type
4	S — Descriptor type (0 = system; 1 = code or data)
6:5	DPL — Descriptor privilege level
7	P — Segment present
11:8	Reserved
12	AVL — Available for use by system software
13	Reserved (except for CS) L — 64-bit mode active (for CS only)
14	D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
15	G — Granularity
16	Segment unusable (0 = usable; 1 = unusable)
31:17	Reserved

The base address, segment limit, and access rights compose the "hidden" part (or "descriptor cache") of each segment register. These data are included in the VMCS because it is possible for a segment register's descriptor cache to be inconsistent with the segment descriptor in memory (in the GDT or the LDT) referenced by the segment register's selector.

The value of the DPL field for SS is always equal to the logical processor's current privilege level (CPL).[2]

- The following fields for each of the registers GDTR and IDTR:

1. There are a few exceptions to this statement. For example, a segment with a non-null selector may be unusable following a task switch that fails after its commit point; see "Interrupt 10—Invalid TSS Exception (#TS)" in Section 6.14, "Exception and Interrupt Handling in 64-bit Mode," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 10-1 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

  - Base address (64 bits; 32 bits on processors that do not support Intel 64 architecture).
  - Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The following MSRs:
  - IA32_DEBUGCTL (64 bits)
  - IA32_SYSENTER_CS (32 bits)
  - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture)
  - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on logical processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-entry control.
  - IA32_PAT (64 bits). This field is supported only on logical processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
  - IA32_EFER (64 bits). This field is supported only on logical processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.
- The register SMBASE (32 bits). This register contains the base address of the logical processor's SMRAM image.

24.4.2 Guest Non-Register State

In addition to the register state described in Section 24.4.1, the guest-state area includes the following fields that characterize guest state but which do not correspond to processor registers:

- **Activity state** (32 bits). This field identifies the logical processor's activity state. When a logical processor is executing instructions normally, it is in the **active state**. Execution of certain instructions and the occurrence of certain events may cause a logical processor to transition to an **inactive state** in which it ceases to execute instructions.

  The following activity states are defined:[1]

  - 0: **Active**. The logical processor is executing instructions normally.
  - 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.

---

2. In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.

1. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 27.1.

— 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**[1] or some other serious error.

— 3: **Wait-for-SIPI**. The logical processor is inactive because it is waiting for a startup-IPI (SIPI).

Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.

- **Interruptibility state** (32 bits). The IA-32 architecture includes features that permit certain events to be blocked for a period of time. This field contains information about such blocking. Details and the format of this field are given in Table 24-3.

Table 24-3. Format of Interruptibility State

Bit Position(s)	Bit Name	Notes
0	Blocking by STI	See the "STI—Set Interrupt Flag" section in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*. Execution of STI with RFLAGS.IF = 0 blocks interrupts (and, optionally, other events) for one instruction after its execution. Setting this bit indicates that this blocking is in effect.
1	Blocking by MOV SS	See the "MOV—Move a Value from the Stack" and "POP—Pop a Value from the Stack" sections in Chapter 4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*, and Section 6.8.3 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. Execution of a MOV to SS or a POP to SS blocks interrupts for one instruction after its execution. In addition, certain debug exceptions are inhibited between a MOV to SS or a POP to SS and a subsequent instruction. Setting this bit indicates that the blocking of all these events is in effect. This document uses the term "blocking by MOV SS," but it applies equally to POP SS.
2	Blocking by SMI	See Section 29.2. System-management interrupts (SMIs) are disabled while the processor is in system-management mode (SMM). Setting this bit indicates that blocking of SMIs is in effect.

1. A triple fault occurs when a logical processor encounters an exception while attempting to deliver a double fault.

Table 24-3. Format of Interruptibility State (Contd.)

Bit Position(s)	Bit Name	Notes
3	Blocking by NMI	See Section 6.7.1 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and Section 29.8. Delivery of a non-maskable interrupt (NMI) or a system-management interrupt (SMI) blocks subsequent NMIs until the next execution of IRET. See Section 25.4 for how this behavior of IRET may change in VMX non-root operation. Setting this bit indicates that blocking of NMIs is in effect. Clearing this bit does not imply that NMIs are not (temporarily) blocked for other reasons. If the "virtual NMIs" VM-execution control (see Section 24.6.1) is 1, this bit does not control the blocking of NMIs. Instead, it refers to "virtual-NMI blocking" (the fact that guest software is not ready for an NMI).
31:4	Reserved	VM entry will fail if these bits are not 0. See Section 26.3.1.5.

- **Pending debug exceptions** (64 bits; 32 bits on processors that do not support Intel 64 architecture). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.[1] This field contains information about such exceptions. This field is described in Table 24-4.

Table 24-4. Format of Pending-Debug-Exceptions

Bit Position(s)	Bit Name	Notes
3:0	B3 – B0	When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if the corresponding enabling bit in DR7 is not set.
11:4	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5.
12	Enabled breakpoint	When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7.

1. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 6.8.3 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. In addition, certain events incident to an instruction (for example, an INIT signal) may take priority over debug traps generated by that instruction. See Table 6-2 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

Table 24-4. Format of Pending-Debug-Exceptions (Contd.)

Bit Position(s)	Bit Name	Notes
13	Reserved	VM entry fails if this bit is not 0. See Section 26.3.1.5.
14	BS	When set, this bit indicates that a debug exception would have been triggered by single-step execution mode.
63:15	Reserved	VM entry fails if these bits are not 0. See Section 26.3.1.5. Bits 63:32 exist only on processors that support Intel 64 architecture.

- **VMCS link pointer** (64 bits). This field is included for future expansion. Software should set this field to FFFFFFFF_FFFFFFFFH to avoid VM-entry failures (see Section 26.3.1.5).
- **VMX-preemption timer value** (32 bits). This field is supported only on logical processors that support the 1-setting of the "activate VMX-preemption timer" VM-execution control. This field contains the value that the VMX-preemption timer will use following the next VM entry with that setting. See Section 25.7.1 and Section 26.6.4.
- **Page-directory-pointer-table entries** (PDPTEs; 64 bits each). These four (4) fields (PDPTE0, PDPTE1, PDPTE2, and PDPTE3) are supported only on logical processors that support the 1-setting of the "enable EPT" VM-execution control. They correspond to the PDPTEs referenced by CR3 when PAE paging is in use (see Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*). They are used only if the "enable EPT" VM-execution control is 1.

24.5 HOST-STATE AREA

This section describes fields contained in the host-state area of the VMCS. As noted earlier, processor state is loaded from these fields on every VM exit (see Section 27.5).

All fields in the host-state area correspond to processor registers:

- CR0, CR3, and CR4 (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- RSP and RIP (64 bits each; 32 bits on processors that do not support Intel 64 architecture).
- Selector fields (16 bits each) for the segment registers CS, SS, DS, ES, FS, GS, and TR. There is no field in the host-state area for the LDTR selector.
- Base-address fields for FS, GS, TR, GDTR, and IDTR (64 bits each; 32 bits on processors that do not support Intel 64 architecture).

- The following MSRs:
  - IA32_SYSENTER_CS (32 bits)
  - IA32_SYSENTER_ESP and IA32_SYSENTER_EIP (64 bits; 32 bits on processors that do not support Intel 64 architecture).
  - IA32_PERF_GLOBAL_CTRL (64 bits). This field is supported only on logical processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-exit control.
  - IA32_PAT (64 bits). This field is supported only on logical processors that support either the 1-setting of the "load IA32_PAT" VM-exit control.
  - IA32_EFER (64 bits). This field is supported only on logical processors that support either the 1-setting of the "load IA32_EFER" VM-exit control.

In addition to the state identified here, some processor state components are loaded with fixed values on every VM exit; there are no fields corresponding to these components in the host-state area. See Section 27.5 for details of how state is loaded on VM exits.

24.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 24.6.1 through Section 24.6.8.

24.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts).[1] Table 24-5 lists the controls. See Chapter 25 for how these controls affect processor behavior in VMX non-root operation.

1. Some asynchronous events cause VM exits regardless of the settings of the pin-based VM-execution controls (see Section 25.3).

Table 24-5. Definitions of Pin-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	External-interrupt exiting	If this control is 1, external interrupts cause VM exits. Otherwise, they are delivered normally through the guest interrupt-descriptor table (IDT). If this control is 1, the value of RFLAGS.IF does not affect interrupt blocking.
3	NMI exiting	If this control is 1, non-maskable interrupts (NMIs) cause VM exits. Otherwise, they are delivered normally using descriptor 2 of the IDT. This control also determines interactions between IRET and blocking by NMI (see Section 25.4).
5	Virtual NMIs	If this control is 1, NMIs are never blocked and the “blocking by NMI” bit (bit 3) in the interruptibility-state field indicates “virtual-NMI blocking” (see Table 24-3). This control also interacts with the “NMI-window exiting” VM-execution control (see Section 24.6.2). This control can be set only if the “NMI exiting” VM-execution control (above) is 1.
6	Activate VMX-preemption timer	If this control is 1, the VMX-preemption timer counts down in VMX non-root operation; see Section 25.7.1. A VM exit occurs when the timer counts down to zero; see Section 25.3.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PINBASED_CTLS and IA32_VMX_TRUE_PINBASED_CTLS (see Appendix A.3.1) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 2, and 4. The VMX capability MSR IA32_VMX_PINBASED_CTLS will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PINBASED_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

24.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions.[1] These are the **primary processor-based VM-execution controls** and the **secondary processor-based VM-execution controls**.

Table 24-6 lists the primary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
2	Interrupt-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if RFLAGS.IF = 1 and there are no other blocking of interrupts (see Section 24.4.2).
3	Use TSC offsetting	This control determines whether executions of RDTSC, executions of RDTSCP, and executions of RDMSR that read from the IA32_TIME_STAMP_COUNTER MSR return a value modified by the TSC offset field (see Section 24.6.5 and Section 25.4).
7	HLT exiting	This control determines whether executions of HLT cause VM exits.
9	INVLPG exiting	This determines whether executions of INVLPG cause VM exits.
10	MWAIT exiting	This control determines whether executions of MWAIT cause VM exits.
11	RDPMC exiting	This control determines whether executions of RDPMC cause VM exits.
12	RDTSC exiting	This control determines whether executions of RDTSC and RDTSCP cause VM exits.
15	CR3-load exiting	In conjunction with the CR3-target controls (see Section 24.6.7), this control determines whether executions of MOV to CR3 cause VM exits. See Section 25.1.3. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
16	CR3-store exiting	This control determines whether executions of MOV from CR3 cause VM exits. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
19	CR8-load exiting	This control determines whether executions of MOV to CR8 cause VM exits. This control must be 0 on processors that do not support Intel 64 architecture.

1. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 25.1.2), as do task switches (see Section 25.3).

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
20	CR8-store exiting	This control determines whether executions of MOV from CR8 cause VM exits. This control must be 0 on processors that do not support Intel 64 architecture.
21	Use TPR shadow	Setting this control to 1 activates the TPR shadow, which is maintained in a page of memory addressed by the virtual-APIC address. See Section 25.4. This control must be 0 on processors that do not support Intel 64 architecture.
22	NMI-window exiting	If this control is 1, a VM exit occurs at the beginning of any instruction if there is no virtual-NMI blocking (see Section 24.4.2). This control can be set only if the “virtual NMIs” VM-execution control (see Section 24.6.1) is 1.
23	MOV-DR exiting	This control determines whether executions of MOV DR cause VM exits.
24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits. This control is ignored if the “use I/O bitmaps” control is 1.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 24.6.4 and Section 25.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.
27	Monitor trap flag	If this control is 1, the monitor trap flag debugging feature is enabled. See Section 25.7.2.
28	Use MSR bitmaps	This control determines whether MSR bitmaps are used to control execution of the RDMSR and WRMSR instructions (see Section 24.6.9 and Section 25.1.3). For this control, “0” means “do not use MSR bitmaps” and “1” means “use MSR bitmaps.” If the MSR bitmaps are not used, all executions of the RDMSR and WRMSR instructions cause VM exits.
29	MONITOR exiting	This control determines whether executions of MONITOR cause VM exits.

Table 24-6. Definitions of Primary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
30	PAUSE exiting	This control determines whether executions of PAUSE cause VM exits.
31	Activate secondary controls	This control determines whether the secondary processor-based VM-execution controls are used. If this control is 0, the logical processor operates as if all the secondary processor-based VM-execution controls were also 0.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_PROCBASED_CTLS and IA32_VMX_TRUE_PROCBASED_CTLS (see Appendix A.3.2) to determine how to set reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.1).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 1, 4–6, 8, 13–16, and 26. The VMX capability MSR IA32_VMX_PROCBASED_CTLS will always report that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_PROCBASED_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

Bit 31 of the primary processor-based VM-execution controls determines whether the secondary processor-based VM-execution controls are used. If that bit is 0, VM entry and VMX non-root operation function as if all the secondary processor-based VM-execution controls were 0. Processors that support only the 0-setting of bit 31 of the primary processor-based VM-execution controls do not support the secondary processor-based VM-execution controls.

Table 24-7 lists the secondary processor-based VM-execution controls. See Chapter 25 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-7. Definitions of Secondary Processor-Based VM-Execution Controls

Bit Position(s)	Name	Description
0	Virtualize APIC accesses	If this control is 1, a VM exit occurs on any attempt to access data on the page with the APIC-access address. See Section 25.2.
1	Enable EPT	If this control is 1, extended page tables (EPT) are enabled. See Section 28.2.
2	Descriptor-table exiting	This control determines whether executions of LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, and STR cause VM exits.
3	Enable RDTSCP	If this control is 0, any execution of RDTSCP causes an invalid-opcode exception (#UD).

Table 24-7. Definitions of Secondary Processor-Based VM-Execution Controls (Contd.)

Bit Position(s)	Name	Description
4	Virtualize x2APIC mode	Setting this control to 1 causes RDMSR and WRMSR to MSR 808H to use the TPR shadow, which is maintained on the virtual-APIC page. See Section 25.4.
5	Enable VPID	If this control is 1, cached translations of linear addresses are associated with a virtual-processor identifier (VPID). See Section 28.1.
6	WBINVD exiting	This control determines whether executions of WBINVD cause VM exits.
7	Unrestricted guest	This control determines whether guest software may run in unpaged protected mode or in real-address mode.
10	PAUSE-loop exiting	This control determines whether a series of executions of PAUSE can cause a VM exit (see Section 24.6.13 and Section 25.1.3).
11	RDRAND exiting	This control determines whether executions of RDRAND cause VM exits.
12	Enable INVPCID	If this control is 0, any execution of INVPCID causes an invalid-opcode exception (#UD).
13	Enable VM functions	Setting this control to 1 enables use of the VMFUNC instruction in VMX non-root operation. See Section 25.7.4.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_PROCBASED_CTLS2 (see Appendix A.3.3) to determine which bits may be set to 1. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 26.2.1.1).

24.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally through the IDT, using the descriptor corresponding to the exception’s vector.

Whether a page fault (exception with vector 14) causes a VM exit is determined by bit 14 in the exception bitmap as well as the error code produced by the page fault and two 32-bit fields in the VMCS (the **page-fault error-code mask** and **page-fault error-code match)**. See Section 25.3 for details.

24.6.4 I/O-Bitmap Addresses

The VM-execution control fields include the 64-bit physical addresses of **I/O bitmaps** A and B (each of which are 4 KBytes in size). I/O bitmap A contains one bit for each I/O port in the range 0000H through 7FFFH; I/O bitmap B contains bits for ports in the range 8000H through FFFFH.

A logical processor uses these bitmaps if and only if the “use I/O bitmaps” control is 1. If the bitmaps are used, execution of an I/O instruction causes a VM exit if any bit in the I/O bitmaps corresponding to a port it accesses is 1. See Section 25.1.3 for details. If the bitmaps are used, their addresses must be 4-KByte aligned.

24.6.5 Time-Stamp Counter Offset

VM-execution control fields include a 64-bit **TSC-offset** field. If the “RDTSC exiting” control is 0 and the “use TSC offsetting” control is 1, this field controls executions of the RDTSC and RDTSCP instructions. It also controls executions of the RDMSR instruction that read from the IA32_TIME_STAMP_COUNTER MSR. For all of these, the signed value of the TSC offset is combined with the contents of the time-stamp counter (using signed addition) and the sum is reported to guest software in EDX:EAX. See Chapter 25 for a detailed treatment of the behavior of RDTSC, RDTSCP, and RDMSR in VMX non-root operation.

24.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

VM-execution control fields include **guest/host masks** and **read shadows** for the CR0 and CR4 registers. These fields control executions of instructions that access those registers (including CLTS, LMSW, MOV CR, and SMSW). They are 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

In general, bits set to 1 in a guest/host mask correspond to bits “owned” by the host:

- Guest attempts to set them (using CLTS, LMSW, or MOV to CR) to values differing from the corresponding bits in the corresponding read shadow cause VM exits.
- Guest reads (using MOV from CR or SMSW) return values for these bits from the corresponding read shadow.

Bits cleared to 0 correspond to bits “owned” by the guest; guest attempts to modify them succeed and guest reads return values for these bits from the control register itself.

See Chapter 25 for details regarding how these fields affect VMX non-root operation.

24.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 **CR3-target values** and a **CR3-target count**. The CR3-target values each have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not. The CR3-target count has 32 bits on all processors.

An execution of MOV to CR3 in VMX non-root operation does not cause a VM exit if its source operand matches one of these values. If the CR3-target count is *n*, only the first *n* CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit

There are no limitations on the values that can be written for the CR3-target values. VM entry fails (see Section 26.2) if the CR3-target count is greater than 4.

Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine the number of values supported.

24.6.8 Controls for APIC Accesses

There are three mechanisms by which software accesses registers of the logical processor's local APIC:

- If the local APIC is in xAPIC mode, it can perform memory-mapped accesses to addresses in the 4-KByte page referenced by the physical address in the IA32_APIC_BASE MSR (see Section 10.4.4, "Local APIC Status and Location" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* and *Intel® 64 Architecture Processor Topology Enumeration*).[1]
- If the local APIC is in x2APIC mode, it can accesses the local APIC's registers using the RDMSR and WRMSR instructions (see *Intel® 64 Architecture Processor Topology Enumeration*).
- In 64-bit mode, it can access the local APIC's task-priority register (TPR) using the MOV CR8 instruction.

There are three processor-based VM-execution controls (see Section 24.6.2) that control such accesses. There are "use TPR shadow", "virtualize APIC accesses", and "virtualize x2APIC mode". These controls interact with the following fields:

- **APIC-access address** (64 bits). This field contains the physical address of the 4-KByte **APIC-access page**. If the "virtualize APIC accesses" VM-execution control is 1, operations that access this page may cause VM exits. See Section 25.2 and Section 25.5.

  The APIC-access address exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.
- **Virtual-APIC address** (64 bits). This field contains the physical address of the 4-KByte **virtual-APIC page**.

  If the "use TPR shadow" VM-execution control is 1, the virtual-APIC address must be 4-KByte aligned. The virtual-APIC page is accessed by the following operations if the "use TPR shadow" VM-execution control is 1:

  — The MOV CR8 instructions (see Section 25.1.3 and Section 25.4).

  — Accesses to byte 80H on the APIC-access page if, in addition, the "virtualize APIC accesses" VM-execution control is 1 (see Section 25.5.3).

1. If the local APIC does not support x2APIC mode, it is always in xAPIC mode.

  — The RDMSR and WRMSR instructions if, in addition, the value of ECX is 808H (indicating the TPR MSR) and the "virtualize x2APIC mode" VM-execution control is 1 (see Section 25.4).

  The virtual-APIC address exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which the TPR shadow (bits 7:4 of byte 80H of the virtual-APIC page) cannot fall. A VM exit occurs after an operation (e.g., an execution of MOV to CR8) that reduces the TPR shadow below this value. See Section 25.4 and Section 25.5.3.

  The TPR threshold exists only on processors that support the 1-setting of the "use TPR shadow" VM-execution control.

24.6.9 MSR-Bitmap Address

On processors that support the 1-setting of the "use MSR bitmaps" VM-execution control, the VM-execution control fields include the 64-bit physical address of four contiguous **MSR bitmaps**, which are each 1-KByte in size. This field does not exist on processors that do not support the 1-setting of that control. The four bitmaps are:

- **Read bitmap for low MSRs** (located at the MSR-bitmap address)**.** This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Read bitmap for high MSRs** (located at the MSR-bitmap address plus 1024)**.** This contains one bit for each MSR address in the range C0000000H toC0001FFFH. The bit determines whether an execution of RDMSR applied to that MSR causes a VM exit.
- **Write bitmap for low MSRs** (located at the MSR-bitmap address plus 2048)**.** This contains one bit for each MSR address in the range 00000000H to 00001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.
- **Write bitmap for high MSRs** (located at the MSR-bitmap address plus 3072)**.** This contains one bit for each MSR address in the range C0000000H toC0001FFFH. The bit determines whether an execution of WRMSR applied to that MSR causes a VM exit.

A logical processor uses these bitmaps if and only if the "use MSR bitmaps" control is 1. If the bitmaps are used, an execution of RDMSR or WRMSR causes a VM exit if the value of RCX is in neither of the ranges covered by the bitmaps or if the appropriate bit in the MSR bitmaps (corresponding to the instruction and the RCX value) is 1. See Section 25.1.3 for details. If the bitmaps are used, their address must be 4-KByte aligned.

24.6.10 Executive-VMCS Pointer

The executive-VMCS pointer is a 64-bit field used in the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). SMM VM exits save this field as described in Section 29.15.2. VM entries that return from SMM use this field as described in Section 29.15.4.

24.6.11 Extended-Page-Table Pointer (EPTP)

The **extended-page-table pointer** (EPTP) contains the address of the base of EPT PML4 table (see Section 28.2.2), as well as other EPT configuration information. The format of this field is shown in Table 24-8.

Table 24-8. Format of Extended-Page-Table Pointer

Bit Position(s)	Field
2:0	EPT paging-structure memory type (see Section 28.2.4): 0 = Uncacheable (UC) 6 = Write-back (WB) Other values are reserved.[1]
5:3	This value is 1 less than the EPT page-walk length (see Section 28.2.2)
11:6	Reserved
N–1:12	Bits N–1:12 of the physical address of the 4-KByte aligned EPT PML4 table[2]
63:N	Reserved

NOTES:

1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT paging-structure memory types are supported.
2. N is the physical-address width supported by the logical processor. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

The EPTP exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

24.6.12 Virtual-Processor Identifier (VPID)

The **virtual-processor identifier** (VPID) is a 16-bit field. It exists only on processors that support the 1-setting of the "enable VPID" VM-execution control. See Section 28.1 for details regarding the use of this field.

24.6.13 Controls for PAUSE-Loop Exiting

On processors that support the 1-setting of the "PAUSE-loop exiting" VM-execution control, the VM-execution control fields include the following 32-bit fields:

- **PLE_Gap**. Software can configure this field as an upper bound on the amount of time between two successive executions of PAUSE in a loop.
- **PLE_Window**. Software can configure this field as an upper bound on the amount of time a guest is allowed to execute in a PAUSE loop.

These fields measure time based on a counter that runs at the same rate as the timestamp counter (TSC). See Section 25.1.3 for more details regarding PAUSE-loop exiting.

24.6.14 VM-Function Controls

The **VM-function controls** constitute a 64-bit vector that governs use of the VMFUNC instruction in VMX non-root operation. This field is supported only on processors that support the 1-settings of both the "activate secondary controls" primary processor-based VM-execution control and the "enable VM functions" secondary processor-based VM-execution control.

Table 24-9 lists the VM-function controls. See Section 25.7.4 for more details of how these controls affect processor behavior in VMX non-root operation.

Table 24-9. Definitions of VM-Function Controls

Bit Position(s)	Name	Description
0	EPTP switching	The EPTP-switching VM function changes the EPT pointer to a value chosen from the EPTP list. See Section 25.7.4.3.

All other bits in this field are reserved to 0. Software should consult the VMX capability MSR IA32_VMX_VMFUNC (see Appendix A.11) to determine which bits are reserved. Failure to clear reserved bits causes subsequent VM entries to fail (see Section 26.2.1.1).

Processors that support the 1-setting of the "EPTP switching" VM-function control also support a 64-bit field called the **EPTP-list address**. This field contains the physical address of the 4-KByte **EPTP list**. The EPTP list comprises 512 8-Byte entries (each an EPTP value) and is used by the EPTP-switching VM function (see Section 25.7.4.3).

24.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 24.7.1 and Section 24.7.2.

24.7.1 VM-Exit Controls

The **VM-exit controls** constitute a 32-bit vector that governs the basic operation of VM exits. Table 24-10 lists the controls supported. See Chapter 27 for complete details of how these controls affect VM exits.

Table 24-10. Definitions of VM-Exit Controls

Bit Position(s)	Name	Description
2	Save debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are saved on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	Host address-space size	On processors that support Intel 64 architecture, this control determines whether a logical processor is in 64-bit mode after the next VM exit. Its value is loaded into CS.L, IA32_EFER.LME, and IA32_EFER.LMA on every VM exit.[1] This control must be 0 on processors that do not support Intel 64 architecture.
12	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM exit.
15	Acknowledge interrupt on exit	This control affects VM exits due to external interrupts: ▪ If such a VM exit occurs and this control is 1, the logical processor acknowledges the interrupt controller, acquiring the interrupt's vector. The vector is stored in the VM-exit interruption-information field, which is marked valid. ▪ If such a VM exit occurs and this control is 0, the interrupt is not acknowledged and the VM-exit interruption-information field is marked invalid.
18	Save IA32_PAT	This control determines whether the IA32_PAT MSR is saved on VM exit.
19	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM exit.
20	Save IA32_EFER	This control determines whether the IA32_EFER MSR is saved on VM exit.
21	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM exit.
22	Save VMX-preemption timer value	This control determines whether the value of the VMX-preemption timer is saved on VM exit.

NOTES:

1. Since Intel 64 architecture specifies that IA32_EFER.LMA is always set to the logical-AND of CR0.PG and IA32_EFER.LME, and since CR0.PG is always 1 in VMX operation, IA32_EFER.LMA is always identical to IA32_EFER.LME in VMX operation.

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_EXIT_CTLS and IA32_VMX_TRUE_EXIT_CTLS (see Appendix A.4) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.2).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8, 10, 11, 13, 14, 16, and 17. The VMX capability MSR IA32_VMX_EXIT_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_EXIT_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

24.7.2 VM-Exit Controls for MSRs

A VMM may specify lists of MSRs to be stored and loaded on VM exits. The following VM-exit control fields determine how MSRs are stored on VM exits:

- **VM-exit MSR-store count** (32 bits). This field specifies the number of MSRs to be stored on VM exit. It is recommended that this count not exceed 512 bytes.[1] Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.
- **VM-exit MSR-store address** (64 bits). This field contains the physical address of the VM-exit MSR-store area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-store count. The format of each entry is given in Table 24-11. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

Table 24-11. Format of an MSR Entry

Bit Position(s)	Contents
31:0	MSR index
63:32	Reserved
127:64	MSR data

See Section 27.4 for how this area is used on VM exits.

1. Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

The following VM-exit control fields determine how MSRs are loaded on VM exits:

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512 bytes. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.[1]
- **VM-exit MSR-load address** (64 bits). This field contains the physical address of the VM-exit MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-exit MSR-load count (see Table 24-11). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 27.6 for how this area is used on VM exits.

24.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Sections 24.8.1 through 24.8.3.

24.8.1 VM-Entry Controls

The **VM-entry controls** constitute a 32-bit vector that governs the basic operation of VM entries. Table 24-12 lists the controls supported. See Chapter 26 for how these controls affect VM entries.

Table 24-12. Definitions of VM-Entry Controls

Bit Position(s)	Name	Description
2	Load debug controls	This control determines whether DR7 and the IA32_DEBUGCTL MSR are loaded on VM exit. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
9	IA-32e mode guest	On processors that support Intel 64 architecture, this control determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA as part of VM entry.[1] This control must be 0 on processors that do not support Intel 64 architecture.
10	Entry to SMM	This control determines whether the logical processor is in system-management mode (SMM) after VM entry. This control must be 0 for any VM entry from outside SMM.

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

Table 24-12. Definitions of VM-Entry Controls (Contd.)

Bit Position(s)	Name	Description
11	Deactivate dual-monitor treatment	If set to 1, the default treatment of SMIs and SMM is in effect after the VM entry (see Section 29.15.7). This control must be 0 for any VM entry from outside SMM.
13	Load IA32_PERF_GLOBAL_CTRL	This control determines whether the IA32_PERF_GLOBAL_CTRL MSR is loaded on VM entry.
14	Load IA32_PAT	This control determines whether the IA32_PAT MSR is loaded on VM entry.
15	Load IA32_EFER	This control determines whether the IA32_EFER MSR is loaded on VM entry.

NOTES:

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the "unrestricted guest" VM-execution control. If it is read as 1, every VM exit stores the value of IA32_EFER.LMA into the "IA-32e mode guest" VM-entry control (see Section 27.2).

All other bits in this field are reserved, some to 0 and some to 1. Software should consult the VMX capability MSRs IA32_VMX_ENTRY_CTLS and IA32_VMX_TRUE_ENTRY_CTLS (see Appendix A.5) to determine how it should set the reserved bits. Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 26.2.1.3).

The first processors to support the virtual-machine extensions supported only the 1-settings of bits 0–8 and 12. The VMX capability MSR IA32_VMX_ENTRY_CTLS always reports that these bits must be 1. Logical processors that support the 0-settings of any of these bits will support the VMX capability MSR IA32_VMX_TRUE_ENTRY_CTLS MSR, and software should consult this MSR to discover support for the 0-settings of these bits. Software that is not aware of the functionality of any one of these bits should set that bit to 1.

24.8.2 VM-Entry Controls for MSRs

A VMM may specify a list of MSRs to be loaded on VM entries. The following VM-entry control fields manage this functionality:

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512 bytes. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.[1]

1. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR IA32_VMX_MISC to determine the number supported (see Appendix A.6).

- **VM-entry MSR-load address** (64 bits). This field contains the physical address of the VM-entry MSR-load area. The area is a table of entries, 16 bytes per entry, where the number of entries is given by the VM-entry MSR-load count. The format of entries is described in Table 24-11. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 26.4 for details of how this area is used on VM entries.

24.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the IDT (after all guest state and MSRs have been loaded). This process is called **event injection** and is controlled by the following three VM-entry control fields:

- **VM-entry interruption-information field** (32 bits). This field provides details about the event to be injected. Table 24-13 describes the field.

Table 24-13. Format of the VM-Entry Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Reserved 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Other event
11	Deliver error code (0 = do not deliver; 1 = deliver)
30:12	Reserved
31	Valid

— The **vector** (bits 7:0) determines which entry in the IDT is used or which other event is injected.

— The **interruption type** (bits 10:8) determines details of how the injection is performed. In general, a VMM should use the type **hardware exception** for all exceptions other than breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); it should use the type **software exception** for #BP and #OF. The type **other event** is used for injection of events that are not delivered through the IDT.

— For exceptions, the **deliver-error-code bit** (bit 11) determines whether delivery pushes an error code on the guest stack.

  — VM entry injects an event if and only if the **valid bit** (bit 31) is 1. The valid bit in this field is cleared on every VM exit (see Section 27.2).

- **VM-entry exception error code** (32 bits). This field is used if and only if the valid bit (bit 31) and the deliver-error-code bit (bit 11) are both set in the VM-entry interruption-information field.
- **VM-entry instruction length** (32 bits). For injection of events whose type is software interrupt, software exception, or privileged software exception, this field is used to determine the value of RIP that is pushed on the stack.

See Section 26.5 for details regarding the mechanics of event injection, including the use of the interruption type and the VM-entry instruction length.

VM exits clear the valid bit (bit 31) in the VM-entry interruption-information field.

24.9 VM-EXIT INFORMATION FIELDS

The VMCS contains a section of read-only fields that contain information about the most recent VM exit. Attempts to write to these fields with VMWRITE fail (see "VMWRITE—Write Field to Virtual-Machine Control Structure" in Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

24.9.1 Basic VM-Exit Information

The following VM-exit information fields provide basic information about a VM exit:

- **Exit reason** (32 bits). This field encodes the reason for the VM exit and has the structure given in Table 24-14.

Table 24-14. Format of Exit Reason

Bit Position(s)	Contents
15:0	Basic exit reason
27:16	Reserved (cleared to 0)
28	Pending MTF VM exit
29	VM exit from VMX root operation
30	Reserved (cleared to 0)
31	VM-entry failure (0 = true VM exit; 1 = VM-entry failure)

  - Bits 15:0 provide basic information about the cause of the VM exit (if bit 31 is clear) or of the VM-entry failure (if bit 31 is set). Appendix C enumerates the basic exit reasons.
  - Bit 28 is set only by an SMM VM exit (see Section 29.15.2) that took priority over an MTF VM exit (see Section 25.7.2) that would have occurred had the SMM VM exit not occurred. See Section 29.15.2.3.
  - Bit 29 is set if and only if the processor was in VMX root operation at the time the VM exit occurred. This can happen only for SMM VM exits. See Section 29.15.2.
  - Because some VM-entry failures load processor state from the host-state area (see Section 26.7), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose.
- **Exit qualification** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field contains additional information about the cause of VM exits due to the following: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); task switches; INVEPT; INVLPG;INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; and MWAIT. The format of the field depends on the cause of the VM exit. See Section 27.2.1 for details.
- **Guest-linear address** (64 bits; 32 bits on processors that do not support Intel 64 architecture). This field is used in the following cases:
  - VM exits due to attempts to execute LMSW with a memory operand.
  - VM exits due to attempts to execute INS or OUTS.
  - VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions.
  - Certain VM exits due to EPT violations

  See Section 27.2.1 and Section 29.15.2.3 for details of when and how this field is used.
- **Guest-physical address** (64 bits). This field is used VM exits due to EPT violations and EPT misconfigurations. See Section 27.2.1 for details of when and how this field is used.

24.9.2 Information for VM Exits Due to Vectored Events

Event-specific information is provided for VM exits due to the following vectored events: exceptions (including those generated by the instructions INT3, INTO, BOUND, and UD2); external interrupts that occur while the "acknowledge interrupt on exit" VM-exit control is 1; and non-maskable interrupts (NMIs). This information is provided in the following fields:

- **VM-exit interruption information** (32 bits). This field receives basic information associated with the event causing the VM exit. Table 24-15 describes this field.

Table 24-15. Format of the VM-Exit Interruption-Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4 – 5: Not used 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	NMI unblocking due to IRET
30:13	Reserved (cleared to 0)
31	Valid

- **VM-exit interruption error code** (32 bits). For VM exits caused by hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

Section 27.2.2 provides details of how these fields are saved on VM exits.

24.9.3 Information for VM Exits That Occur During Event Delivery

Additional information is provided for VM exits that occur during event delivery in VMX non-root operation.[1] This information is provided in the following fields:

- **IDT-vectoring information** (32 bits). This field receives basic information associated with the event that was being delivered when the VM exit occurred. Table 24-16 describes this field.

Table 24-16. Format of the IDT-Vectoring Information Field

Bit Position(s)	Content
7:0	Vector of interrupt or exception

1. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 26.5.1.2.

Table 24-16. Format of the IDT-Vectoring Information Field (Contd.)

Bit Position(s)	Content
10:8	Interruption type: 0: External interrupt 1: Not used 2: Non-maskable interrupt (NMI) 3: Hardware exception 4: Software interrupt 5: Privileged software exception 6: Software exception 7: Not used
11	Error code valid (0 = invalid; 1 = valid)
12	Undefined
30:13	Reserved (cleared to 0)
31	Valid

- **IDT-vectoring error code** (32 bits). For VM exits the occur during delivery of hardware exceptions that would have delivered an error code on the stack, this field receives that error code.

See Section 27.2.3 provides details of how these fields are saved on VM exits.

24.9.4 Information for VM Exits Due to Instruction Execution

The following fields are used for VM exits caused by attempts to execute certain instructions in VMX non-root operation:

- **VM-exit instruction length** (32 bits). For VM exits resulting from instruction execution, this field receives the length in bytes of the instruction whose execution led to the VM exit.[1] See Section 27.2.4 for details of when and how this field is used.
- **VM-exit instruction information** (32 bits). This field is used for VM exits due to attempts to execute INS, INVEPT, INVVPID, LIDT, LGDT, LLDT, LTR, OUTS, SIDT, SGDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON.[2] The format of the field depends on the cause of the VM exit. See Section 27.2.4 for details.

1. This field is also used for VM exits that occur during the delivery of a software interrupt or software exception.
2. Whether the processor provides this information on VM exits due to attempts to execute INS or OUTS can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

The following fields (64 bits each; 32 bits on processors that do not support Intel 64 architecture) are used only for VM exits due to SMIs that arrive immediately after retirement of I/O instructions. They provide information about that I/O instruction:

- **I/O RCX**. The value of RCX before the I/O instruction started.
- **I/O RSI**. The value of RSI before the I/O instruction started.
- **I/O RDI**. The value of RDI before the I/O instruction started.
- **I/O RIP**. The value of RIP before the I/O instruction started (the RIP that addressed the I/O instruction).

24.9.5 VM-Instruction Error Field

The 32-bit **VM-instruction error field** does not provide information about the most recent VM exit. In fact, it is not modified on VM exits. Instead, it provides information about errors encountered by a non-faulting execution of one of the VMX instructions.

24.10 SOFTWARE USE OF THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when using a VMCS and related structures. It also provides descriptions of consequences for failing to follow guidelines.

24.10.1 Software Use of Virtual-Machine Control Structures

To ensure proper processor behavior, software should observe certain guidelines when using an active VMCS.

No VMCS should ever be active on more than one logical processor. If a VMCS is to be “migrated” from one logical processor to another, the first logical processor should execute VMCLEAR for the VMCS (to make it inactive on that logical processor and to ensure that all VMCS data are in memory) before the other logical processor executes VMPTRLD for the VMCS (to make it active on the second logical processor). A VMCS that is made active on more than one logical processor may become **corrupted** (see below).

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 24.10.2). Software should never access or modify the VMCS data of an active VMCS using ordinary memory operations, in part because the format used to store the VMCS data is implementation-specific and not architecturally defined, and also because a logical processor may maintain some VMCS data of an active VMCS on the processor and not in the VMCS region. The following items detail some of the hazards of accessing VMCS data using ordinary memory operations:

- Any data read from a VMCS with an ordinary memory read does not reliably reflect the state of the VMCS. Results may vary from time to time or from logical processor to logical processor.
- Writing to a VMCS with an ordinary memory write is not guaranteed to have a deterministic effect on the VMCS. Doing so may cause the VMCS to become corrupted (see below).

(Software can avoid these hazards by removing any linear-address mappings to a VMCS region before executing a VMPTRLD for that region and by not remapping it until after executing VMCLEAR for that region.)

If a logical processor leaves VMX operation, any VMCSs active on that logical processor may be corrupted (see below). To prevent such corruption of a VMCS that may be used either after a return to VMX operation or on another logical processor, software should VMCLEAR that VMCS before executing the VMXOFF instruction or removing power from the processor (e.g., as part of a transition to the S3 and S4 power states).

This section has identified operations that may cause a VMCS to become corrupted. These operations may cause the VMCS's data to become undefined. Behavior may be unpredictable if that VMCS used subsequently on any logical processor. The following items detail some hazards of VMCS corruption:

- VM entries may fail for unexplained reasons or may load undesired processor state.
- The processor may not correctly support VMX non-root operation as documented in Chapter 25 and may generate unexpected VM exits.
- VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

24.10.2 VMREAD, VMWRITE, and Encodings of VMCS Fields

Every field of the VMCS is associated with a 32-bit value that is its **encoding**. The encoding is provided in an operand to VMREAD and VMWRITE when software wishes to read or write that field. These instructions fail if given, in 64-bit mode, an operand that sets an encoding bit beyond bit 32. See Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*, for a description of these instructions.

The structure of the 32-bit encodings of the VMCS components is determined principally by the width of the fields and their function in the VMCS. See Table 24-17.

Table 24-17. Structure of VMCS Component Encoding

Bit Position(s)	Contents
0	Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural-width fields

Table 24-17. Structure of VMCS Component Encoding (Contd.)

Bit Position(s)	Contents
9:1	Index
11:10	Type: 0: control 1: read-only data 2: guest state 3: host state
12	Reserved (must be 0)
14:13	Width: 0: 16-bit 1: 64-bit 2: 32-bit 3: natural-width
31:15	Reserved (must be 0)

The following items detail the meaning of the bits in each encoding:

- **Field width.** Bits 14:13 encode the width of the field.
  - A value of 0 indicates a 16-bit field.
  - A value of 1 indicates a 64-bit field.
  - A value of 2 indicates a 32-bit field.
  - A value of 3 indicates a **natural-width** field. Such fields have 64 bits on processors that support Intel 64 architecture and 32 bits on processors that do not.

  Fields whose encodings use value 1 are specially treated to allow 32-bit software access to all 64 bits of the field. Such access is allowed by defining, for each such field, an encoding that allows direct access to the high 32 bits of the field. See below.
- **Field type.** Bits 11:10 encode the type of VMCS field: control, guest-state, host-state, or read-only data. The last category includes the VM-exit information fields and the VM-instruction error field.
- **Index.** Bits 9:1 distinguish components with the same field width and type.
- **Access type.** Bit 0 must be 0 for all fields except for 64-bit fields (those with field-width 1; see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a 64-bit field with field-width 1, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix B gives the encodings of all fields in the VMCS.

The following describes the operation of VMREAD and VMWRITE based on processor mode, VMCS-field width, and access type:

- 16-bit fields:
  - A VMREAD returns the value of the field in bits 15:0 of the destination operand; other bits of the destination operand are cleared to 0.
  - A VMWRITE writes the value of bits 15:0 of the source operand into the VMCS field; other bits of the source operand are not used.
- 32-bit fields:
  - A VMREAD returns the value of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
  - A VMWRITE writes the value of bits 31:0 of the source operand into the VMCS field; in 64-bit mode, bits 63:32 of the source operand are not used.
- 64-bit fields and natural-width fields using the full access type outside IA-32e mode.
  - A VMREAD returns the value of bits 31:0 of the field in its destination operand; bits 63:32 of the field are ignored.
  - A VMWRITE writes the value of its source operand to bits 31:0 of the field and clears bits 63:32 of the field.
- 64-bit fields and natural-width fields using the full access type in 64-bit mode (only on processors that support Intel 64 architecture).
  - A VMREAD returns the value of the field in bits 63:0 of the destination operand
  - A VMWRITE writes the value of bits 63:0 of the source operand into the VMCS field.
- 64-bit fields using the high access type.
  - A VMREAD returns the value of bits 63:32 of the field in bits 31:0 of the destination operand; in 64-bit mode, bits 63:32 of the destination operand are cleared to 0.
  - A VMWRITE writes the value of bits 31:0 of the source operand to bits 63:32 of the field; in 64-bit mode, bits 63:32 of the source operand are not used.

Software seeking to read a 64-bit field outside IA-32e mode can use VMREAD with the full access type (reading bits 31:0 of the field) and VMREAD with the high access type (reading bits 63:32 of the field); the order of the two VMREAD executions is not important. Software seeking to modify a 64-bit field outside IA-32e mode should first use VMWRITE with the full access type (establishing bits 31:0 of the field while clearing bits 63:32) and then use VMWRITE with the high access type (establishing bits 63:32 of the field).

24.10.3 Initializing a VMCS

Software should initialize fields in a VMCS (using VMWRITE) before using the VMCS for VM entry. Failure to do so may result in unpredictable behavior; for example, a VM entry may fail for unexplained reasons, or a successful transition (VM entry or VM exit) may load processor state with unexpected values.

It is not necessary to initialize fields that the logical processor will not use. (For example, it is not necessary to unitize the MSR-bitmap address if the "use MSR bitmaps" VM-execution control is 0.)

A processor maintains some VMCS information that cannot be modified with the VMWRITE instruction; this includes a VMCS's launch state (see Section 24.1). Such information may be stored in the VMCS data portion of a VMCS region. Because the format of this information is implementation-specific, there is no way for software to know, when it first allocates a region of memory for use as a VMCS region, how the processor will determine this information from the contents of the memory region.

In addition to its other functions, the VMCLEAR instruction initializes any implementation-specific information in the VMCS region referenced by its operand. To avoid the uncertainties of implementation-specific behavior, software should execute VMCLEAR on a VMCS region before making the corresponding VMCS active with VMPTRLD for the first time. (Figure 24-1 illustrates how execution of VMCLEAR puts a VMCS into a well-defined state.)

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry for the first time.
- VMLAUNCH should be used for the first VM entry using a VMCS after VMCLEAR has been executed for that VMCS.
- VMRESUME should be used for any subsequent VM entry using a VMCS (until the next execution of VMCLEAR for the VMCS).

It is expected that, in general, VMRESUME will have lower latency than VMLAUNCH. Since "migrating" a VMCS from one logical processor to another requires use of VMCLEAR (see Section 24.10.1), which sets the launch state of the VMCS to "clear", such migration requires the next VM entry to be performed using VMLAUNCH. Software developers can avoid the performance cost of increased VM-entry latency by avoiding unnecessary migration of a VMCS from one logical processor to another.

24.10.4 Software Access to Related Structures

In addition to data in the VMCS region itself, VMX non-root operation can be controlled by data structures that are referenced by pointers in a VMCS (for example, the I/O bitmaps). While the pointers to these data structures are parts of the VMCS, the data structures themselves are not. They are not accessible using VMREAD and VMWRITE but by ordinary memory writes.

Software should ensure that each such data structure is modified only when no logical processor with a current VMCS that references it is in VMX non-root operation. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 24.10.1).

24.10.5 VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region)[1] that the logical processor uses to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. The VMXON pointer is subject to the limitations that apply to VMCS pointers:

- The VMXON pointer must be 4-KByte aligned (bits 11:0 must be zero).
- The VMXON pointer must not set any bits beyond the processor's physical-address width.[2,3]

Before executing VMXON, software should write the VMCS revision identifier (see Section 24.2) to the VMXON region. It need not initialize the VMXON region in any other way. Software should use a separate region for each logical processor and should not access or modify the VMXON region of a logical processor between execution of VMXON and VMXOFF on that logical processor. Doing otherwise may lead to unpredictable behavior (including behaviors identified in Section 24.10.1).

1. The amount of memory required for the VMXON region is the same as that required for a VMCS region. This size is implementation specific and can be determined by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. If IA32_VMX_BASIC[48] is read as 1, the VMXON pointer must not set any bits in the range 63:32; see Appendix A.1.

CHAPTER 25 VMX NON-ROOT OPERATION

In a virtualized environment using VMX, the guest software stack typically runs on a logical processor in VMX non-root operation. This mode of operation is similar to that of ordinary processor operation outside of the virtualized environment. This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits (which bring a logical processor from VMX non-root operation to root operation). The differences between VMX non-root operation and ordinary processor operation are described in the following sections:

- Section 25.1, “Instructions That Cause VM Exits”
- Section 25.2, “APIC-Access VM Exits”
- Section 25.3, “Other Causes of VM Exits”
- Section 25.4, “Changes to Instruction Behavior in VMX Non-Root Operation”
- Section 25.5, “APIC Accesses That Do Not Cause VM Exits”
- Section 25.6, “Other Changes in VMX Non-Root Operation”
- Section 25.7, “Features Specific to VMX Non-Root Operation”

Chapter 24, “Virtual-Machine Control Structures,” describes the data control structures that govern VMX non-root operation. Chapter 26, “VM Entries,” describes the operation of VM entries by which the processor transitions from VMX root operation to VMX non-root operation. Chapter 27, “VM Exits,” describes the operation of VM exits by which the processor transitions from VMX non-root operation to VMX root operation.

25.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, such VM exits are “fault-like,” meaning that the instruction causing the VM exit does not execute and no processor state is updated by the instruction. Section 27.1 details architectural state in the context of a VM exit.

Section 25.1.1 defines the prioritization between faults and VM exits for instructions subject to both. Section 25.1.2 identifies instructions that cause VM exits whenever they are executed in VMX non-root operation (and thus can never be executed in VMX non-root operation). Section 25.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields (see Section 24.6).

25.1.1 Relative Priority of Faults and VM Exits

The following principles describe the ordering between existing faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level,[1] and general-protection exceptions that are based on checking I/O permission bits in the task-state segment (TSS). For example, execution of RDMSR with CPL = 3 generates a general-protection exception and not a VM exit.[2]
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see LMSW in Section 25.1.3).
- VM exits caused by execution of the INS and OUTS instructions (resulting either because the "unconditional I/O exiting" VM-execution control is 1 or because the "use I/O bitmaps control is 1) have priority over the following faults:
  - — A general-protection fault due to the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) being unusable
  - — A general-protection fault due to an offset beyond the limit of the relevant segment
  - — An alignment-check exception
- Fault-like VM exits have priority over exceptions other than those mentioned above. For example, RDMSR of a non-existent MSR with CPL = 0 generates a VM exit and not a general-protection exception.

When Section 25.1.2 or Section 25.1.3 (below) identify an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that takes priority over a VM exit.

25.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits when they are executed in VMX non-root operation: CPUID, GETSEC,[3] INVD, and XSETBV.[4] This is also true of instructions introduced with VMX, which include: INVEPT, INVVPID, VMCALL,[5] VMCLEAR,

---

1. These include faults generated by attempts to execute, in virtual-8086 mode, privileged instructions that are not recognized in that mode.
2. MOV DR is an exception to this rule; see Section 25.1.3.
3. An execution of GETSEC in VMX non-root operation causes a VM exit if CR4.SMXE[Bit 14] = 1 regardless of the value of CPL or RAX. An execution of GETSEC causes an invalid-opcode exception (#UD) if CR4.SMXE[Bit 14] = 0.
4. An execution of XSETBV in VMX non-root operation causes a VM exit if CR4.OSXSAVE[Bit 18] = 1 regardless of the value of CPL, RAX, RCX, or RDX. An execution of XSETBV causes an invalid-opcode exception (#UD) if CR4.OSXSAVE[Bit 18] = 0.

VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.

25.1.3 Instructions That Cause VM Exits Conditionally

Certain instructions cause VM exits in VMX non-root operation depending on the setting of the VM-execution controls. The following instructions can cause "fault-like" VM exits based on the conditions described:

- **CLTS**. The CLTS instruction causes a VM exit if the bits in position 3 (corresponding to CR0.TS) are set in both the CR0 guest/host mask and the CR0 read shadow.
- **HLT.** The HLT instruction causes a VM exit if the "HLT exiting" VM-execution control is 1.
- **IN, INS/INSB/INSW/INSD, OUT, OUTS/OUTSB/OUTSW/OUTSD.** The behavior of each of these instructions is determined by the settings of the "unconditional I/O exiting" and "use I/O bitmaps" VM-execution controls:
  - If both controls are 0, the instruction executes normally.
  - If the "unconditional I/O exiting" VM-execution control is 1 and the "use I/O bitmaps" VM-execution control is 0, the instruction causes a VM exit.
  - If the "use I/O bitmaps" VM-execution control is 1, the instruction causes a VM exit if it attempts to access an I/O port corresponding to a bit set to 1 in the appropriate I/O bitmap (see Section 24.6.4). If an I/O operation "wraps around" the 16-bit I/O-port space (accesses ports FFFFH and 0000H), the I/O instruction causes a VM exit (the "unconditional I/O exiting" VM-execution control is ignored if the "use I/O bitmaps" VM-execution control is 1).

  See Section 25.1.1 for information regarding the priority of VM exits relative to faults that may be caused by the INS and OUTS instructions.
- **INVLPG.** The INVLPG instruction causes a VM exit if the "INVLPG exiting" VM-execution control is 1.
- **INVPCID.** The INVPCID instruction causes a VM exit if the "INVLPG exiting" and "enable INVPCID" VM-execution controls are both 1.[1]
- **LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR.** These instructions cause VM exits if the "descriptor-table exiting" VM-execution control is 1.[2]

---

5. Under the dual-monitor treatment of SMIs and SMM, executions of VMCALL cause SMM VM exits in VMX root operation outside SMM. See Section 29.15.2.

1. "Enable INVPCID" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable INVPCID" VM-execution control were 0. See Section 24.6.2.

2. "Descriptor-table exiting" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "descriptor-table exiting" VM-execution control were 0. See Section 24.6.2.

- **LMSW.** In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding bit in the CR0 read shadow. LMSW never clears bit 0 of CR0 (CR0.PE); thus, LMSW causes a VM exit if either of the following are true:
  - The bits in position 0 (corresponding to CR0.PE) are set in both the CR0 guest/mask and the source operand, and the bit in position 0 is clear in the CR0 read shadow.
  - For any bit position in the range 3:1, the bit in that position is set in the CR0 guest/mask and the values of the corresponding bits in the source operand and the CR0 read shadow differ.
- **MONITOR.** The MONITOR instruction causes a VM exit if the "MONITOR exiting" VM-execution control is 1.
- **MOV from CR3.** The MOV from CR3 instruction causes a VM exit if the "CR3-store exiting" VM-execution control is 1. The first processors to support the virtual-machine extensions supported only the 1-setting of this control.
- **MOV from CR8.** The MOV from CR8 instruction (which can be executed only in 64-bit mode) causes a VM exit if the "CR8-store exiting" VM-execution control is 1. If this control is 0, the behavior of the MOV from CR8 instruction is modified if the "use TPR shadow" VM-execution control is 1 (see Section 25.4).
- **MOV to CR0.** The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- **MOV to CR3.** The MOV to CR3 instruction causes a VM exit unless the "CR3-load exiting" VM-execution control is 0 or the value of its source operand is equal to one of the CR3-target values specified in the VMCS. If the CR3-target count in *n*, only the first *n* CR3-target values are considered; if the CR3-target count is 0, MOV to CR3 always causes a VM exit.

  The first processors to support the virtual-machine extensions supported only the 1-setting of the "CR3-load exiting" VM-execution control. These processors always consult the CR3-target controls to determine whether an execution of MOV to CR3 causes a VM exit.
- **MOV to CR4.** The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- **MOV to CR8.** The MOV to CR8 instruction (which can be executed only in 64-bit mode) causes a VM exit if the "CR8-load exiting" VM-execution control is 1. If this control is 0, the behavior of the MOV to CR8 instruction is modified if the "use TPR shadow" VM-execution control is 1 (see Section 25.4) and it may cause a trap-like VM exit (see below).
- **MOV DR.** The MOV DR instruction causes a VM exit if the "MOV-DR exiting" VM-execution control is 1. Such VM exits represent an exception to the principles identified in Section 25.1.1 in that they take priority over the following: general-

protection exceptions based on privilege level; and invalid-opcode exceptions that occur because CR4.DE=1 and the instruction specified access to DR4 or DR5.

- **MWAIT.** The MWAIT instruction causes a VM exit if the "MWAIT exiting" VM-execution control is 1. If this control is 0, the behavior of the MWAIT instruction may be modified (see Section 25.4).
- **PAUSE.**The behavior of each of this instruction depends on CPL and the settings of the "PAUSE exiting" and "PAUSE-loop exiting" VM-execution controls**:**[1]
  - CPL = 0.
    - If the "PAUSE exiting" and "PAUSE-loop exiting" VM-execution controls are both 0, the PAUSE instruction executes normally.
    - If the "PAUSE exiting" VM-execution control is 1, the PAUSE instruction causes a VM exit (the "PAUSE-loop exiting" VM-execution control is ignored if CPL = 0 and the "PAUSE exiting" VM-execution control is 1).
    - If the "PAUSE exiting" VM-execution control is 0 and the "PAUSE-loop exiting" VM-execution control is 1, the following treatment applies.

      The logical processor determines the amount of time between this execution of PAUSE and the previous execution of PAUSE at CPL 0. If this amount of time exceeds the value of the VM-execution control field PLE_Gap, the processor considers this execution to be the first execution of PAUSE in a loop. (It also does so for the first execution of PAUSE at CPL 0 after VM entry.)

      Otherwise, the logical processor determines the amount of time since the most recent execution of PAUSE that was considered to be the first in a loop. If this amount of time exceeds the value of the VM-execution control field PLE_Window, a VM exit occurs.

      For purposes of these computations, time is measured based on a counter that runs at the same rate as the timestamp counter (TSC).
  - CPL > 0.
    - If the "PAUSE exiting" VM-execution control is 0, the PAUSE instruction executes normally.
    - If the "PAUSE exiting" VM-execution control is 1, the PAUSE instruction causes a VM exit.

    The "PAUSE-loop exiting" VM-execution control is ignored if CPL > 0.
- **RDMSR.** The RDMSR instruction causes a VM exit if any of the following are true:
  - The "use MSR bitmaps" VM-execution control is 0.

1. "PAUSE-loop exiting" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "PAUSE-loop exiting" VM-execution control were 0. See Section 24.6.2.

- — The value of ECX is not in the range 00000000H – 00001FFFH or C0000000H – C0001FFFH.
- — The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in read bitmap for low MSRs is 1, where *n* is the value of ECX.
- — The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in read bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

See Section 24.6.9 for details regarding how these bitmaps are identified.

- **RDPMC.** The RDPMC instruction causes a VM exit if the "RDPMC exiting" VM-execution control is 1.
- **RDRAND.** The RDRAND instruction causes a VM exit if the "RDRAND exiting" VM-execution control is 1.[1]
- **RDTSC.** The RDTSC instruction causes a VM exit if the "RDTSC exiting" VM-execution control is 1.
- **RDTSCP.** The RDTSCP instruction causes a VM exit if the "RDTSC exiting" and "enable RDTSCP" VM-execution controls are both 1.[2]
- **RSM.** The RSM instruction causes a VM exit if executed in system-management mode (SMM).[3]
- **WBINVD.** The WBINVD instruction causes a VM exit if the "WBINVD exiting" VM-execution control is 1.[4]
- **WRMSR.** The WRMSR instruction causes a VM exit if any of the following are true:
  - — The "use MSR bitmaps" VM-execution control is 0.
  - — The value of ECX is not in the range 00000000H – 00001FFFH or C0000000H – C0001FFFH.
  - — The value of ECX is in the range 00000000H – 00001FFFH and bit *n* in write bitmap for low MSRs is 1, where *n* is the value of ECX.
  - — The value of ECX is in the range C0000000H – C0001FFFH and bit *n* in write bitmap for high MSRs is 1, where *n* is the value of ECX & 00001FFFH.

1. "RDRAND exiting" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "RDRAND exiting" VM-execution control were 0. See Section 24.6.2.
2. "Enable RDTSCP" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable RDTSCP" VM-execution control were 0. See Section 24.6.2.
3. Execution of the RSM instruction outside SMM causes an invalid-opcode exception regardless of whether the processor is in VMX operation. It also does so in VMX root operation in SMM; see Section 29.15.3.
4. "WBINVD exiting" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "WBINVD exiting" VM-execution control were 0. See Section 24.6.2.

See Section 24.6.9 for details regarding how these bitmaps are identified.

If an execution of WRMSR does not cause a VM exit as specified above and ECX = 808H (indicating the TPR MSR), instruction behavior is modified if the "virtualize x2APIC mode" VM-execution control is 1 (see Section 25.4) and it may cause a trap-like VM exit (see below).[1]

The MOV to CR8 and WRMSR instructions may cause "trap-like" VM exits. In such a case, the instruction completes before the VM exit occurs and that processor state is updated by the instruction (for example, the value of CS:RIP saved in the guest-state area of the VMCS references the next instruction).

Specifically, a trap-like VM exit occurs following either instruction if the execution reduces the value of the TPR shadow below that of the TPR threshold VM-execution control field (see Section 24.6.8 and Section 25.4) and the following hold:

- For MOV to CR8:
  - The "CR8-load exiting" VM-execution control is 0.
  - The "use TPR shadow" VM-execution control is 1.
- For WRMSR:
  - The "use MSR bitmaps" VM-execution control is 1, the value of ECX is 808H, and bit 808H in write bitmap for low MSRs is 0 (see above).
  - The "virtualize x2APIC mode" VM-execution control is 1.

25.2 APIC-ACCESS VM EXITS

If the "virtualize APIC accesses" VM-execution control is 1, an attempt to access memory using a physical address on the APIC-access page (see Section 24.6.8) causes a VM exit.[2,3] Such a VM exit is called an **APIC-access VM exit**.

Whether an operation that attempts to access memory with a physical address on the APIC-access page causes an APIC-access VM exit may be qualified based on the type of access. Section 25.2.1 describes the treatment of linear accesses, while Section 25.2.3 describes that of physical accesses. Section 25.2.4 discusses accesses to the TPR field on the APIC-access page (called VTPR accesses), which do not, if the "use TPR shadow" VM-execution control is 1, cause APIC-access VM exits.

1. "Virtualize x2APIC mode" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "virtualize x2APIC mode" VM-execution control were 0. See Section 24.6.2.
2. "Virtualize APIC accesses" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "virtualize APIC accesses" VM-execution control were 0. See Section 24.6.2.
3. Even when addresses are translated using EPT (see Section 28.2), the determination of whether an APIC-access VM exit occurs depends on an access's physical address, not its guest-physical address.

25.2.1 Linear Accesses to the APIC-Access Page

An access to the APIC-access page is called a **linear access** if (1) it results from a memory access using a linear address; and (2) the access's physical address is the translation of that linear address. Section 25.2.1.1 specifies which linear accesses to the APIC-access page cause APIC-access VM exits.

In general, the treatment of APIC-access VM exits caused by linear accesses is similar to that of page faults and EPT violations. Based upon this treatment, Section 25.2.1.2 specifies the priority of such VM exits with respect to other events, while Section 25.2.1.3 discusses instructions that may cause page faults without accessing memory and the treatment when they access the APIC-access page.

25.2.1.1 Linear Accesses That Cause APIC-Access VM Exits

Whether a linear access to the APIC-access page causes an APIC-access VM exit depends in part of the nature of the translation used by the linear address:

- If the linear access uses a translation with a 4-KByte page, it causes an APIC-access VM exit.
- If the linear access uses a translation with a large page (2-MByte, 4-MByte, or 1-GByte), the access may or may not cause an APIC-access VM exit. Section 25.5.1 describes the treatment of such accesses that do not cause an APIC-access VM exits.

  If CR0.PG = 1 and EPT is in use (the "enable EPT" VM-execution control is 1), a linear access uses a translation with a large page only if a large page is specified by both the guest paging structures and the EPT paging structures.[1]

It is recommended that software configure the paging structures so that any translation to the APIC-access page uses a 4-KByte page.

A linear access to the APIC-access page might not cause an APIC-access VM exit if the "enable EPT" VM-execution control is 1 and software has not properly invalidate information cached from the EPT paging structures:

- At time t_1, EPT was in use, the EPTP value was X, and some guest-physical address Y translated to an address that was not on the APIC-access page at that time. (This might be because the "virtualize APIC accesses" VM-execution control was 0.)
- At later time t_2, EPT is in use, the EPTP value is X, and a memory access uses a linear address that translates to Y, which now translates to an address on the

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

APIC-access page. (This implies that the "virtualize APIC accesses" VM-execution control is 1 at this time.)

- Software did not execute the INVEPT instruction between times t_1 and t_2, either with the all-context INVEPT type or with the single-context INVEPT type and X as the INVEPT descriptor.

In this case, the linear access at time t_2 might or might not cause an APIC-access VM exit. If it does not, the access operates on memory on the APIC-access page.

Software can avoid this situation through appropriate use of the INVEPT instruction; see Section 28.3.3.4.

A linear access to the APIC-access page might not cause an APIC-access VM exit if the "enable VPID" VM-execution control is 1 and software has not properly invalidated the TLBs and paging-structure caches:

- At time t_1, the processor was in VMX non-root operation with non-zero VPID X, and some linear address Y translated to an address that was not on the APIC-access page at that time. (This might be because the "virtualize APIC accesses" VM-execution control was 0.)
- At later time t_2, the processor was again in VMX non-root operation with VPID X, and a memory access uses linear address, which now translates to an address on the APIC-access page. (This implies that the "virtualize APIC accesses" VM-execution control is 1 at this time.)
- Software did not execute the INVVPID instruction in any of the following ways between times t_1 and t_2:
  - With the individual-address INVVPID type and an INVVPID descriptor specifying VPID X and linear address Y.
  - With the single-context INVVPID type and an INVVPID descriptor specifying VPID X.
  - With the all-context INVEPT type.
  - With the single-context-retaining-globals INVVPID type and an INVVPID descriptor specifying VPID X (assuming that, at time t1, the translation for Y was global; see Section 4.10, "Caching Translation Information" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details regarding global translations).

In this case, the linear access at time t_2 might or might not cause an APIC-access VM exit. If it does not, the access operates on memory on the APIC-access page.

Software can avoid this situation through appropriate use of the INVVPID instruction; see Section 28.3.3.3.

25.2.1.2 Priority of APIC-Access VM Exits Caused by Linear Accesses

The following items specify the priority relative to other events of APIC-access VM exits caused by linear accesses.

- The priority of an APIC-access VM exit on a linear access to memory is below that of any page fault or EPT violation that that access may incur. That is, a linear access does not cause an APIC-access VM exit if it would cause a page fault or an EPT violation.
- A linear access does not cause an APIC-access VM exit until after the accessed bits are set in the paging structures.
- A linear write access will not cause an APIC-access VM exit until after the dirty bit is set in the appropriate paging structure.
- With respect to all other events, any APIC-access VM exit due to a linear access has the same priority as any page fault or EPT violation that the linear access could cause. (This item applies to other events that the linear access may generate as well as events that may be generated by other accesses by the same instruction or operation.)

These principles imply among other things, that an APIC-access VM exit may occur during the execution of a repeated string instruction (including INS and OUTS). Suppose, for example, that the first *n* iterations (*n* may be 0) of such an instruction do not access the APIC-access page and that the next iteration does access that page. As a result, the first *n* iterations may complete and be followed by an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers reflect the completion of *n* iterations.

25.2.1.3 Instructions That May Cause Page Faults or EPT Violations Without Accessing Memory

APIC-access VM exits may occur as a result of executing an instruction that can cause a page fault or an EPT violation even if that instruction would not access the APIC-access page. The following are some examples:

- The CLFLUSH instruction is considered to read from the linear address in its source operand. If that address translates to one on the APIC-access page, the instruction causes an APIC-access VM exit.
- The ENTER instruction causes a page fault if the byte referenced by the final value of the stack pointer is not writable (even though ENTER does not write to that byte if its size operand is non-zero). If that byte is writable but is on the APIC-access page, ENTER causes an APIC-access VM exit.[1]
- An execution of the MASKMOVQ or MASKMOVDQU instructions with a zero mask may or may not cause a page fault or an EPT violation if the destination page is unwritable (the behavior is implementation-specific). An execution with a zero mask causes an APIC-access VM exit only on processors for which it could cause a page fault or an EPT violation.

1. The ENTER instruction may also cause page faults due to the memory accesses that it actually does perform. With regard to APIC-access VM exits, these are treated just as accesses by any other instruction.

- The MONITOR instruction is considered to read from the effective address in RAX. If the linear address corresponding to that address translates to one on the APIC-access page, the instruction causes an APIC-access VM exit.[1]
- An execution of the PREFETCH instruction that would result in an access to the APIC-access page does not cause an APIC-access VM exit.

25.2.2 Guest-Physical Accesses to the APIC-Access Page

An access to the APIC-access page is called a **guest-physical access** if (1) CR0.PG = 1;[2] (2) the "enable EPT" VM-execution control is 1;[3] (3) the access's physical address is the result of an EPT translation; and (4) either (a) the access was not generated by a linear address; or (b) the access's guest-physical address is not the translation of the access's linear address. Guest-physical accesses include the following when guest-physical addresses are being translated using EPT:

- Reads from the guest paging structures when translating a linear address (such an access uses a guest-physical address that is not the translation of that linear address).
- Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging.[4]
- Updates to the accessed and dirty bits in the guest paging structures when using a linear address (such an access uses a guest-physical address that is not the translation of that linear address).

Section 25.2.2.1 specifies when guest-physical accesses to the APIC-access page might not cause APIC-access VM exits. In general, the treatment of APIC-access VM exits caused by guest-physical accesses is similar to that of EPT violations. Based upon this treatment, Section 25.2.2.2 specifies the priority of such VM exits with respect to other events.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
3. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.
4. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

25.2.2.1 Guest-Physical Accesses That Might Not Cause APIC-Access VM Exits

Whether a guest-physical access to the APIC-access page causes an APIC-access VM exit depends on the nature of the EPT translation used by the guest-physical address and on how software is managing information cached from the EPT paging structures. The following items detail cases in which a guest-physical access to the APIC-access page might not cause an APIC-access VM exit:

- If the access uses a guest-physical address whose translation to the APIC-access page uses an EPT PDPTE that maps a 1-GByte page (because bit 7 of the EPT PDPTE is 1).
- If the access uses a guest-physical address whose translation to the APIC-access page uses an EPT PDE that maps a 2-MByte page (because bit 7 of the EPT PDE is 1).
- Software has not properly invalidated information cached from the EPT paging structures:
  - At time t_1, EPT was in use, the EPTP value was X, and some guest-physical address Y translated to an address that was not on the APIC-access page at that time. (This might be because the "virtualize APIC accesses" VM-execution control was 0.)
  - At later time t_2, the EPTP value is X and a memory access uses guest-physical address Y, which now translates to an address on the APIC-access page. (This implies that the "virtualize APIC accesses" VM-execution control is 1 at this time.)
  - Software did not execute the INVEPT instruction, either with the all-context INVEPT type or with the single-context INVEPT type and X as the INVEPT descriptor, between times t_1 and t_2.

In any of the above cases, the guest-physical access at time t_2 might or might not an APIC-access VM exit. If it does not, the access operates on memory on the APIC-access page.

Software can avoid this situation through appropriate use of the INVEPT instruction; see Section 28.3.3.4.

25.2.2.2 Priority of APIC-Access VM Exits Caused by Guest-Physical Accesses

The following items specify the priority relative to other events of APIC-access VM exits caused by guest-physical accesses.

- The priority of an APIC-access VM exit caused by a guest-physical access to memory is below that of any EPT violation that that access may incur. That is, a guest-physical access does not cause an APIC-access VM exit if it would cause an EPT violation.

- With respect to all other events, any APIC-access VM exit caused by a guest-physical access has the same priority as any EPT violation that the guest-physical access could cause.

25.2.3 Physical Accesses to the APIC-Access Page

An access to the APIC-access page is called a **physical access** if (1) either (a) the "enable EPT" VM-execution control is 0;[1] or (b) the access's physical address is not the result of a translation through the EPT paging structures; and (2) either (a) the access is not generated by a linear address; or (b) the access's physical address is not the translation of its linear address.

Physical accesses include the following:

- If the "enable EPT" VM-execution control is 0:
  - Reads from the paging structures when translating a linear address.
  - Loads of the page-directory-pointer-table entries by MOV to CR when the logical processor is using (or that causes the logical processor to use) PAE paging.[2]
  - Updates to the accessed and dirty bits in the paging structures.
- If the "enable EPT" VM-execution control is 1, accesses to the EPT paging structures.
- Any of the following accesses made by the processor to support VMX non-root operation:
  - Accesses to the VMCS region.
  - Accesses to data structures referenced (directly or indirectly) by physical addresses in VM-execution control fields in the VMCS. These include the I/O bitmaps, the MSR bitmaps, and the virtual-APIC page.
- Accesses that effect transitions into and out of SMM.[3] These include the following:
  - Accesses to SMRAM during SMI delivery and during execution of RSM.
  - Accesses during SMM VM exits (including accesses to MSEG) and during VM entries that return from SMM.

---

1. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.
2. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.
3. Technically, these accesses do not occur in VMX non-root operation. They are included here for clarity.

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. (A physical write to the APIC-access page may write to memory as specified in Section 25.5.2 before causing the APIC-access VM exit.) The priority of an APIC-access VM exit caused by physical access is not defined relative to other events that the access may cause. Section 25.5.2 describes the treatment of physical accesses to the APIC-access page that do not cause APIC-access VM exits.

It is recommended that software not set the APIC-access address to any of those used by physical memory accesses (identified above). For example, it should not set the APIC-access address to the physical address of any of the active paging structures if the "enable EPT" VM-execution control is 0.

25.2.4 VTPR Accesses

A memory access is a **VTPR access** if all of the following hold: (1) the "use TPR shadow" VM-execution control is 1; (2) the access is not for an instruction fetch; (3) the access is at most 32 bits in width; and (4) the access is to offset 80H on the APIC-access page.

A memory access is not a VTPR access (even if it accesses only bytes in the range 80H–83H on the APIC-access page) if any of the following hold: (1) the "use TPR shadow" VM-execution control is 0; (2) the access is for an instruction fetch; (3) the access is more than 32 bits in width; or (4) the access is to some offset is on the APIC-access page other than 80H. For example, a 16-bit access to offset 81H on the APIC-access page is **not** a VTPR access, even if the "use TPR shadow" VM-execution control is 1.

In general, VTPR accesses do not cause APIC-access VM exits. Instead, they are treated as described in Section 25.5.3. Physical VTPR accesses (see Section 25.2.3) may or may not cause APIC-access VM exits; see Section 25.5.2.

25.3 OTHER CAUSES OF VM EXITS

In addition to VM exits caused by instruction execution, the following events can cause VM exits:

- **Exceptions.** Exceptions (faults, traps, and aborts) cause VM exits based on the exception bitmap (see Section 24.6.3). If an exception occurs, its vector (in the range 0–31) is used to select a bit in the exception bitmap. If the bit is 1, a VM exit occurs; if the bit is 0, the exception is delivered normally through the guest IDT. This use of the exception bitmap applies also to exceptions generated by the instructions INT3, INTO, BOUND, and UD2.

  Page faults (exceptions with vector 14) are specially treated. When a page fault occurs, a logical processor consults (1) bit 14 of the exception bitmap; (2) the error code produced with the page fault [PFEC]; (3) the page-fault error-code mask field [PFEC_MASK]; and (4) the page-fault error-code match field [PFEC_MATCH]. It checks if PFEC & PFEC_MASK = PFEC_MATCH. If there is equality, the specification of bit 14 in the exception bitmap is followed (for example, a VM exit occurs if that bit is set). If there is inequality, the meaning of that bit is reversed (for example, a VM exit occurs if that bit is clear).

Thus, if software desires VM exits on all page faults, it can set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If software desires VM exits on no page faults, it can set bit 14 in the exception bitmap to 1, the page-fault error-code mask field to 00000000H, and the page-fault error-code match field to FFFFFFFFH.

- **Triple fault.** A VM exit occurs if the logical processor encounters an exception while attempting to call the double-fault handler and that exception itself does not cause a VM exit due to the exception bitmap. This applies to the case in which the double-fault exception was generated within VMX non-root operation, the case in which the double-fault exception was generated during event injection by VM entry, and to the case in which VM entry is injecting a double-fault exception.
- **External interrupts.** An external interrupt causes a VM exit if the "external-interrupt exiting" VM-execution control is 1. Otherwise, the interrupt is delivered normally through the IDT. (If a logical processor is in the shutdown state or the wait-for-SIPI state, external interrupts are blocked. The interrupt is not delivered through the IDT and no VM exit occurs.)
- **Non-maskable interrupts (NMIs).** An NMI causes a VM exit if the "NMI exiting" VM-execution control is 1. Otherwise, it is delivered using descriptor 2 of the IDT. (If a logical processor is in the wait-for-SIPI state, NMIs are blocked. The NMI is not delivered through the IDT and no VM exit occurs.)
- **INIT signals.** INIT signals cause VM exits. A logical processor performs none of the operations normally associated with these events. Such exits do not modify register state or clear pending events as they would outside of VMX operation. (If a logical processor is in the wait-for-SIPI state, INIT signals are blocked. They do not cause VM exits in this case.)
- **Start-up IPIs (SIPIs). SIPIs cause VM exits.** If a logical processor is not in the wait-for-SIPI activity state when a SIPI arrives, no VM exit occurs and the SIPI is discarded. VM exits due to SIPIs do not perform any of the normal operations associated with those events: they do not modify register state as they would outside of VMX operation. (If a logical processor is not in the wait-for-SIPI state, SIPIs are blocked. They do not cause VM exits in this case.)
- **Task switches.** Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. See Section 25.6.2.
- **System-management interrupts (SMIs).** If the logical processor is using the dual-monitor treatment of SMIs and system-management mode (SMM), SMIs cause SMM VM exits. See Section 29.15.2.[1]

1. Under the dual-monitor treatment of SMIs and SMM, SMIs also cause SMM VM exits if they occur in VMX root operation outside SMM. If the processor is using the default treatment of SMIs and SMM, SMIs are delivered as described in Section 29.14.1.

- **VMX-preemption timer.** A VM exit occurs when the timer counts down to zero. See Section 25.7.1 for details of operation of the VMX-preemption timer. As noted in that section, the timer does not cause VM exits if the logical processor is outside the C-states C0, C1, and C2.

  Debug-trap exceptions and higher priority events take priority over VM exits caused by the VMX-preemption timer. VM exits caused by the VMX-preemption timer take priority over VM exits caused by the "NMI-window exiting" VM-execution control and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would a non-maskable interrupt. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

In addition, there are controls that cause VM exits based on the readiness of guest software to receive interrupts:

- If the "interrupt-window exiting" VM-execution control is 1, a VM exit occurs before execution of any instruction if RFLAGS.IF = 1 and there is no blocking of events by STI or by MOV SS (see Table 24-3). Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.6.5).

  Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would an external interrupt. Specifically, they wake a logical processor from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.
- If the "NMI-window exiting" VM-execution control is 1, a VM exit occurs before execution of any instruction if there is no virtual-NMI blocking and there is no blocking of events by MOV SS (see Table 24-3). (A logical processor may also prevent such a VM exit if there is blocking of events by STI.) Such a VM exit occurs immediately after VM entry if the above conditions are true (see Section 26.6.6).

  VM exits caused by the VMX-preemption timer and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.

  These VM exits wake a logical processor from the same inactive states as would an NMI. Specifically, they wake a logical processor from the shutdown state and from the states entered using the HLT and MWAIT instructions. These VM exits do not occur if the logical processor is in the wait-for-SIPI state.

25.4 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation. Some of these changes are determined by the settings of certain VM-execution control fields. The following items detail such changes:

- **CLTS.** Behavior of the CLTS instruction is determined by the bits in position 3 (corresponding to CR0.TS) in the CR0 guest/host mask and the CR0 read shadow:
  - — If bit 3 in the CR0 guest/host mask is 0, CLTS clears CR0.TS normally (the value of bit 3 in the CR0 read shadow is irrelevant in this case), unless CR0.TS is fixed to 1 in VMX operation (see Section 23.8), in which case CLTS causes a general-protection exception.
  - — If bit 3 in the CR0 guest/host mask is 1 and bit 3 in the CR0 read shadow is 0, CLTS completes but does not change the contents of CR0.TS.
  - — If the bits in position 3 in the CR0 guest/host mask and the CR0 read shadow are both 1, CLTS causes a VM exit.
- **INVPCID.** Behavior of the INVPCID instruction is determined first by the setting of the "enable INVPCID" VM-execution control:[1]
  - — If the "enable INVPCID" VM-execution control is 0, INVPCID causes an invalid-opcode exception (#UD).
  - — If the "enable INVPCID" VM-execution control is 1, treatment is based on the setting of the "INVLPG exiting" VM-execution control:
    - If the "INVLPG exiting" VM-execution control is 0, INVPCID operates normally.
    - If the "INVLPG exiting" VM-execution control is 1, INVPCID causes a VM exit.
- **IRET.** Behavior of IRET with regard to NMI blocking (see Table 24-3) is determined by the settings of the "NMI exiting" and "virtual NMIs" VM-execution controls:
  - — If the "NMI exiting" VM-execution control is 0, IRET operates normally and unblocks NMIs. (If the "NMI exiting" VM-execution control is 0, the "virtual NMIs" control must be 0; see Section 26.2.1.1.)
  - — If the "NMI exiting" VM-execution control is 1, IRET does not affect blocking of NMIs. If, in addition, the "virtual NMIs" VM-execution control is 1, the logical processor tracks virtual-NMI blocking. In this case, IRET removes any virtual-NMI blocking.

1. "Enable INVPCID" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable INVPCID" VM-execution control were 0. See Section 24.6.2.

The unblocking of NMIs or virtual NMIs specified above occurs even if IRET causes a fault.

- **LMSW.** Outside of VMX non-root operation, LMSW loads its source operand into CR0[3:0], but it does not clear CR0.PE if that bit is set. In VMX non-root operation, an execution of LMSW that does not cause a VM exit (see Section 25.1.3) leaves unmodified any bit in CR0[3:0] corresponding to a bit set in the CR0 guest/host mask. An attempt to set any other bit in CR0[3:0] to a value not supported in VMX operation (see Section 23.8) causes a general-protection exception. Attempts to clear CR0.PE are ignored without fault.
- **MOV from CR0.** The behavior of MOV from CR0 is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

  Depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.
- **MOV from CR3.** If the "enable EPT" VM-execution control is 1 and an execution of MOV from CR3 does not cause a VM exit (see Section 25.1.3), the value loaded from CR3 is a guest-physical address; see Section 28.2.1.
- **MOV from CR4.** The behavior of MOV from CR4 is determined by the CR4 guest/host mask and the CR4 read shadow. For each position corresponding to a bit clear in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR4. For each position corresponding to a bit set in the CR4 guest/host mask, the destination operand is loaded with the value of the corresponding bit in the CR4 read shadow. Thus, if every bit is cleared in the CR4 guest/host mask, MOV from CR4 reads normally from CR4; if every bit is set in the CR4 guest/host mask, MOV from CR4 returns the value of the CR4 read shadow.

  Depending on the contents of the CR4 guest/host mask and the CR4 read shadow, bits may be set in the destination that would never be set when reading directly from CR4.
- **MOV from CR8.** Behavior of the MOV from CR8 instruction (which can be executed only in 64-bit mode) is determined by the settings of the "CR8-store exiting" and "use TPR shadow" VM-execution controls:
  - If both controls are 0, MOV from CR8 operates normally.
  - If the "CR8-store exiting" VM-execution control is 0 and the "use TPR shadow" VM-execution control is 1, MOV from CR8 reads from the TPR shadow. Specifically, it loads bits 3:0 of its destination operand with the value

of bits 7:4 of byte 80H of the virtual-APIC page (see Section 24.6.8). Bits 63:4 of the destination operand are cleared.

— If the "CR8-store exiting" VM-execution control is 1, MOV from CR8 causes a VM exit; the "use TPR shadow" VM-execution control is ignored in this case.

- **MOV to CR0.** An execution of MOV to CR0 that does not cause a VM exit (see Section 25.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. Treatment of attempts to modify other bits in CR0 depends on the setting of the "unrestricted guest" VM-execution control:[1]
  - — If the control is 0, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 to a value not supported in VMX operation (see Section 23.8).
  - — If the control is 1, MOV to CR0 causes a general-protection exception if it attempts to set any bit in CR0 other than bit 0 (PE) or bit 31 (PG) to a value not supported in VMX operation. It remains the case, however, that MOV to CR0 causes a general-protection exception if it would result in CR0.PE = 0 and CR0.PG = 1 or if it would result in CR0.PG = 1, CR4.PAE = 0, and IA32_EFER.LME = 1.
- **MOV to CR3.** If the "enable EPT" VM-execution control is 1 and an execution of MOV to CR3 does not cause a VM exit (see Section 25.1.3), the value loaded into CR3 is treated as a guest-physical address; see Section 28.2.1.
  - — If PAE paging is not being used, the instruction does not use the guest-physical address to access memory and it does not cause it to be translated through EPT.[2]
  - — If PAE paging is being used, the instruction translates the guest-physical address through EPT and uses the result to load the four (4) page-directory-pointer-table entries (PDPTEs). The instruction does not use the guest-physical addresses the PDPTEs to access memory and it does not cause them to be translated through EPT.
- **MOV to CR4.** An execution of MOV to CR4 that does not cause a VM exit (see Section 25.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. Such an execution causes a general-protection exception if it attempts to set any bit in CR4 (not corresponding to a bit set in the CR4 guest/host mask) to a value not supported in VMX operation (see Section 23.8).
- **MOV to CR8.** Behavior of the MOV to CR8 instruction (which can be executed only in 64-bit mode) is determined by the settings of the "CR8-load exiting" and "use TPR shadow" VM-execution controls:
  - — If both controls are 0, MOV to CR8 operates normally.

1. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "unrestricted guest" VM-execution control were 0. See Section 24.6.2.

2. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.

  - If the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution control is 1, MOV to CR8 writes to the TPR shadow. Specifically, it stores bits 3:0 of its source operand into bits 7:4 of byte 80H of the virtual-APIC page (see Section 24.6.8); bits 3:0 of that byte and bytes 129-131 of that page are cleared. Such a store may cause a VM exit to occur after it completes (see Section 25.1.3).
  - If the “CR8-load exiting” VM-execution control is 1, MOV to CR8 causes a VM exit; the “use TPR shadow” VM-execution control is ignored in this case.
- **MWAIT.** Behavior of the MWAIT instruction (which always causes an invalid-opcode exception—#UD—if CPL > 0) is determined by the setting of the “MWAIT exiting” VM-execution control:
  - If the “MWAIT exiting” VM-execution control is 1, MWAIT causes a VM exit.
  - If the “MWAIT exiting” VM-execution control is 0, MWAIT operates normally if any of the following is true: (1) the “interrupt-window exiting” VM-execution control is 0; (2) ECX[0] is 0; or (3) RFLAGS.IF = 1.
  - If the “MWAIT exiting” VM-execution control is 0, the “interrupt-window exiting” VM-execution control is 1, ECX[0] = 1, and RFLAGS.IF = 0, MWAIT does not cause the processor to enter an implementation-dependent optimized state; instead, control passes to the instruction following the MWAIT instruction.
- **RDMSR.** Section 25.1.3 identifies when executions of the RDMSR instruction cause VM exits. If such an execution causes neither a fault due to CPL > 0 nor a VM exit, the instruction’s behavior may be modified for certain values of ECX:
  - If ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR), the value returned by the instruction is determined by the setting of the “use TSC offsetting” VM-execution control as well as the TSC offset:
    - If the control is 0, the instruction operates normally, loading EAX:EDX with the value of the IA32_TIME_STAMP_COUNTER MSR.
    - If the control is 1, the instruction loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value).

    The 1-setting of the “use TSC-offsetting” VM-execution control does not effect executions of RDMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions return the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.
  - If ECX contains 808H (indicating the TPR MSR), instruction behavior is determined by the setting of the “virtualize x2APIC mode” VM-execution control:[1]

1. “Virtualize x2APIC mode” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “virtualize x2APIC mode” VM-execution control were 0. See Section 24.6.2.

    - If the control is 0, the instruction operates normally. If the local APIC is in x2APIC mode, EAX[7:0] is loaded with the value of the APIC's task-priority register (EDX and EAX[31:8] are cleared to 0). If the local APIC is not in x2APIC mode, a general-protection fault occurs.
    - If the control is 1, the instruction loads EAX:EDX with the value of bytes 87H:80H of the virtual-APIC page. This occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode).
- **RDTSC.** Behavior of the RDTSC instruction is determined by the settings of the "RDTSC exiting" and "use TSC offsetting" VM-execution controls as well as the TSC offset:
  - — If both controls are 0, RDTSC operates normally.
  - — If the "RDTSC exiting" VM-execution control is 0 and the "use TSC offsetting" VM-execution control is 1, RDTSC loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value).
  - — If the "RDTSC exiting" VM-execution control is 1, RDTSC causes a VM exit.
- **RDTSCP.** Behavior of the RDTSCP instruction is determined first by the setting of the "enable RDTSCP" VM-execution control:[1]
  - — If the "enable RDTSCP" VM-execution control is 0, RDTSCP causes an invalid-opcode exception (#UD).
  - — If the "enable RDTSCP" VM-execution control is 1, treatment is based on the settings of the "RDTSC exiting" and "use TSC offsetting" VM-execution controls as well as the TSC offset:
    - If both controls are 0, RDTSCP operates normally.
    - If the "RDTSC exiting" VM-execution control is 0 and the "use TSC offsetting" VM-execution control is 1, RDTSCP loads EAX:EDX with the sum (using signed addition) of the value of the IA32_TIME_STAMP_COUNTER MSR and the value of the TSC offset (interpreted as a signed value); it also loads ECX with the value of bits 31:0 of the IA32_TSC_AUX MSR.
    - If the "RDTSC exiting" VM-execution control is 1, RDTSCP causes a VM exit.
- **SMSW.** The behavior of SMSW is determined by the CR0 guest/host mask and the CR0 read shadow. For each position corresponding to a bit clear in the CR0 guest/host mask, the destination operand is loaded with the value of the corresponding bit in CR0. For each position corresponding to a bit set in the CR0 guest/host mask, the destination operand is loaded with the value of the corre-

1. "Enable RDTSCP" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "enable RDTSCP" VM-execution control were 0. See Section 24.6.2.

sponding bit in the CR0 read shadow. Thus, if every bit is cleared in the CR0 guest/host mask, MOV from CR0 reads normally from CR0; if every bit is set in the CR0 guest/host mask, MOV from CR0 returns the value of the CR0 read shadow.

Note the following: (1) for any memory destination or for a 16-bit register destination, only the low 16 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:16 of a register destination are left unchanged); (2) for a 32-bit register destination, only the low 32 bits of the CR0 guest/host mask and the CR0 read shadow are used (bits 63:32 of the destination are cleared); and (3) depending on the contents of the CR0 guest/host mask and the CR0 read shadow, bits may be set in the destination that would never be set when reading directly from CR0.

- **WRMSR.** Section 25.1.3 identifies when executions of the WRMSR instruction cause VM exits. If such an execution neither a fault due to CPL > 0 nor a VM exit, the instruction's behavior may be modified for certain values of ECX:
  - If ECX contains 79H (indicating IA32_BIOS_UPDT_TRIG MSR), no microcode update is loaded, and control passes to the next instruction. This implies that microcode updates cannot be loaded in VMX non-root operation.
  - If ECX contains 808H (indicating the TPR MSR) and either EDX or EAX[31:8] is non-zero, a general-protection fault occurs (this is true even if the logical processor is not in VMX non-root operation). Otherwise, instruction behavior is determined by the setting of the "virtualize x2APIC mode" VM-execution control and the value of the TPR-threshold VM-execution control field:
    - If the control is 0, the instruction operates normally. If the local APIC is in x2APIC mode, the value of EAX[7:0] is written to the APIC's task-priority register. If the local APIC is not in x2APIC mode, a general-protection fault occurs.
    - If the control is 1, the instruction stores the value of EAX:EDX to bytes 87H:80H of the virtual-APIC page. This store occurs even if the local APIC is not in x2APIC mode (no general-protection fault occurs because the local APIC is not x2APIC mode). The store may cause a VM exit to occur after the instruction completes (see Section 25.1.3).
    - The 1-setting of the "use TSC-offsetting" VM-execution control does not effect executions of WRMSR if ECX contains 10H (indicating the IA32_TIME_STAMP_COUNTER MSR). Such executions modify the actual timestamp counter without regard to the TSC offset.
    - The 1-setting of the "use TSC-offsetting" VM-execution control does not effect executions of WRMSR if ECX contains 6E0H (indicating the IA32_TSC_DEADLINE MSR). Such executions modify the APIC-timer deadline relative to the actual timestamp counter without regard to the TSC offset.

25.5 APIC ACCESSES THAT DO NOT CAUSE VM EXITS

As noted in Section 25.2, if the “virtualize APIC accesses” VM-execution control is 1, most memory accesses to the APIC-access page (see Section 24.6.2) cause APIC-access VM exits.[1] Section 25.2 identifies potential exceptions. These are covered in Section 25.5.1 through Section 25.5.3.

In some cases, an attempt to access memory on the APIC-access page is converted to an access to the virtual-APIC page (see Section 24.6.8). In these cases, the access uses the memory type reported in bit 53:50 of the IA32_VMX_BASIC MSR (see Appendix A.1).

25.5.1 Linear Accesses to the APIC-Access Page Using Large-Page Translations

As noted in Section 25.2.1, a linear access to the APIC-access page using translation with a large page (2-MByte, 4-MByte, or 1-GByte) may or may not cause an APIC-access VM exit. If it does not and the access is not a VTPR access (see Section 25.2.4), the access operates on memory on the APIC-access page. Section 25.5.3 describes the treatment if there is no APIC-access VM exit and the access is a VTPR access.

25.5.2 Physical Accesses to the APIC-Access Page

A physical access to the APIC-access page may or may not cause an APIC-access VM exit. If it does not and the access is not a VTPR access (see Section 25.2.4), the access operates on memory on the APIC-access page (this may happen if the access causes an APIC-access VM exit). Section 25.5.3 describes the treatment if there is no APIC-access VM exit and the access is a VTPR access.

25.5.3 VTPR Accesses

As noted in Section 25.2.4, a memory access is a VTPR access if all of the following hold: (1) the “use TPR shadow” VM-execution control is 1; (2) the access is not for an instruction fetch; (3) the access is at most 32 bits in width; and (4) the access is to offset 80H on the APIC-access page.

The treatment of VTPR accesses depends on the nature of the access:

- A linear VTPR access using a translation with a 4-KByte page does not cause an APIC-access VM exit. Instead, it is converted so that, instead of accessing offset

1. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 24.6.2.

80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 25.5.3.1 to Section 25.5.3.3.

- A linear VTPR access using a translation with a large page (2-MByte, 4-MByte, or 1-GByte) may be treated in either of two ways:
  - It may operate on memory on the APIC-access page. The details in Section 25.5.3.1 to Section 25.5.3.3 do not apply.
  - It may be converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 25.5.3.1 to Section 25.5.3.3.
- A physical VTPR access may be treated in one of three ways:
  - It may cause an APIC-access VM exit. The details in Section 25.5.3.1 to Section 25.5.3.3 do not apply.
  - It may operate on memory on the APIC-access page (and possibly then cause an APIC-access VM exit). The details in Section 25.5.3.1 to Section 25.5.3.3 do not apply.
  - It may be converted so that, instead of accessing offset 80H on the APIC-access page, it accesses offset 80H on the virtual-APIC page. Further details are provided in Section 25.5.3.1 to Section 25.5.3.3.

Linear VTPR accesses never cause APIC-access VM exits (recall that an access is a VTPR access only if the “use TPR shadow” VM-execution control is 1).

25.5.3.1 Treatment of Individual VTPR Accesses

The following items detail the treatment of VTPR accesses:

- VTPR read accesses. Such an access completes normally (reading data from the field at offset 80H on the virtual-APIC page).

  The following items detail certain instructions that are considered to perform read accesses and how they behavior when accessing the VTPR:
  - A VTPR access using the CLFLUSH instruction flushes data for offset 80H on the virtual-APIC page.
  - A VTPR access using the LMSW instruction may cause a VM exit due to the CR0 guest/host mask and the CR0 read shadow.
  - A VTPR access using the MONITOR instruction causes the logical processor to monitor offset 80H on the virtual-APIC page.
  - A VTPR access using the PREFETCH instruction may prefetch data; if so, it is from offset 80H on the virtual-APIC page.
- VTPR write accesses. Such an access completes normally (writing data to the field at offset 80H on the virtual-APIC page) and causes a TPR-shadow update (see Section 25.5.3.3).

The following items detail certain instructions that are considered to perform write accesses and how they behavior when accessing the VTPR:

— The ENTER instruction is considered to write to VTPR if the byte referenced by the final value of the stack pointer is at offset 80H on the APIC-access page (even though ENTER does not write to that byte if its size operand is non-zero). The instruction is followed by a TPR-shadow update.

— A VTPR access using the SMSW instruction stores data determined by the current CR0 contents, the CR0 guest/host mask, and the CR0 read shadow. The instruction is followed by a TPR-shadow update.

25.5.3.2 Operations with Multiple Accesses

Some operations may access multiple addresses. These operations include the execution of some instructions and the delivery of events through the IDT (including those injected with VM entry). In some cases, the Intel® 64 architecture specifies the ordering of these memory accesses. The following items describe the treatment of VTPR accesses that are part of such multi-access operations:

- Read-modify-write instructions may first perform a VTPR read access and then a VTPR write access. Both accesses complete normally (as described in Section 25.5.3.1). The instruction is followed by a TPR-shadow update (see Section 25.5.3.3).
- Some operations may perform a VTPR write access and subsequently cause a fault. This situation is treated as follows:

  — If the fault leads to a VM exit, no TPR-shadow update occurs.

  — If the fault does not lead to a VM exit, a TPR-shadow update occurs after fault delivery completes and before execution of the fault handler.
- If an operation includes a VTPR access and an access to some other field on the APIC-access page, the latter access causes an APIC-access VM exit as described in Section 25.2.

  If the operation performs a VTPR write access before the APIC-access VM exit, there is no TPR-shadow update.
- Suppose that the first iteration of a repeated string instruction (including OUTS) that accesses the APIC-access page performs a VTPR read access and that the next iteration would read from the APIC-access page using an offset other than 80H. The following items describe the behavior of the logical processor:

  — The iteration that performs the VTPR read access completes successfully, reading data from offset 80H on the virtual-APIC page.

  — The iteration that would read from the other offset causes an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers are such that iteration would be repeated if the instruction were restarted.

- Suppose that the first iteration of a repeated string instruction (including INS) that accesses the APIC-access page performs a VTPR write access and that the next iteration would write to the APIC-access page using an offset other than 80H. The following items describe the behavior of the logical processor:
  - The iteration that performs the VTPR write access writes data to offset 80H on the virtual-APIC page. The write is followed by a TPR-shadow update, which may cause a VM exit (see Section 25.5.3.3).
  - If the TPR-shadow update does cause a VM exit, the instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers are such that the next iteration would be performed if the instruction were restarted.
  - If the TPR-shadow update does not cause a VM exit, the iteration that would write to the other offset causes an APIC-access VM exit. The instruction pointer saved in the VMCS references the repeated string instruction and the values of the general-purpose registers are such that that iteration would be repeated if the instruction were restarted.
- Suppose that the last iteration of a repeated string instruction (including INS) performs a VTPR write access. The iteration writes data to offset 80H on the virtual-APIC page. The write is followed by a TPR-shadow update, which may cause a VM exit (see Section 25.5.3.3). If it does, the instruction pointer saved in the VMCS references the instruction after the string instruction and the values of the general-purpose registers reflect completion of the string instruction.

25.5.3.3 TPR-Shadow Updates

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1, a logical processor performs certain actions after any operation (or iteration of a repeated string instruction) with a VTPR write access. These actions are called a **TPR-shadow update**. (As noted in Section 25.5.3.2, a TPR-shadow update does not occur following an access that causes a VM exit.)

A TPR-shadow update includes the following actions:

1. Bits 31:8 at offset 80H on the virtual-APIC page are cleared.
2. If the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 at offset 80H on the virtual-APIC page, a VM exit will occur.

TPR-shadow updates take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. A TPR-shadow update thus has priority over any debug exceptions that may have been triggered by the operation causing the TPR-shadow update. TPR-shadow updates (and any VM exits they cause) are not blocked if RFLAGS.IF = 0 or by the MOV SS, POP SS, or STI instructions.

25.6 OTHER CHANGES IN VMX NON-ROOT OPERATION

Treatments of event blocking and of task switches differ in VMX non-root operation as described in the following sections.

25.6.1 Event Blocking

Event blocking is modified in VMX non-root operation as follows:

- If the "external-interrupt exiting" VM-execution control is 1, RFLAGS.IF does not control the blocking of external interrupts. In this case, an external interrupt that is not blocked for other reasons causes a VM exit (even if RFLAGS.IF = 0).
- If the "external-interrupt exiting" VM-execution control is 1, external interrupts may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).
- If the "NMI exiting" VM-execution control is 1, non-maskable interrupts (NMIs) may or may not be blocked by STI or by MOV SS (behavior is implementation-specific).

25.6.2 Treatment of Task Switches

Task switches are not allowed in VMX non-root operation. Any attempt to effect a task switch in VMX non-root operation causes a VM exit. However, the following checks are performed (in the order indicated), possibly resulting in a fault, before there is any possibility of a VM exit due to task switch:

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
    a. If CALL, INT *n*, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
    b. If CALL, INT *n*, INT3, INTO, or JMP accesses a task gate outside IA-32e mode, privilege-levels checks are performed on the task gate but, if they pass, privilege levels are not checked on the referenced task-state segment (TSS) descriptor.
    c. If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.
    d. If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
    e. If a non-maskable interrupt (NMI), an exception, or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.

   f. If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP) and overflow exceptions (#OF), or an external interrupt accesses a task gate in the IDT outside IA-32e mode, no privilege checks are performed.

   g. If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.

   h. If IRET is executed with RFLAGS.NT = 1 outside IA-32e mode, a TSS descriptor is accessed directly and no privilege checks are made.

2. Checks are made on the new TSS selector (for example, that is within GDT limits).
3. The new TSS descriptor is read. (A page fault results if a relevant GDT page is not present).
4. The TSS descriptor is checked for proper values of type (depends on type of task switch), P bit, S bit, and limit.

Only if checks 1–4 all pass (do not generate faults) might a VM exit occur. However, the ordering between a VM exit due to a task switch and a page fault resulting from accessing the old TSS or the new TSS is implementation-specific. Some logical processors may generate a page fault (instead of a VM exit due to a task switch) if accessing either TSS would cause a page fault. Other logical processors may generate a VM exit due to a task switch even if accessing either TSS would cause a page fault.

If an attempt at a task switch through a task gate in the IDT causes an exception (before generating a VM exit due to the task switch) and that exception causes a VM exit, information about the event whose delivery that accessed the task gate is recorded in the IDT-vectoring information fields and information about the exception that caused the VM exit is recorded in the VM-exit interruption-information fields. See Section 27.2. The fact that a task gate was being accessed is not recorded in the VMCS.

If an attempt at a task switch through a task gate in the IDT causes VM exit due to the task switch, information about the event whose delivery accessed the task gate is recorded in the IDT-vectoring fields of the VMCS. Since the cause of such a VM exit is a task switch and not an interruption, the valid bit for the VM-exit interruption information field is 0. See Section 27.2.

25.7 FEATURES SPECIFIC TO VMX NON-ROOT OPERATION

Some VM-execution controls support features that are specific to VMX non-root operation. These are the VMX-preemption timer (Section 25.7.1) and the monitor trap flag (Section 25.7.2), translation of guest-physical addresses (Section 25.7.3), and VM functions (Section 25.7.4).

25.7.1 VMX-Preemption Timer

If the last VM entry was performed with the 1-setting of “activate VMX-preemption timer” VM-execution control, the **VMX-preemption timer** counts down (from the value loaded by VM entry; see Section 26.6.4) in VMX non-root operation. When the timer counts down to zero, it stops counting down and a VM exit occurs (see Section 25.3).

The VMX-preemption timer counts down at rate proportional to that of the timestamp counter (TSC). Specifically, the timer counts down by 1 every time bit X in the TSC changes due to a TSC increment. The value of X is in the range 0–31 and can be determined by consulting the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).

The VMX-preemption timer operates in the C-states C0, C1, and C2; it also operates in the shutdown and wait-for-SIPI states. If the timer counts down to zero in C1, C2, or shutdown, the logical processor transitions to the C0 C-state and causes a VM exit. (The timer does not cause a VM exit if it counts down to zero in the wait-for-SIPI state.) The timer is not decremented and does not cause VM exits in C-states deeper than C2.

Treatment of the timer in the case of system management interrupts (SMIs) and system-management mode (SMM) depends on whether the treatment of SMIs and SMM:

- If the default treatment of SMIs and SMM (see Section 29.14) is active, the VMX-preemption timer counts across an SMI to VMX non-root operation, subsequent execution in SMM, and the return from SMM via the RSM instruction. However, the timer can cause a VM exit only from VMX non-root operation. If the timer expires during SMI, in SMM, or during RSM, a timer-induced VM exit occurs immediately after RSM with its normal priority unless it is blocked based on activity state (Section 25.3).
- If the dual-monitor treatment of SMIs and SMM (see Section 29.15) is active, transitions into and out of SMM are VM exits and VM entries, respectively. The treatment of the VMX-preemption timer by those transitions is mostly the same as for ordinary VM exits and VM entries; Section 29.15.2 and Section 29.15.4 detail some differences.

25.7.2 Monitor Trap Flag

The **monitor trap flag** is a debugging feature that causes VM exits to occur on certain instruction boundaries in VMX non-root operation. Such VM exits are called **MTF VM exits**. An MTF VM exit may occur on an instruction boundary in VMX non-root operation as follows:

- If the “monitor trap flag” VM-execution control is 1 and VM entry is injecting a vectored event (see Section 26.5.1), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry.

- If VM entry is injecting a pending MTF VM exit (see Section 26.5.2), an MTF VM exit is pending on the instruction boundary before the first instruction following the VM entry. This is the case even if the “monitor trap flag” VM-execution control is 0.
- If the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and a pending event (e.g., debug exception or interrupt) is delivered before an instruction can execute, an MTF VM exit is pending on the instruction boundary following delivery of the event (or any nested exception).
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is a REP-prefixed string instruction:
  - If the first iteration of the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).
  - If the first iteration of the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary after that iteration.
- Suppose that the “monitor trap flag” VM-execution control is 1, VM entry is not injecting an event, and the first instruction following VM entry is **not** a REP-prefixed string instruction:
  - If the instruction causes a fault, an MTF VM exit is pending on the instruction boundary following delivery of the fault (or any nested exception).[1]
  - If the instruction does not cause a fault, an MTF VM exit is pending on the instruction boundary following execution of that instruction. If the instruction is INT3 or INTO, this boundary follows delivery of any software exception. If the instruction is INT *n*, this boundary follows delivery of a software interrupt. If the instruction is HLT, the MTF VM exit will be from the HLT activity state.

No MTF VM exit occurs if another VM exit occurs before reaching the instruction boundary on which an MTF VM exit would be pending (e.g., due to an exception or triple fault).

An MTF VM exit occurs on the instruction boundary on which it is pending unless a higher priority event takes precedence or the MTF VM exit is blocked due to the activity state:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over MTF VM exits. MTF VM exits take priority over debug-trap exceptions and lower priority events.
- No MTF VM exit occurs if the processor is in either the shutdown activity state or wait-for-SIPI activity state. If a non-maskable interrupt subsequently takes the

1. This item includes the cases of an invalid opcode exception—#UD— generated by the UD2 instruction and a BOUND-range exceeded exception—#BR—generated by the BOUND instruction.

logical processor out of the shutdown activity state without causing a VM exit, an MTF VM exit is pending after delivery of that interrupt.

25.7.3 Translation of Guest-Physical Addresses Using EPT

The extended page-table mechanism (EPT) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain physical addresses are treated as guest-physical addresses and are not used to access memory directly. Instead, guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Details of the EPT are given in Section 28.2.

25.7.4 VM Functions

A **VM function** is an operation provided by the processor that can be invoked from VMX non-root operation without a VM exit. VM functions are enabled and configured by the settings of different fields in the VMCS. Software in VMX non-root operation invokes a VM function with the **VMFUNC** instruction; the value of EAX selects the specific VM function being invoked.

Section 25.7.4.1 explains how VM functions are enabled. Section 25.7.4.2 specifies the behavior of the VMFUNC instruction. Section 25.7.4.3 describes a specific VM function called **EPTP switching**.

25.7.4.1 Enabling VM Functions

Software enables VM functions generally by setting the “enable VM functions” VM-execution control. A specific VM function is enabled by setting the corresponding VM-function control.

Suppose, for example, that software wants to enable EPTP switching (VM function 0; see Section 24.6.14).To do so, it must set the “activate secondary controls” VM-execution control (bit 31 of the primary processor-based VM-execution controls), the “enable VM functions” VM-execution control (bit 13 of the secondary processor-based VM-execution controls) and the “EPTP switching” VM-function control (bit 0 of the VM-function controls).

25.7.4.2 General Operation of the VMFUNC Instruction

The VMFUNC instruction causes an invalid-opcode exception (#UD) unless all of the following are true: (1) the “enable VM functions” VM-execution controls is 1;[1] (2) the

1. “Enable VM functions” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the “enable VM functions” VM-execution control were 0. See Section 24.6.2.

value of EAX is less than 64; and (3) the bit at position EAX is 1 in the VM-function controls. If these are all true, the VMFUNC instruction performs the functionality of the VM function specified by the value in EAX.

Individual VM functions may perform additional fault checking (e.g., one might cause a general-protection exception if CPL > 0). The general operation of VMFUNC includes no checks that might result in a VM exit, but specific VM functions might do so. If such a VM exit occurs, the basic exit reason used is 59 (3BH), indicating "VMFUNC", and the length of the VMFUNC instruction is saved into the VM-exit instruction-length field. The specification of a VM function may indicate that other exit information is provided.

The specific behavior of the EPTP-switching function (including checks that result in VM exits) is given in Section 25.7.4.3.

25.7.4.3 EPTP Switching

EPTP switching is VM function 0. This VM function allows software in VMX non-root operation to load a new value for the EPT pointer (EPTP), thereby establishing a different EPT paging-structure hierarchy (see Section 28.2 for details of the operation of EPT). Software is limited to selecting from a list of potential EPTP values configured in advance by software in VMX root operation.

Specifically, the value of ECX is used to select an entry from the EPTP list, the 4-KByte structure referenced by the EPTP-list address (see Section 24.6.14; because this structure contains 512 8-Byte entries, VMFUNC causes a VM exit if ECX ≥ 512). If the selected entry is a valid EPTP value (it would not cause VM entry to fail; see Section 26.2.1.1), it is stored in the EPTP field of the current VMCS and is used for subsequent accesses using guest-physical addresses. The following pseudocode provides details:

```
IF ECX ≥ 512
   THEN VM exit;
   ELSE
       tent_EPTP ← 8 bytes from EPTP-list address + 8 * ECX;
       IF tent_EPTP is not a valid EPTP value (would cause VM entry to fail if in EPTP)
           THEN VMexit;
           ELSE
               write tent_EPTP to the EPTP field in the current VMCS;
               start using tent_EPTP as the new EPTP value for address translation;
       FI;
FI;
```

Execution of the EPTP-switching VM function does not modify the state of any registers; no flags are modified.

If an execution of the EPTP-switching VM function causes a VM exit (as specified above), the basic exit reason used is 59, indication "VMFUNC". The length of the

VMFUNC instruction is saved into the VM-exit instruction-length field. No additional VM-exit information is provided.

An execution of VMFUNC loads EPTP from the EPTP list (and thus does not cause a fault or VM exit) is called an **EPTP-switching VMFUNC**. After an EPTP-switching VMFUNC, control passes to the next instruction. The logical processor starts creating and using guest-physical and combined mappings associated with the new value of bits 51:12 of EPTP; the combined mappings created and used are associated with the current VPID and PCID (these are not changed by VMFUNC).[1] If the "enable VPID" VM-execution control is 0, an EPTP-switching VMFUNC invalidates combined mappings associated with VPID 0000H (for all PCIDs and for all EP4TA values, where EP4TA is the value of bits 51:12 of EPTP).

Because an EPTP-switching VMFUNC may change the translation of guest-physical addresses, it may affect use of the guest-physical address in CR3. The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration due to the translation of that guest-physical address through the new EPT paging structures. The following items provide details that apply if CR0.PG = 1:

- If 32-bit paging or IA-32e paging is in use (either CR4.PAE = 0 or IA32_EFER.LMA = 1), the next memory access with a linear address uses the translation of the guest-physical address in CR3 through the new EPT paging structures. As a result, this access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.
- If PAE paging is in use (CR4.PAE = 1 and IA32_EFER.LMA = 0), an EPTP-switching VMFUNC **does not** load the four page-directory-pointer-table entries (PDPTEs) from the guest-physical address in CR3. The logical processor continues to use the four guest-physical addresses already present in the PDPTEs. The guest-physical address in CR3 is not translated through the new EPT paging structures (until some operation that would load the PDPTEs).

  The EPTP-switching VMFUNC cannot itself cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during the translation of a guest-physical address in any of the PDPTEs. A subsequent memory access with a linear address uses the translation of the guest-physical address in the appropriate PDPTE through the new EPT paging structures. As a result, such an access may cause a VM exit due to an EPT violation or an EPT misconfiguration encountered during that translation.

25.8 UNRESTRICTED GUESTS

The first processors to support VMX operation require CR0.PE and CR0.PG to be 1 in VMX operation (see Section 23.8). This restriction implies that guest software cannot

1. If the "enable VPID" VM-execution control is 0, the current VPID is 0000H; if CR4.PCIDE = 0, the current PCID is 000H.

be run in unpaged protected mode or in real-address mode. Later processors support a VM-execution control called "unrestricted guest".[1] If this control is 1, CR0.PE and CR0.PG may be 0 in VMX non-root operation. Such processors allow guest software to run in unpaged protected mode or in real-address mode. The following items describe the behavior of such software:

- The MOV CR0 instructions does not cause a general-protection exception simply because it would set either CR0.PE and CR0.PG to 0. See Section 25.4 for details.
- A logical processor treats the values of CR0.PE and CR0.PG in VMX non-root operation just as it does outside VMX operation. Thus, if CR0.PE = 0, the processor operates as it does normally in real-address mode (for example, it uses the 16-bit **interrupt table** to deliver interrupts and exceptions). If CR0.PG = 0, the processor operates as it does normally when paging is disabled.
- Processor operation is modified by the fact that the processor is in VMX non-root operation and by the settings of the VM-execution controls just as it is in protected mode or when paging is enabled. Instructions, interrupts, and exceptions that cause VM exits in protected mode or when paging is enabled also do so in real-address mode or when paging is disabled. The following examples should be noted:
  - If CR0.PG = 0, page faults do not occur and thus cannot cause VM exits.
  - If CR0.PE = 0, invalid-TSS exceptions do not occur and thus cannot cause VM exits.
  - If CR0.PE = 0, the following instructions cause invalid-opcode exceptions and do not cause VM exits: INVEPT, INVVPID, LLDT, LTR, SLDT, STR, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.
- If CR0.PG = 0, each linear address is passed directly to the EPT mechanism for translation to a physical address.[2] The guest memory type passed on to the EPT mechanism is WB (writeback).

1. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VMX non-root operation functions as if the "unrestricted guest" VM-execution control were 0. See Section 24.6.2.
2. As noted in Section 26.2.1.1, the "enable EPT" VM-execution control must be 1 if the "unrestricted guest" VM-execution control is 1.

CHAPTER 26
VM ENTRIES

Software can enter VMX non-root operation using either of the VM-entry instructions VMLAUNCH and VMRESUME. VMLAUNCH can be used only with a VMCS whose launch state is clear and VMRESUME can be used only with a VMCS whose the launch state is launched. VMLAUNCH should be used for the first VM entry after VMCLEAR; VMRESUME should be used for subsequent VM entries with the same VMCS.

Each VM entry performs the following steps in the order indicated:

1. Basic checks are performed to ensure that VM entry can commence (Section 26.1).
2. The control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation and that the VMCS is correctly configured to support the next VM exit (Section 26.2).
3. The following may be performed in parallel or in any order (Section 26.3):
   - The guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures.
   - Processor state is loaded from the guest-state area and based on controls in the VMCS.
   - Address-range monitoring is cleared.
4. MSRs are loaded from the VM-entry MSR-load area (Section 26.4).
5. If VMLAUNCH is being executed, the launch state of the VMCS is set to "launched."
6. An event may be injected in the guest context (Section 26.5).

Steps 1–4 above perform checks that may cause VM entry to fail. Such failures occur in one of the following three ways:

- Some of the checks in Section 26.1 may generate ordinary faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 26.1 and all the checks in Section 26.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF[1] (if there is a current VMCS) or RFLAGS.CF (if there is no current VMCS). If there is a current VMCS, an error number indicating the cause of the failure is stored in the VM-instruction error field. See Chapter 33

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For IA-32 processors, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* for the error numbers.

- The checks in Section 26.3 and Section 26.4 cause processor state to be loaded from the host-state area of the VMCS (as would be done on a VM exit). Information about the failure is stored in the VM-exit information fields. See Section 26.7 for details.

EFLAGS.TF = 1 causes a VM-entry instruction to generate a single-step debug exception only if failure of one of the checks in Section 26.1 and Section 26.2 causes control to pass to the following instruction. A VM-entry does not generate a single-step debug exception in any of the following cases: (1) the instruction generates a fault; (2) failure of one of the checks in Section 26.3 or in loading MSRs causes processor state to be loaded from the host-state area of the VMCS; or (3) the instruction passes all checks in Section 26.1, Section 26.2, and Section 26.3 and there is no failure in loading MSRs.

Section 29.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, code running in SMM returns using VM entries instead of the RSM instruction. A VM entry **returns from SMM** if it is executed in SMM and the "entry to SMM" VM-entry control is 0. VM entries that return from SMM differ from ordinary VM entries in ways that are detailed in Section 29.15.4.

26.1 BASIC VM-ENTRY CHECKS

Before a VM entry commences, the current state of the logical processor is checked in the following order:

1. If the logical processor is in virtual-8086 mode or compatibility mode, an invalid-opcode exception is generated.
2. If the current privilege level (CPL) is not zero, a general-protection exception is generated.
3. If there is no current VMCS, RFLAGS.CF is set to 1 and control passes to the next instruction.
4. If there is a current VMCS, the following conditions are evaluated in order; any of these cause VM entry to fail:
    a. if there is MOV-SS blocking (see Table 24-3)
    b. if the VM entry is invoked by VMLAUNCH and the VMCS launch state is not clear
    c. if the VM entry is invoked by VMRESUME and the VMCS launch state is not launched

    If any of these checks fail, RFLAGS.ZF is set to 1 and control passes to the next instruction. An error number indicating the cause of the failure is stored in the

VM-instruction error field. See Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* for the error numbers.

26.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 26.1 do not cause VM entry to fail, the control and host-state areas of the VMCS are checked to ensure that they are proper for supporting VMX non-root operation, that the VMCS is correctly configured to support the next VM exit, and that, after the next VM exit, the processor's state is consistent with the Intel 64 and IA-32 architectures.

VM entry fails if any of these checks fail. When such failures occur, control is passed to the next instruction, RFLAGS.ZF is set to 1 to indicate the failure, and the VM-instruction error field is loaded with an error number that indicates whether the failure was due to the controls or the host-state area (see Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, host state) does not imply that there are not also other errors. Different processors may thus give different error numbers for the same VMCS. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The checks on the controls and the host-state area are presented in Section 26.2.1 through Section 26.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

26.2.1 Checks on VMX Controls

This section identifies VM-entry checks on the VMX control fields.

26.2.1.1 VM-Execution Control Fields

VM entries perform the following checks on the VM-execution control fields:[1]

- Reserved bits in the pin-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.1).

1. If the "activate secondary controls" primary processor-based VM-execution control is 0, VM entry operates as if each secondary processor-based VM-execution control were 0.

- Reserved bits in the primary processor-based VM-execution controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.3.2).
- If the "activate secondary controls" primary processor-based VM-execution control is 1, reserved bits in the secondary processor-based VM-execution controls must be cleared. Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.3.3).

  If the "activate secondary controls" primary processor-based VM-execution control is 0 (or if the processor does not support the 1-setting of that control), no checks are performed on the secondary processor-based VM-execution controls. The logical processor operates as if all the secondary processor-based VM-execution controls were 0.
- The CR3-target count must not be greater than 4. Future processors may support a different number of CR3-target values. Software should read the VMX capability MSR IA32_VMX_MISC to determine the number of values supported (see Appendix A.6).
- If the "use I/O bitmaps" VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. Neither address should set any bits beyond the processor's physical-address width.[1,2]
- If the "use MSR bitmaps" VM-execution control is 1, bits 11:0 of the MSR-bitmap address must be 0. The address should not set any bits beyond the processor's physical-address width.[3]
- If the "use TPR shadow" VM-execution control is 1, the virtual-APIC address must satisfy the following checks:
  - Bits 11:0 of the address must be 0.
  - The address should not set any bits beyond the processor's physical-address width.[4]

  The following items describe the treatment of bytes 81H-83H on the virtual-APIC page (see Section 24.6.8) if all of the above checks are satisfied and the "use TPR shadow" VM-execution control is 1, treatment depends upon the setting of the "virtualize APIC accesses" VM-execution control:[5]
  - If the "virtualize APIC accesses" VM-execution control is 0, the bytes may be cleared. (If the bytes are not cleared, they are left unmodified.)

---

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
2. If IA32_VMX_BASIC[48] is read as 1, these addresses must not set any bits in the range 63:32; see Appendix A.1.
3. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
4. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.

  - If the “virtualize APIC accesses” VM-execution control is 1, the bytes are cleared.
  - If the VM entry fails, the any clearing of the bytes may or may not occur. This is true either if the failure causes control to pass to the instruction following the VM-entry instruction or if it cause processor state to be loaded from the host-state area of the VMCS. Behavior may be implementation-specific.
- If the “use TPR shadow” VM-execution control is 1, bits 31:4 of the TPR threshold VM-execution control field must be 0.
- The following check is performed if the “use TPR shadow” VM-execution control is 1 and the “virtualize APIC accesses” VM-execution control is 0: the value of bits 3:0 of the TPR threshold VM-execution control field should not be greater than the value of bits 7:4 in byte 80H on the virtual-APIC page (see Section 24.6.8).
- If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” VM-execution control must be 0.
- If the “virtual NMIs” VM-execution control is 0, the “NMI-window exiting” VM-execution control must be 0.
- If the “virtualize APIC-accesses” VM-execution control is 1, the APIC-access address must satisfy the following checks:
  - Bits 11:0 of the address must be 0.
  - The address should not set any bits beyond the processor’s physical-address width.[1]
- If the “virtualize x2APIC mode” VM-execution control is 1, the “use TPR shadow” VM-execution control must be 1 and the “virtualize APIC accesses” VM-execution control must be 0.[2]
- If the “enable VPID” VM-execution control is 1, the value of the VPID VM-execution control field must not be 0000H.[3]
- If the “enable EPT” VM-execution control is 1, the EPTP VM-execution control field (see Table 24-8 in Section 24.6.11) must satisfy the following checks:[4]

---

5. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 24.6.2.
1. If IA32_VMX_BASIC[48] is read as 1, this address must not set any bits in the range 63:32; see Appendix A.1.
2. “Virtualize x2APIC mode” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize x2APIC mode” VM-execution control were 0. See Section 24.6.2.
3. “Enable VPID” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “enable VPID” VM-execution control were 0. See Section 24.6.2.

 - — The EPT memory type (bits 2:0) must be a value supported by the logical processor as indicated in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A.10).
 - — Bits 5:3 (1 less than the EPT page-walk length) must be 3, indicating an EPT page-walk length of 4; see Section 28.2.2.
 - — Reserved bits 11:6 and 63:N (where N is the processor's physical-address width) must all be 0.
 - — If the "unrestricted guest" VM-execution control is 1, the "enable EPT" VM-execution control must also be 1.[1]

- If the "enable VM functions" processor-based VM-execution control is 1, reserved bits in the VM-function controls must be clear.[2] Software may consult the VMX capability MSRs to determine which bits are reserved (see Appendix A.11). In addition, the following check is performed based on the setting of bits in the VM-function controls (see Section 24.6.14):
  - — If "EPTP switching" VM-function control is 1, the "enable EPT" VM-execution control must also 1. In addition, the EPTP-list address must satisfy the following checks:
    - Bits 11:0 of the address must be 0.
    - The address must not set any bits beyond the processor's physical-address width.

  If the "enable VM functions" processor-based VM-execution control is 0, no checks are performed on the VM-function controls.

26.2.1.2 VM-Exit Control Fields

VM entries perform the following checks on the VM-exit control fields.

- Reserved bits in the VM-exit controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.4).
- If "activate VMX-preemption timer" VM-execution control is 0, the "save VMX-preemption timer value" VM-exit control must also be 0.

4. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

1. "Unrestricted guest" and "enable EPT" are both secondary processor-based VM-execution controls. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if both these controls were 0. See Section 24.6.2.

2. "Enable VM functions" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable VM functions" VM-execution control were 0. See Section 24.6.2.

- The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
  - The lower 4 bits of the VM-exit MSR-store address must be 0. The address should not set any bits beyond the processor's physical-address width.[1]
  - The address of the last byte in the VM-exit MSR-store area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

  If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.
- The following checks are performed for the VM-exit MSR-load address if the VM-exit MSR-load count field is non-zero:
  - The lower 4 bits of the VM-exit MSR-load address must be 0. The address should not set any bits beyond the processor's physical-address width.
  - The address of the last byte in the VM-exit MSR-load area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-exit MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

  If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

26.2.1.3 VM-Entry Control Fields

VM entries perform the following checks on the VM-entry control fields.

- Reserved bits in the VM-entry controls must be set properly. Software may consult the VMX capability MSRs to determine the proper settings (see Appendix A.5).
- Fields relevant to VM-entry event injection must be set properly. These fields are the VM-entry interruption-information field (see Table 24-13 in Section 24.8.3), the VM-entry exception error code, and the VM-entry instruction length. If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the following must hold:
  - The field's interruption type (bits 10:8) is not set to a reserved value. Value 1 is reserved on all logical processors; value 7 (other event) is reserved on logical processors that do not support the 1-setting of the "monitor trap flag" VM-execution control.

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

  - The field’s vector (bits 7:0) is consistent with the interruption type:
    - If the interruption type is non-maskable interrupt (NMI), the vector is 2.
    - If the interruption type is hardware exception, the vector is at most 31.
    - If the interruption type is other event, the vector is 0 (pending MTF VM exit).
  - The field's deliver-error-code bit (bit 11) is 1 if and only if (1) either (a) the "unrestricted guest" VM-execution control is 0; or (b) bit 0 (corresponding to CR0.PE) is set in the CR0 field in the guest-state area; (2) the interruption type is hardware exception; and (3) the vector indicates an exception that would normally deliver an error code (8 = #DF; 10 = TS; 11 = #NP; 12 = #SS; 13 = #GP; 14 = #PF; or 17 = #AC).
  - Reserved bits in the field (30:12) are 0.
  - If the deliver-error-code bit (bit 11) is 1, bits 31:15 of the VM-entry exception error-code field are 0.
  - If the interruption type is software interrupt, software exception, or privileged software exception, the VM-entry instruction-length field is in the range 1–15.
- The following checks are performed for the VM-entry MSR-load address if the VM-entry MSR-load count field is non-zero:
  - The lower 4 bits of the VM-entry MSR-load address must be 0. The address should not set any bits beyond the processor’s physical-address width.[1]
  - The address of the last byte in the VM-entry MSR-load area should not set any bits beyond the processor’s physical-address width. The address of this last byte is VM-entry MSR-load address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor’s physical-address width.)

  If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.
- If the processor is not in SMM, the “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls must be 0.
- The “entry to SMM” and “deactivate dual-monitor treatment” VM-entry controls cannot both be 1.

26.2.2 Checks on Host Control Registers and MSRs

The following checks are performed on fields in the host-state area that correspond to control registers and MSRs:

1. Software can determine a processor’s physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 23.8).[1]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 23.8).
- On processors that support Intel 64 architecture, the CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width must be 0.[2,3]
- On processors that support Intel 64 architecture, the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.
- If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 18-3).
- If the "load IA32_PAT" VM-exit control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the "load IA32_EFER" VM-exit control is 1, bits reserved in the IA32_EFER MSR must be 0 in the field for that register. In addition, the values of the LMA and LME bits in the field must each be that of the "host address-space size" VM-exit control.

26.2.3 Checks on Host Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area that correspond to segment and descriptor-table registers:

- In the selector field for each of CS, SS, DS, ES, FS, GS and TR, the RPL (bits 1:0) and the TI flag (bit 2) must be 0.
- The selector fields for CS and TR cannot be 0000H.
- The selector field for SS cannot be 0000H if the "host address-space size" VM-exit control is 0.
- On processors that support Intel 64 architecture, the base-address fields for FS, GS, GDTR, IDTR, and TR must contain canonical addresses.

1. The bits corresponding to CR0.NW (bit 29) and CR0.CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 27.5.1.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. Bit 63 of the CR3 field in the host-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.

26.2.4 Checks Related to Address-Space Size

On processors that support Intel 64 architecture, the following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If the logical processor is outside IA-32e mode (if IA32_EFER.LMA = 0) at the time of VM entry, the following must hold:
  - The "IA-32e mode guest" VM-entry control is 0.
  - The "host address-space size" VM-exit control is 0.
- If the logical processor is in IA-32e mode (if IA32_EFER.LMA = 1) at the time of VM entry, the "host address-space size" VM-exit control must be 1.
- If the "host address-space size" VM-exit control is 0, the following must hold:
  - The "IA-32e mode guest" VM-entry control is 0.
  - Bit 17 of the CR4 field (corresponding to CR4.PCIDE) is 0.
  - Bits 63:32 in the RIP field is 0.
- If the "host address-space size" VM-exit control is 1, the following must hold:
  - Bit 5 of the CR4 field (corresponding to CR4.PAE) is 1.
  - The RIP field contains a canonical address.

On processors that do not support Intel 64 architecture, checks are performed to ensure that the "IA-32e mode guest" VM-entry control and the "host address-space size" VM-exit control are both 0.

26.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 26.2), the following operations take place concurrently: (1) the guest-state area of the VMCS is checked to ensure that, after the VM entry completes, the state of the logical processor is consistent with IA-32 and Intel 64 architectures; (2) processor state is loaded from the guest-state area or as specified by the VM-entry control fields; and (3) address-range monitoring is cleared.

Because the checking and the loading occur concurrently, a failure may be discovered only after some state has been loaded. For this reason, the logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 26.7.

26.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area. These checks may be performed in any order. Some checks prevent establishment of settings (or combinations of settings) that are currently reserved. Future processors may allow such settings (or combinations) and may not perform the corresponding checks. The correctness of software should not rely on VM-entry failures resulting from the checks documented in this section.

The following subsections reference fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

26.3.1.1 Checks on Guest Control Registers, Debug Registers, and MSRs

The following checks are performed on fields in the guest-state area corresponding to control registers, debug registers, and MSRs:

- The CR0 field must not set any bit to a value not supported in VMX operation (see Section 23.8). The following are exceptions:
  - Bit 0 (corresponding to CR0.PE) and bit 31 (PG) are not checked if the "unrestricted guest" VM-execution control is 1.[1]
  - Bit 29 (corresponding to CR0.NW) and bit 30 (CD) are never checked because the values of these bits are not changed by VM entry; see Section 26.3.2.1.
- If bit 31 in the CR0 field (corresponding to PG) is 1, bit 0 in that field (PE) must also be 1.[2]
- The CR4 field must not set any bit to a value not supported in VMX operation (see Section 23.8).
- If the "load debug controls" VM-entry control is 1, bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally.
- The following checks are performed on processors that support Intel 64 architecture:
  - If the "IA-32e mode guest" VM-entry control is 1, bit 31 in the CR0 field (corresponding to CR0.PG) and bit 5 in the CR4 field (corresponding to CR4.PAE) must each be 1.[3]
  - If the "IA-32e mode guest" VM-entry control is 0, bit 17 in the CR4 field (corresponding to CR4.PCIDE) must each be 0.

1. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "unrestricted guest" VM-execution control were 0. See Section 24.6.2.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
3. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- — The CR3 field must be such that bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width are 0.[1,2]
- — If the "load debug controls" VM-entry control is 1, bits 63:32 in the DR7 field must be 0. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus performed this check unconditionally (if they supported Intel 64 architecture).
- — The IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field must each contain a canonical address.

- If the "load IA32_PERF_GLOBAL_CTRL" VM-entry control is 1, bits reserved in the IA32_PERF_GLOBAL_CTRL MSR must be 0 in the field for that register (see Figure 18-3).
- If the "load IA32_PAT" VM-entry control is 1, the value of the field for the IA32_PAT MSR must be one that could be written by WRMSR without fault at CPL 0. Specifically, each of the 8 bytes in the field must have one of the values 0 (UC), 1 (WC), 4 (WT), 5 (WP), 6 (WB), or 7 (UC-).
- If the "load IA32_EFER" VM-entry control is 1, the following checks are performed on the field for the IA32_EFER MSR :
  - — Bits reserved in the IA32_EFER MSR must be 0.
  - — Bit 10 (corresponding to IA32_EFER.LMA) must equal the value of the "IA-32e mode guest" VM-exit control. It must also be identical to bit 8 (LME) if bit 31 in the CR0 field (corresponding to CR0.PG) is 1.[3]

26.3.1.2 Checks on Guest Segment Registers

This section specifies the checks on the fields for CS, SS, DS, ES, FS, GS, TR, and LDTR. The following terms are used in defining these checks:

- The guest will be **virtual-8086** if the VM flag (bit 17) is 1 in the RFLAGS field in the guest-state area.
- The guest will be **IA-32e mode** if the "IA-32e mode guest" VM-entry control is 1. (This is possible only on processors that support Intel 64 architecture.)
- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

---

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
2. Bit 63 of the CR3 field in the guest-state area must be 0. This is true even though, If CR4.PCIDE = 1, bit 63 of the source operand to MOV to CR3 is used to determine whether cached translation information is invalidated.
3. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, bit 31 in the CR0 field must be 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- Selector fields.
  - — TR. The TI flag (bit 2) must be 0.
  - — LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
  - — SS. If the guest will not be virtual-8086 and the "unrestricted guest" VM-execution control is 0, the RPL (bits 1:0) must equal the RPL of the selector field for CS.[1]
- Base-address fields.
  - — CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the address must be the selector field shifted left 4 bits (multiplied by 16).
  - — The following checks are performed on processors that support Intel 64 architecture:
    - TR, FS, GS. The address must be canonical.
    - LDTR. If LDTR is usable, the address must be canonical.
    - CS. Bits 63:32 of the address must be zero.
    - SS, DS, ES. If the register is usable, bits 63:32 of the address must be zero.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
  - — CS, SS, DS, ES, FS, GS.
    - If the guest will be virtual-8086, the field must be 000000F3H. This implies the following:
      - — Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
      - — Bit 4 (S) must be 1.
      - — Bits 6:5 (DPL) must be 3.
      - — Bit 7 (P) must be 1.
      - — Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.
    - If the guest will not be virtual-8086, the different sub-fields are considered separately:
      - — Bits 3:0 (Type).

1. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "unrestricted guest" VM-execution control were 0. See Section 24.6.2.

  - CS. The values allowed depend on the setting of the "unrestricted guest" VM-execution control:
    - If the control is 0, the Type must be 9, 11, 13, or 15 (accessed code segment).
    - If the control is 1, the Type must be either 3 (read/write accessed expand-up data segment) or one of 9, 11, 13, and 15 (accessed code segment).
  - SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
  - DS, ES, FS, GS. The following checks apply if the register is usable:
    - Bit 0 of the Type must be 1 (accessed).
    - If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
- Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
- Bits 6:5 (DPL).
  - CS.
    - If the Type is 3 (read/write accessed expand-up data segment), the DPL must be 0. The Type can be 3 only if the "unrestricted guest" VM-execution control is 1.
    - If the Type is 9 or 11 (non-conforming code segment), the DPL must equal the DPL in the access-rights field for SS.
    - If the Type is 13 or 15 (conforming code segment), the DPL cannot be greater than the DPL in the access-rights field for SS.
  - SS.
    - If the "unrestricted guest" VM-execution control is 0, the DPL must equal the RPL from the selector field.
    - The DPL must be 0 either if the Type in the access-rights field for CS is 3 (read/write accessed expand-up data segment) or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.[1]
  - DS, ES, FS, GS. The DPL cannot be less than the RPL in the selector field if (1) the "unrestricted guest" VM-execution control is 0; (2) the register is usable; and (3) the Type in the access-

1. The following apply if either the "unrestricted guest" VM-execution control or bit 31 of the primary processor-based VM-execution controls is 0: (1) bit 0 in the CR0 field must be 1 if the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation; and (2) the Type in the access-rights field for CS cannot be 3.

rights field is in the range 0 – 11 (data segment or non-conforming code segment).

- Bit 7 (P). If the register is CS or if the register is usable, P must be 1.
- Bits 11:8 (reserved). If the register is CS or if the register is usable, these bits must all be 0.
- Bit 14 (D/B). For CS, D/B must be 0 if the guest will be IA-32e mode and the L bit (bit 13) in the access-rights field is 1.
- Bit 15 (G). The following checks apply if the register is CS or if the register is usable:
    - If any bit in the limit field in the range 11:0 is 0, G must be 0.
    - If any bit in the limit field in the range 31:20 is 1, G must be 1.
- Bits 31:17 (reserved). If the register is CS or if the register is usable, these bits must all be 0.

— TR. The different sub-fields are considered separately:

- Bits 3:0 (Type).
    - If the guest will not be IA-32e mode, the Type must be 3 (16-bit busy TSS) or 11 (32-bit busy TSS).
    - If the guest will be IA-32e mode, the Type must be 11 (64-bit busy TSS).
- Bit 4 (S). S must be 0.
- Bit 7 (P). P must be 1.
- Bits 11:8 (reserved). These bits must all be 0.
- Bit 15 (G).
    - If any bit in the limit field in the range 11:0 is 0, G must be 0.
    - If any bit in the limit field in the range 31:20 is 1, G must be 1.
- Bit 16 (Unusable). The unusable bit must be 0.
- Bits 31:17 (reserved). These bits must all be 0.

— LDTR. The following checks on the different sub-fields apply only if LDTR is usable:

- Bits 3:0 (Type). The Type must be 2 (LDT).
- Bit 4 (S). S must be 0.
- Bit 7 (P). P must be 1.
- Bits 11:8 (reserved). These bits must all be 0.
- Bit 15 (G).
    - If any bit in the limit field in the range 11:0 is 0, G must be 0.

    — If any bit in the limit field in the range 31:20 is 1, G must be 1.
  - Bits 31:17 (reserved). These bits must all be 0.

26.3.1.3 Checks on Guest Descriptor-Table Registers

The following checks are performed on the fields for GDTR and IDTR:

- On processors that support Intel 64 architecture, the base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

26.3.1.4 Checks on Guest RIP and RFLAGS

The following checks are performed on fields in the guest-state area corresponding to RIP and RFLAGS:

- RIP. The following checks are performed on processors that support Intel 64 architecture:
  — Bits 63:32 must be 0 if the “IA-32e mode guest” VM-entry control is 0 or if the L bit (bit 13) in the access-rights field for CS is 0.
  — If the processor supports N < 64 linear-address bits, bits 63:N must be identical if the “IA-32e mode guest” VM-entry control is 1 and the L bit in the access-rights field for CS is 1.[1] (No check applies if the processor supports 64 linear-address bits.)
- RFLAGS.
  — Reserved bits 63:22 (bits 31:22 on processors that do not support Intel 64 architecture), bit 15, bit 5 and bit 3 must be 0 in the field, and reserved bit 1 must be 1.
  — The VM flag (bit 17) must be 0 either if the “IA-32e mode guest” VM-entry control is 1 or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.[2]
  — The IF flag (RFLAGS[bit 9]) must be 1 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) is external interrupt.

26.3.1.5 Checks on Guest Non-Register State

The following checks are performed on fields in the guest-state area corresponding to non-register state:

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, bit 0 in the CR0 field must be 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- Activity state.
  - The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 24.4.2). Future processors may include support for other activity states. Software should read the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine what activity states are supported.
  - The activity-state field must not indicate the HLT state if the DPL (bits 6:5) in the access-rights field for SS is not 0.[1]
  - The activity-state field must indicate the active state if the interruptibility-state field indicates blocking by either MOV-SS or by STI (if either bit 0 or bit 1 in that field is 1).
  - If the valid bit (bit 31) in the VM-entry interruption-information field is 1, the interruption to be delivered (as defined by interruption type and vector) must not be one that would normally be blocked while a logical processor is in the activity state corresponding to the contents of the activity-state field. The following items enumerate the interruptions (as specified in the VM-entry interruption-information field) whose injection is allowed for the different activity states:
    - Active. Any interruption is allowed.
    - HLT. The only events allowed are the following:
      - Those with interruption type external interrupt or non-maskable interrupt (NMI).
      - Those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
      - Those with interruption type other event and vector 0 (pending MTF VM exit).

      See Table 24-13 in Section 24.8.3 for details regarding the format of the VM-entry interruption-information field.
    - Shutdown. Only NMIs and machine-check exceptions are allowed.
    - Wait-for-SIPI. No interruptions are allowed.
  - The activity-state field must not indicate the wait-for-SIPI state if the “entry to SMM” VM-entry control is 1.
- Interruptibility state.
  - The reserved bits (bits 31:4) must be 0.
  - The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).

1. As noted in Section 24.4.1, SS.DPL corresponds to the logical processor’s current privilege level (CPL).

- — Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
- — Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 0, indicating external interrupt.
- — Bit 1 (blocking by MOV-SS) must be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 2, indicating non-maskable interrupt (NMI).
- — Bit 2 (blocking by SMI) must be 0 if the processor is not in SMM.
- — Bit 2 (blocking by SMI) must be 1 if the "entry to SMM" VM-entry control is 1.
- — A processor may require bit 0 (blocking by STI) to be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 2, indicating NMI. Other processors may not make this requirement.
- — Bit 3 (blocking by NMI) must be 0 if the "virtual NMIs" VM-execution control is 1, the valid bit (bit 31) in the VM-entry interruption-information field is 1, and the interruption type (bits 10:8) in that field has value 2 (indicating NMI).

NOTE

If the "virtual NMIs" VM-execution control is 0, there is no requirement that bit 3 be 0 if the valid bit in the VM-entry interruption-information field is 1 and the interruption type in that field has value 2.

- Pending debug exceptions.
  - — Bits 11:4, bit 13, and bits 63:15 (bits 31:15 on processors that do not support Intel 64 architecture) must be 0.
  - — The following checks are performed if any of the following holds: (1) the interruptibility-state field indicates blocking by STI (bit 0 in that field is 1); (2) the interruptibility-state field indicates blocking by MOV SS (bit 1 in that field is 1); or (3) the activity-state field indicates HLT:
    - Bit 14 (BS) must be 1 if the TF flag (bit 8) in the RFLAGS field is 1 and the BTF flag (bit 1) in the IA32_DEBUGCTL field is 0.
    - Bit 14 (BS) must be 0 if the TF flag (bit 8) in the RFLAGS field is 0 or the BTF flag (bit 1) in the IA32_DEBUGCTL field is 1.
- VMCS link pointer. The following checks apply if the field contains a value other than FFFFFFFF_FFFFFFFFH:
  - — Bits 11:0 must be 0.
  - — Bits beyond the processor's physical-address width must be 0.[1,2]

- — The 32 bits located in memory referenced by the value of the field (as a physical address) must contain the processor's VMCS revision identifier (see Section 24.2).
- — If the processor is not in SMM or the "entry to SMM" VM-entry control is 1, the field must not contain the current VMCS pointer.
- — If the processor is in SMM and the "entry to SMM" VM-entry control is 0, the field must not contain the VMXON pointer.

26.3.1.6 Checks on Guest Page-Directory-Pointer-Table Entries

If CR0.PG =1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor also uses **PAE paging** (see Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).[1] When PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs.

A VM entry is to a guest that uses PAE paging if (1) bit 31 (corresponding to CR0.PG) is set in the CR0 field in the guest-state area; (2) bit 5 (corresponding to CR4.PAE) is set in the CR4 field; and (3) the "IA-32e mode guest" VM-entry control is 0. Such a VM entry checks the validity of the PDPTEs:

- If the "enable EPT" VM-execution control is 0, VM entry checks the validity of the PDPTEs referenced by the CR3 field in the guest-state area if either (1) PAE paging was not in use before the VM entry; or (2) the value of CR3 is changing as a result of the VM entry. VM entry may check their validity even if neither (1) nor (2) hold.[2]
- If the "enable EPT" VM-execution control is 1, VM entry checks the validity of the PDPTE fields in the guest-state area (see Section 24.4.2).

A VM entry to a guest that does not use PAE paging does not check the validity of any PDPTEs.

A VM entry that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use.[3] If MOV to CR3

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
2. If IA32_VMX_BASIC[48] is read as 1, this field must not set any bits in the range 63:32; see Appendix A.1.

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine the number physical-address bits supported by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
2. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

26.3.2 Loading Guest State

Processor state is updated on VM entries in the following ways:

- Some state is loaded from the guest-state area.
- Some state is determined by VM-entry controls.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order and in parallel with the checking of VMCS contents (see Section 26.3.1).

The loading of guest state is detailed in Section 26.3.2.1 to Section 26.3.2.4. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

In addition to the state loading described in this section, VM entries may load MSRs from the VM-entry MSR-load area (see Section 26.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 26.3.1.

26.3.2.1 Loading Guest Control Registers, Debug Registers, and MSRs

The following items describe how guest control registers, debug registers, and MSRs are loaded on VM entry:

- CR0 is loaded from the CR0 field with the exception of the following bits, which are never modified on VM entry: ET (bit 4); reserved bits 15:6, 17, and 28:19; NW (bit 29) and CD (bit 30).[1] The values of these bits in the CR0 field are ignored.
- CR3 and CR4 are loaded from the CR3 field and the CR4 field, respectively.
- If the "load debug controls" VM-execution control is 1, DR7 is loaded from the DR7 field with the exception that bit 12 and bits 15:14 are always 0 and bit 10 is always 1. The values of these bits in the DR7 field are ignored.

  The first processors to support the virtual-machine extensions supported only the 1-setting of the "load debug controls" VM-execution control and thus always loaded DR7 from the DR7 field.

3. This implies that (1) bits 11:9 in each PDPTE are ignored; and (2) if bit 0 (present) is clear in one of the PDPTEs, bits 63:1 of that PDPTE are ignored.

1. Bits 15:6, bit 17, and bit 28:19 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. Bits 15:6, bit 17, and bit 28:19 of CR0 are always 0 and CR0.ET is always 1.

- The following describes how some MSRs are loaded using fields in the guest-state area:
  - — If the “load debug controls” VM-execution control is 1, the IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always loaded the IA32_DEBUGCTL MSR from the IA32_DEBUGCTL field.
  - — The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since this field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
  - — The IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively. On processors that do not support Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.
  - — The following are performed on processors that support Intel 64 architecture:
    - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 26.3.2.2).
    - If the “load IA32_EFER” VM-entry control is 0, bits in the IA32_EFER MSR are modified as follows:
      - — IA32_EFER.LMA is loaded with the setting of the “IA-32e mode guest” VM-entry control.
      - — If CR0 is being loaded so that CR0.PG = 1, IA32_EFER.LME is also loaded with the setting of the “IA-32e mode guest” VM-entry control.[1] Otherwise, IA32_EFER.LME is unmodified.

      See below for the case in which the “load IA32_EFER” VM-entry control is 1
  - — If the “load IA32_PERF_GLOBAL_CTRL” VM-entry control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field.
  - — If the “load IA32_PAT” VM-entry control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field.
  - — If the “load IA32_EFER” VM-entry control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field.

  With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 26.4.
- The SMBASE register is unmodified by all VM entries except those that return from SMM.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, VM entry must be loading CR0 so that CR0.PG = 1 unless the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

26.3.2.2 Loading Guest Segment Registers and Descriptor-Table Registers

For each of CS, SS, DS, ES, FS, GS, TR, and LDTR, fields are loaded from the guest-state area as follows:

- The unusable bit is loaded from the access-rights field. This bit can never be set for TR (see Section 26.3.1.2). If it is set for one of the other registers, the following apply:
  - For each of CS, SS, DS, ES, FS, and GS, uses of the segment cause faults (general-protection exception or stack-fault exception) outside 64-bit mode, just as they would had the segment been loaded using a null selector. This bit does not cause accesses to fault in 64-bit mode.
  - If this bit is set for LDTR, uses of LDTR cause general-protection exceptions in all modes, just as they would had LDTR been loaded using a null selector.

  If this bit is clear for any of CS, SS, DS, ES, FS, GS, TR, and LDTR, a null selector value does not cause a fault (general-protection exception or stack-fault exception).
- TR. The selector, base, limit, and access-rights fields are loaded.
- CS.
  - The following fields are always loaded: selector, base address, limit, and (from the access-rights field) the L, D, and G bits.
  - For the other fields, the unusable bit of the access-rights field is consulted:
    - If the unusable bit is 0, all of the access-rights field is loaded.
    - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, GS, and LDTR.
  - The selector fields are loaded.
  - For the other fields, the unusable bit of the corresponding access-rights field is consulted:
    - If the unusable bit is 0, the base-address, limit, and access-rights fields are loaded.
    - If the unusable bit is 1, the base address, the segment limit, and the remainder of the access rights are undefined after VM entry with the following exceptions:
      - Bits 3:0 of the base address for SS are cleared to 0.
      - SS.DPL is always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
      - SS.B is always set to 1.
      - The base addresses for FS and GS are loaded from the corresponding fields in the VMCS. On processors that support Intel 64

architecture, the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.

— On processors that support Intel 64 architecture, the base address for LDTR is set to an undefined but canonical value.

— On processors that support Intel 64 architecture, bits 63:32 of the base addresses for SS, DS, and ES are cleared to 0.

GDTR and IDTR are loaded using the base and limit fields.

26.3.2.3 Loading Guest RIP, RSP, and RFLAGS

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively. The following items regard the upper 32 bits of these fields on VM entries that are not to 64-bit mode:

- Bits 63:32 of RSP are undefined outside 64-bit mode. Thus, a logical processor may ignore the contents of bits 63:32 of the RSP field on VM entries that are not to 64-bit mode.
- As noted in Section 26.3.1.4, bits 63:32 of the RIP and RFLAGS fields must be 0 on VM entries that are not to 64-bit mode.

26.3.2.4 Loading Page-Directory-Pointer-Table Entries

As noted in Section 26.3.1.6, the logical processor uses PAE paging if bit 5 in CR4 (CR4.PAE) is 1 and IA32_EFER.LMA is 0. A VM entry to a guest that uses PAE paging loads the PDPTEs into internal, non-architectural registers based on the setting of the "enable EPT" VM-execution control:

- If the control is 0, the PDPTEs are loaded from the page-directory-pointer table referenced by the physical address in the value of CR3 being loaded by the VM entry (see Section 26.3.2.1). The values loaded are treated as physical addresses in VMX non-root operation.
- If the control is 1, the PDPTEs are loaded from corresponding fields in the guest-state area (see Section 24.4.2). The values loaded are treated as guest-physical addresses in VMX non-root operation.

26.3.2.5 Updating Non-Register State

Section 28.3 describe how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM entries invalidate cached mappings:

- If the "enable VPID" VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).

- VM entries are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

26.3.3 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 8.10.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. VM entries clear any address-range monitoring that may be in effect.

26.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 24.8.2). Specifically each entry in that area (up to the number specified in the VM-entry MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.[1]

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101 (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM entry did not commence in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM entries for model-specific reasons. A processor may prevent loading of certain MSRs even if they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 34.
- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.[2]

1. Because attempts to modify the value of IA32_EFER.LMA by WRMSR are ignored, attempts to modify it using the VM-entry MSR-load area are also ignored.
2. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. If VM entry has established CR0.PG = 1, the IA32_EFER MSR should not be included in the VM-entry MSR-load area for the purpose of modifying the LME bit.

The VM entry fails if processing fails for any entry. The logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 26.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM entry, the logical processor will not use any translations that were cached before the transition.

26.5 EVENT INJECTION

If the valid bit in the VM-entry interruption-information field (see Section 24.8.3) is 1, VM entry causes an event to be delivered (or made pending) after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.

- If the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception), the event is delivered as described in Section 26.5.1.
- If the interruption type in the field is 7 (other event) and the vector field is 0, an MTF VM exit is pending after VM entry. See Section 26.5.2.

26.5.1 Vectored-Event Injection

VM entry delivers an injected vectored event within the guest context established by VM entry. This means that delivery occurs after all components of guest state have been loaded (including MSRs) and after the VM-execution control fields have been established.[1] The event is delivered using the vector in that field to select a descriptor in the IDT. Since event injection occurs after loading IDTR from the guest-state area, this is the guest IDT.

Section 26.5.1.1 provides details of vectored-event injection. In general, the event is delivered exactly as if it had been generated normally.

If event delivery encounters a nested exception (for example, a general-protection exception because the vector indicates a descriptor beyond the IDT limit), the exception bitmap is consulted using the vector of that exception. If the bit is 0, the exception is delivered through the IDT. If the bit is 1, a VM exit occurs. Section 26.5.1.2 details cases in which event injection causes a VM exit.

1. This does not imply that injection of an exception or interrupt will cause a VM exit due to the settings of VM-execution control fields (such as the exception bitmap) that would cause a VM exit if the event had occurred in VMX non-root operation. In contrast, a nested exception encountered during event delivery may cause a VM exit; see Section 26.5.1.1.

26.5.1.1 Details of Vectored-Event Injection

The event-injection process is controlled by the contents of the VM-entry interruption information field (format given in Table 24-13), the VM-entry exception error-code field, and the VM-entry instruction-length field. The following items provide details of the process:

- The value pushed on the stack for RFLAGS is generally that which was loaded from the guest-state area. The value pushed for the RF flag is not modified based on the type of event being delivered. However, the pushed value of RFLAGS may be modified if a software interrupt is being injected into a guest that will be in virtual-8086 mode (see below). After RFLAGS is pushed on the stack, the value in the RFLAGS register is modified as is done normally when delivering an event through the IDT.
- The instruction pointer that is pushed on the stack depends on the type of event and whether nested exceptions occur during its delivery. The term **current guest RIP** refers to the value to be loaded from the guest-state area. The value pushed is determined as follows:[1]
  - If VM entry successfully injects (with no nested exception) an event with interruption type external interrupt, NMI, or hardware exception, the current guest RIP is pushed on the stack.
  - If VM entry successfully injects (with no nested exception) an event with interruption type software interrupt, privileged software exception, or software exception, the current guest RIP is incremented by the VM-entry instruction length before being pushed on the stack.
  - If VM entry encounters an exception while injecting an event and that exception does not cause a VM exit, the current guest RIP is pushed on the stack regardless of event type or VM-entry instruction length. If the encountered exception does cause a VM exit that saves RIP, the saved RIP is current guest RIP.
- If the deliver-error-code bit (bit 11) is set in the VM-entry interruption-information field, the contents of the VM-entry exception error-code field is pushed on the stack as an error code would be pushed during delivery of an exception.
- DR6, DR7, and the IA32_DEBUGCTL MSR are not modified by event injection, even if the event has vector 1 (normal deliveries of debug exceptions, which have vector 1, do update these registers).
- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode (RFLAGS.VM = 1), no general-protection exception can occur due to RFLAGS.IOPL < 3. A VM monitor should check RFLAGS.IOPL before injecting such an event and, if desired, inject a general-protection exception instead of a software interrupt.

1. While these items refer to RIP, the width of the value pushed (16 bits, 32 bits, or 64 bits) is determined normally.

- If VM entry is injecting a software interrupt and the guest will be in virtual-8086 mode with virtual-8086 mode extensions (RFLAGS.VM = CR4.VME = 1), event delivery is subject to VME-based interrupt redirection based on the software interrupt redirection bitmap in the task-state segment (TSS) as follows:
  - If bit *n* in the bitmap is clear (where *n* is the number of the software interrupt), the interrupt is directed to an 8086 program interrupt handler: the processor uses a 16-bit interrupt-vector table (IVT) located at linear address zero. If the value of RFLAGS.IOPL is less than 3, the following modifications are made to the value of RFLAGS that is pushed on the stack: IOPL is set to 3, and IF is set to the value of VIF.
  - If bit *n* in the bitmap is set (where *n* is the number of the software interrupt), the interrupt is directed to a protected-mode interrupt handler. (In other words, the injection is treated as described in the next item.) In this case, the software interrupt does not invoke such a handler if RFLAGS.IOPL < 3 (a general-protection exception occurs instead). However, as noted above, RFLAGS.IOPL cannot cause an injected software interrupt to cause such a exception. Thus, in this case, the injection invokes a protected-mode interrupt handler independent of the value of RFLAGS.IOPL.

  Injection of events of other types are not subject to this redirection.
- If VM entry is injecting a software interrupt (not redirected as described above) or software exception, privilege checking is performed on the IDT descriptor being accessed as would be the case for executions of INT *n*, INT3, or INTO (the descriptor's DPL cannot be less than CPL). There is no checking of RFLAGS.IOPL, even if the guest will be in virtual-8086 mode. Failure of this check may lead to a nested exception. Injection of an event with interruption type external interrupt, NMI, hardware exception, and privileged software exception, or with interruption type software interrupt and being redirected as described above, do not perform these checks.
- If VM entry is injecting a non-maskable interrupt (NMI) and the "virtual NMIs" VM-execution control is 1, virtual-NMI blocking is in effect after VM entry.
- The transition causes a last-branch record to be logged if the LBR bit is set in the IA32_DEBUGCTL MSR. This is true even for events such as debug exceptions, which normally clear the LBR bit before delivery.
- The last-exception record MSRs (LERs) may be updated based on the setting of the LBR bit in the IA32_DEBUGCTL MSR. Events such as debug exceptions, which normally clear the LBR bit before they are delivered, and therefore do not normally update the LERs, may do so as part of VM-entry event injection.
- If injection of an event encounters a nested exception that does not itself cause a VM exit, the value of the EXT bit (bit 0) in any error code pushed on the stack is determined as follows:
  - If event being injected has interruption type external interrupt, NMI, hardware exception, or privileged software exception and encounters a nested exception (but does not produce a double fault), the error code for the first such exception encountered sets the EXT bit.

— If event being injected is a software interrupt or an software exception and encounters a nested exception (but does not produce a double fault), the error code for the first such exception encountered clears the EXT bit.

— If event delivery encounters a nested exception and delivery of that exception encounters another exception (but does not produce a double fault), the error code for that exception sets the EXT bit. If a double fault is produced, the error code for the double fault is 0000H (the EXT bit is clear).

26.5.1.2 VM Exits During Event Injection

An event being injected never causes a VM exit directly regardless of the settings of the VM-execution controls. For example, setting the “NMI exiting” VM-execution control to 1 does not cause a VM exit due to injection of an NMI.

However, the event-delivery process may lead to a VM exit:

- If the vector in the VM-entry interruption-information field identifies a task gate in the IDT, the attempted task switch may cause a VM exit just as it would had the injected event occurred during normal execution in VMX non-root operation (see Section 25.6.2).
- If event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap (see Section 25.3).
- If event delivery generates a double-fault exception (due to a nested exception); the logical processor encounters another nested exception while attempting to call the double-fault handler; and that exception does not cause a VM exit due to the exception bitmap; then a VM exit occurs due to triple fault (see Section 25.3).
- If event delivery injects a double-fault exception and encounters a nested exception that does not cause a VM exit due to the exception bitmap, then a VM exit occurs due to triple fault (see Section 25.3).
- If the “virtualize APIC accesses” VM-execution control is 1 and event delivery generates an access to the APIC-access page, that access may cause an APIC-access VM exit (see Section 25.2) or, if the access is a VTPR access, be treated as specified in Section 25.5.3.[1]

If the event-delivery process does cause a VM exit, the processor state before the VM exit is determined just as it would be had the injected event occurred during normal execution in VMX non-root operation. If the injected event directly accesses a task gate that cause a VM exit or if the first nested exception encountered causes a VM exit, information about the injected event is saved in the IDT-vectoring information field (see Section 27.2.3).

1. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 24.6.2.

26.5.1.3 Event Injection for VM Entries to Real-Address Mode

If VM entry is loading CR0.PE with 0, any injected vectored event is delivered as would normally be done in real-address mode.[1] Specifically, VM entry uses the vector provided in the VM-entry interruption-information field to select a 4-byte entry from an interrupt-vector table at the linear address in IDTR.base. Further details are provided in Section 15.1.4 in Volume 3A of the *IA-32 Intel® Architecture Software Developer's Manual*.

Because bit 11 (deliver error code) in the VM-entry interruption-information field must be 0 if CR0.PE will be 0 after VM entry (see Section 26.2.1.3), vectored events injected with CR0.PE = 0 do not push an error code on the stack. This is consistent with event delivery in real-address mode.

If event delivery encounters a fault (due to a violation of IDTR.limit or of SS.limit), the fault is treated as if it had occurred during event delivery in VMX non-root operation. Such a fault may lead to a VM exit as discussed in Section 26.5.1.2.

26.5.2 Injection of Pending MTF VM Exits

If the interruption type in the VM-entry interruption-information field is 7 (other event) and the vector field is 0, VM entry causes an MTF VM exit to be pending on the instruction boundary following VM entry. This is the case even if the "monitor trap flag" VM-execution control is 0. See Section 25.7.2 for the treatment of pending MTF VM exits.

26.6 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **vectoring** if the valid bit (bit 31) of the VM-entry interruption information field is 1 and the interruption type in the field is 0 (external interrupt), 2 (non-maskable interrupt); 3 (hardware exception), 4 (software interrupt), 5 (privileged software exception), or 6 (software exception).

26.6.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 24-3) contains bits that control blocking by STI, blocking by MOV SS, and blocking by NMI. This field impacts event blocking after VM entry as follows:

- If the VM entry is vectoring, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, VM entry must be loading CR0.PE with 1 unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- If the VM entry is not vectoring, the following apply:
  - Events are blocked by STI if and only if bit 0 in the interruptibility-state field is 1. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry; see Section 26.6.3).
  - Events are blocked by MOV SS if and only if bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 26.6.3. This blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).
- The blocking of non-maskable interrupts (NMIs) is determined as follows:
  - If the “virtual NMIs” VM-execution control is 0, NMIs are blocked if and only if bit 3 (blocking by NMI) in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, execution of the IRET instruction removes this blocking (even if the instruction generates a fault). If the “NMI exiting” control is 1, IRET does not affect this blocking.
  - The following items describe the use of bit 3 (blocking by NMI) in the interruptibility-state field if the “virtual NMIs” VM-execution control is 1:
    - The bit’s value does not affect the blocking of NMIs after VM entry. NMIs are not blocked in VMX non-root operation (except for ordinary blocking for other reasons, such as by the MOV SS instruction, the wait-for-SIPI state, etc.)
    - The bit’s value determines whether there is virtual-NMI blocking after VM entry. If the bit is 1, virtual-NMI blocking is in effect after VM entry. If the bit is 0, there is no virtual-NMI blocking after VM entry unless the VM entry is injecting an NMI (see Section 26.5.1.1). Execution of IRET removes virtual-NMI blocking (even if the instruction generates a fault).

  If the “NMI exiting” VM-execution control is 0, the “virtual NMIs” control must be 0; see Section 26.2.1.1.
- Blocking of system-management interrupts (SMIs) is determined as follows:
  - If the VM entry was not executed in system-management mode (SMM), SMI blocking is unchanged by VM entry.
  - If the VM entry was executed in SMM, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.

26.6.2 Activity State

The activity-state field in the guest-state area controls whether, after VM entry, the logical processor is active or in one of the inactive states identified in Section 24.4.2. The use of this field is determined as follows:

- If the VM entry is vectoring, the logical processor is in the active state after VM entry. While the consistency checks described in Section 26.3.1.5 on the activity-state field do apply in this case, the contents of the activity-state field do not determine the activity state after VM entry.
- If the VM entry is not vectoring, the logical processor ends VM entry in the activity state specified in the guest-state area. If VM entry ends with the logical processor in an inactive activity state, the VM entry generates any special bus cycle that is normally generated when that activity state is entered from the active state. If VM entry would end with the logical processor in the shutdown state and the logical processor is in SMX operation,[1] an Intel® TXT shutdown condition occurs. The error code used is 0000H, indicating "legacy shutdown." See *Intel® Trusted Execution Technology Preliminary Architecture Specification*.
- Some activity states unconditionally block certain events. The following blocking is in effect after any VM entry that puts the processor in the indicated state:
  - The active state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the active state and in VMX non-root operation are discarded and do not cause VM exits.
  - The HLT state blocks start-up IPIs (SIPIs). SIPIs that arrive while a logical processor is in the HLT state and in VMX non-root operation are discarded and do not cause VM exits.
  - The shutdown state blocks external interrupts and SIPIs. External interrupts that arrive while a logical processor is in the shutdown state and in VMX non-root operation do not cause VM exits even if the "external-interrupt exiting" VM-execution control is 1. SIPIs that arrive while a logical processor is in the shutdown state and in VMX non-root operation are discarded and do not cause VM exits.
  - The wait-for-SIPI state blocks external interrupts, non-maskable interrupts (NMIs), INIT signals, and system-management interrupts (SMIs). Such events do not cause VM exits if they arrive while a logical processor is in the wait-for-SIPI state and in VMX non-root operation do not cause VM exits regardless of the settings of the pin-based VM-execution controls.

26.6.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area indicates whether there are debug exceptions that have not yet been delivered (see Section 24.4.2). This section describes how these are treated on VM entry.

There are no pending debug exceptions after VM entry if any of the following are true:

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

- The VM entry is vectoring with one of the following interruption types: external interrupt, non-maskable interrupt (NMI), hardware exception, or privileged software exception.
- The interruptibility-state field does not indicate blocking by MOV SS and the VM entry is vectoring with either of the following interruption type: software interrupt or software exception.
- The VM entry is not vectoring and the activity-state field indicates either shutdown or wait-for-SIPI.

If none of the above hold, the pending debug exceptions field specifies the debug exceptions that are pending for the guest. There are **valid pending debug exceptions** if either the BS bit (bit 14) or the enable-breakpoint bit (bit 12) is 1. If there are valid pending debug exceptions, they are handled as follows:

- If the VM entry is not vectoring, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
  - If the logical processor is not blocking such exceptions (the interruptibility-state field indicates no blocking by MOV SS), a debug exception is delivered after VM entry (see below).
  - If the logical processor is blocking such exceptions (due to blocking by MOV SS), the pending debug exceptions are held pending or lost as would normally be the case.
- If the VM entry is vectoring (with interruption type software interrupt or software exception and with blocking by MOV SS), the following items apply:
  - For injection of a software interrupt or of a software exception with vector 3 (#BP) or vector 4 (#OF), the pending debug exceptions are treated as they would had they been encountered normally in guest execution if the corresponding instruction (INT3 or INTO) were executed after a MOV SS that encountered a debug trap.
  - For injection of a software exception with a vector other than 3 and 4, the pending debug exceptions may be lost or they may be delivered after injection (see below).

If there are no valid pending debug exceptions (as defined above), no pending debug exceptions are delivered after VM entry.

If a pending debug exception is delivered after VM entry, it has the priority of "traps on the previous instruction" (see Section 6.9 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*). Thus, INIT signals and system-management interrupts (SMIs) take priority of such an exception, as do VM exits induced by the TPR shadow (see Section 26.6.7) and pending MTF VM exits (see Section 26.6.8. The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt and also over VM exits due to the 1-settings of the "interrupt-window exiting" and "NMI-window exiting" VM-execution controls.

A pending debug exception delivered after VM entry causes a VM exit if the bit 1 (#DB) is 1 in the exception bitmap. If it does not cause a VM exit, it updates DR6 normally.

26.6.4 VMX-Preemption Timer

If the “activate VMX-preemption timer” VM-execution control is 1, VM entry starts the VMX-preemption timer with the unsigned value in the VMX-preemption timer-value field.

It is possible for the VMX-preemption timer to expire during VM entry (e.g., if the value in the VMX-preemption timer-value field is zero). If this happens (and if the VM entry was not to the wait-for-SIPI state), a VM exit occurs with its normal priority after any event injection and before execution of any instruction following VM entry. For example, any pending debug exceptions established by VM entry (see Section 26.6.3) take priority over a timer-induced VM exit. (The timer-induced VM exit will occur after delivery of the debug exception, unless that exception or its delivery causes a different VM exit.)

See Section 25.7.1 for details of the operation of the VMX-preemption timer in VMX non-root operation, including the blocking and priority of the VM exits that it causes.

26.6.5 Interrupt-Window Exiting

The “interrupt-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 25.3 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.
- Non-maskable interrupts (NMIs) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over external interrupts and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered the HLT state because of a VM entry (see Section 26.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

26.6.6 NMI-Window Exiting

The “NMI-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 25.3 for details).

The following items detail the treatment of these VM exits:

- These VM exits follow event injection if such injection is specified for VM entry.

- Debug-trap exceptions (see Section 26.6.3) and higher priority events take priority over VM exits caused by this control. VM exits caused by this control take priority over non-maskable interrupts (NMIs) and lower priority events.
- VM exits caused by this control wake the logical processor if it just entered either the HLT state or the shutdown state because of a VM entry (see Section 26.6.2). They do not occur if the logical processor just entered the wait-for-SIPI state.

26.6.7 VM Exits Induced by the TPR Shadow

If the “use TPR shadow” and “virtualize APIC accesses” VM-execution controls are both 1, a VM exit occurs immediately after VM entry if the value of bits 3:0 of the TPR threshold VM-execution control field is greater than the value of bits 7:4 in byte 80H on the virtual-APIC page (see Section 24.6.8).[1]

The following items detail the treatment of these VM exits:

- The VM exits are not blocked if RFLAGS.IF = 0 or by the setting of bits in the interruptibility-state field in guest-state area.
- The VM exits follow event injection if such injection is specified for VM entry.
- VM exits caused by this control take priority over system-management interrupts (SMIs), INIT signals, and lower priority events. They thus have priority over the VM exits described in Section 26.6.5, Section 26.6.6, and Section 26.6.8, as well as any interrupts or debug exceptions that may be pending at the time of VM entry.
- These VM exits wake the logical processor if it just entered the HLT state as part of a VM entry (see Section 26.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

  If such a VM exit is suppressed because the processor just entered the shutdown state, it occurs after the delivery of any event that cause the logical processor to leave the shutdown state while remaining in VMX non-root operation (e.g., due to an NMI that occurs while the “NMI-exiting” VM-execution control is 0).
- The basic exit reason is “TPR below threshold.”

26.6.8 Pending MTF VM Exits

As noted in Section 26.5.2, VM entry may cause an MTF VM exit to be pending immediately after VM entry. The following items detail the treatment of these VM exits:

1. “Virtualize APIC accesses” is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the “virtualize APIC accesses” VM-execution control were 0. See Section 24.6.2.

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these VM exits. These VM exits take priority over debug-trap exceptions and lower priority events.
- These VM exits wake the logical processor if it just entered the HLT state because of a VM entry (see Section 26.6.2). They do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

26.6.9 VM Entries and Advanced Debugging Features

VM entries are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

26.7 VM-ENTRY FAILURES DURING OR AFTER LOADING GUEST STATE

VM-entry failures due to the checks identified in Section 26.3.1 and failures during the MSR loading identified in Section 26.4 are treated differently from those that occur earlier in VM entry. In these cases, the following steps take place:

1. Information about the VM-entry failure is recorded in the VM-exit information fields:
   - — Exit reason.
     - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM-entry failure. The following numbers are used:

       33. VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 26.3.1.

       34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 26.4).

       41. VM-entry failure due to machine-check event. A machine-check event occurred during VM entry (see Section 26.8).
     - Bit 31 is set to 1 to indicate a VM-entry failure.
     - The remainder of the field (bits 30:16) is cleared.
   - — Exit qualification. This field is set based on the exit reason.
     - VM-entry failure due to invalid guest state. In most cases, the exit qualification is cleared to 0. The following non-zero values are used in the cases indicated:
       1. Not used.
       2. Failure was due to a problem loading the PDPTEs (see Section 26.3.1.6).

3. Failure was due to an attempt to inject a non-maskable interrupt (NMI) into a guest that is blocking events through the STI blocking bit in the interruptibility-state field. Such failures are implementation-specific (see Section 26.3.1.5).
4. Failure was due to an invalid VMCS link pointer (see Section 26.3.1.5).

VM-entry checks on guest-state fields may be performed in any order. Thus, an indication by exit qualification of one cause does not imply that there are not also other errors. Different processors may give different exit qualifications for the same VMCS.

- VM-entry failure due to MSR loading. The exit qualification is loaded to indicate which entry in the VM-entry MSR-load area caused the problem (1 for the first entry, 2 for the second, etc.).

— All other VM-exit information fields are unmodified.

2. Processor state is loaded as would be done on a VM exit (see Section 27.5). If this results in [CR4.PAE & CR0.PG & ~IA32_EFER.LMA] = 1, page-directory-pointer-table entries (PDPTEs) may be checked and loaded (see Section 27.5.4).
3. The state of blocking by NMI is what it was before VM entry.
4. MSRs are loaded as specified in the VM-exit MSR-load area (see Section 27.6).

Although this process resembles that of a VM exit, many steps taken during a VM exit do not occur for these VM-entry failures:

- Most VM-exit information fields are not updated (see step 1 above).
- The valid bit in the VM-entry interruption-information field is not cleared.
- The guest-state area is not modified.
- No MSRs are saved into the VM-exit MSR-store area.

26.8 MACHINE-CHECK EVENTS DURING VM ENTRY

If a machine-check event occurs during a VM entry, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM entry:

  — If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:[1]

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

        - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating "unrecoverable machine-check condition."
        - If the logical processor is outside SMX operation, it goes to the shutdown state.
    - If CR4.MCE = 1, a machine-check exception (#MC) is delivered through the IDT.
- The machine-check event is handled after VM entry completes:
    - If the VM entry ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
        - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
        - If the logical processor is outside SMX operation, it goes to the shutdown state.
    - If the VM entry ends with CR4.MCE = 1, a machine-check exception (#MC) is generated:
        - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
        - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- A VM-entry failure occurs as described in Section 26.7. The basic exit reason is 41, for "VM-entry failure due to machine-check event."

The first option is not used if the machine-check event occurs after any guest state has been loaded. The second option is used only if VM entry is able to load all guest state.

CHAPTER 27 VM EXITS

VM exits occur in response to certain instructions and events in VMX non-root operation as detailed in Section 25.1 through Section 25.3. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and VM-entry control fields are modified as described in Section 27.2.
2. Processor state is saved in the guest-state area (Section 27.3).
3. MSRs may be saved in the VM-exit MSR-store area (Section 27.4).
4. The following may be performed in parallel and in any order (Section 27.5):
   - Processor state is loaded based in part on the host-state area and some VM-exit controls. This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM. See Section 29.15.6 for information on how processor state is loaded by such VM exits.
   - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 27.6). This step is not performed for SMM VM exits that activate the dual-monitor treatment of SMIs and SMM.

VM exits are not logged with last-branch records, do not produce branch-trace messages, and do not update the branch-trace store.

Section 27.1 clarifies the nature of the architectural state before a VM exit begins. The steps described above are detailed in Section 27.2 through Section 27.6.

Section 29.15 describes the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM). Under this treatment, ordinary transitions to SMM are replaced by VM exits to a separate SMM monitor. Called **SMM VM exits**, these are caused by the arrival of an SMI or the execution of VMCALL in VMX root operation. SMM VM exits differ from other VM exits in ways that are detailed in Section 29.15.2.

27.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section describes the architectural state that exists before a VM exit, especially for VM exits caused by events that would normally be delivered through the IDT. Note the following:

- An exception causes a VM exit **directly** if the bit corresponding to that exception is set in the exception bitmap. A non-maskable interrupt (NMI) causes a VM exit directly if the "NMI exiting" VM-execution control is 1. An external interrupt causes a VM exit directly if the "external-interrupt exiting" VM-execution control is 1. A start-up IPI (SIPI) that arrives while a logical processor is in the wait-for-SIPI activity state causes a VM exit directly. INIT signals that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.

- An exception, NMI, external interrupt, or software interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, task switch, APIC access (see Section 25.2), EPT violation, or EPT misconfiguration that causes a VM exit.
- An event **results** in a VM exit if it causes a VM exit (directly or indirectly).

The following bullets detail when architectural state is and is not updated in response to VM exits:

- If an event causes a VM exit directly, it does not update architectural state as it would have if it had it not caused the VM exit:
  - A debug exception does not update DR6, DR7.GD, or IA32_DEBUGCTL.LBR. (Information about the nature of the debug exception is saved in the exit qualification field.)
  - A page fault does not update CR2. (The linear address causing the page fault is saved in the exit-qualification field.)
  - An NMI causes subsequent NMIs to be blocked, but only after the VM exit completes.
  - An external interrupt does not acknowledge the interrupt controller and the interrupt remains pending, unless the "acknowledge interrupt on exit" VM-exit control is 1. In such a case, the interrupt controller is acknowledged and the interrupt is no longer pending.
  - The flags L0 – L3 in DR7 (bit 0, bit 2, bit 4, and bit 6) are not cleared when a task switch causes a VM exit.
  - If a task switch causes a VM exit, none of the following are modified by the task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT[1]; or the TR register.
  - No last-exception record is made if the event that would do so directly causes a VM exit.
  - If a machine-check exception causes a VM exit directly, this does not prevent machine-check MSRs from being updated. These are updated by the machine-check event itself and not the resulting machine-check exception.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to lower 32 bits of the indicated register.

  — If the logical processor is in an inactive state (see Section 24.4.2) and not executing instructions, some events may be blocked but others may return the logical processor to the active state. Unblocked events may cause VM exits.[1] If an unblocked event causes a VM exit directly, a return to the active state occurs only after the VM exit completes.[2] The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.

    MTF VM exits (see Section 25.7.2 and Section 26.6.8) are not blocked in the HLT activity state. If an MTF VM exit occurs in the HLT activity state, the logical processor returns to the active state only after the VM exit completes. MTF VM exits are blocked the shutdown state and the wait-for-SIPI state.

- If an event causes a VM exit indirectly, the event does update architectural state:
  — A debug exception updates DR6, DR7, and the IA32_DEBUGCTL MSR. No debug exceptions are considered pending.
  — A page fault updates CR2.
  — An NMI causes subsequent NMIs to be blocked before the VM exit commences.
  — An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.
  — If the logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.
  — There is no blocking by STI or by MOV SS when the VM exit commences.
  — Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.
  — The treatment of last-exception records is implementation dependent:
    - Some processors make a last-exception record when beginning the delivery of an event through the IDT (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).
    - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-

1. If a VM exit takes the processor from an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.
2. An exception is made if the logical processor had been inactive due to execution of MWAIT; in this case, it is considered to have become active before the VM exit.

exception record if a VM exit or triple fault occurs before an event handler is reached.

- If the “virtual NMIs” VM-execution control is 1, VM entry injects an NMI, and delivery of the NMI causes a nested exception, double fault, task switch, or APIC access that causes a VM exit, virtual-NMI blocking is in effect before the VM exit commences.
- If a VM exit results from a fault, EPT violation, or EPT misconfiguration encountered during execution of IRET and the “NMI exiting” VM-execution control is 0, any blocking by NMI is cleared before the VM exit commences. However, the previous state of blocking by NMI may be recorded in the VM-exit interruption-information field; see Section 27.2.2.
- If a VM exit results from a fault, EPT violation, or EPT misconfiguration encountered during execution of IRET and the “virtual NMIs” VM-execution control is 1, virtual-NMI blocking is cleared before the VM exit commences. However, the previous state of virtual-NMI blocking may be recorded in the VM-exit interruption-information field; see Section 27.2.2.
- Suppose that a VM exit is caused directly by an x87 FPU Floating-Point Error (#MF) or by any of the following events if the event was unblocked due to (and given priority over) an x87 FPU Floating-Point Error: an INIT signal, an external interrupt, an NMI, an SMI; or a machine-check exception. In these cases, there is no blocking by STI or by MOV SS when the VM exit commences.
- Normally, a last-branch record may be made when an event is delivered through the IDT. However, if such an event results in a VM exit before delivery is complete, no last-branch record is made.
- If machine-check exception results in a VM exit, processor state is suspect and may result in suspect state being saved to the guest-state area. A VM monitor should consult the RIPV and EIPV bits in the IA32_MCG_STATUS MSR before resuming a guest that caused a VM exit resulting from a machine-check exception.
- If a VM exit results from a fault, APIC access (see Section 25.2), EPT violation, or EPT misconfiguration encountered while executing an instruction, data breakpoints due to that instruction may have been recognized and information about them may be saved in the pending debug exceptions field (see Section 27.3.4).
- The following VM exits are considered to happen after an instruction is executed:
  - VM exits resulting from debug traps (single-step, I/O breakpoints, and data breakpoints).
  - VM exits resulting from debug exceptions whose recognition was delayed by blocking by MOV SS.
  - VM exits resulting from some machine-check exceptions.
  - Trap-like VM exits due to execution of MOV to CR8 when the “CR8-load exiting” VM-execution control is 0 and the “use TPR shadow” VM-execution

control is 1. (Such VM exits can occur only from 64-bit mode and thus only on processors that support Intel 64 architecture.)

- Trap-like VM exits due to execution of WRMSR when the “use MSR bitmaps” VM-execution control is 1, the value of ECX is 808H, bit 808H in write bitmap for low MSRs is 0, and the “virtualize x2APIC mode” VM-execution control is 1. See Section 25.1.3.
- VM exits caused by TPR-shadow updates (see Section 25.5.3.3) that result from APIC accesses as part of instruction execution.

For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to the logical processor’s interruptibility state (see Table 24-3). If there had been blocking by MOV SS, POP SS, or STI before the instruction executed, such blocking is no longer in effect.

27.2 RECORDING VM-EXIT INFORMATION AND UPDATING VM-ENTRY CONTROL FIELDS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 27.2.1 to Section 27.2.4 detail the use of these fields.

In addition to updating the VM-exit information fields, the valid bit (bit 31) is cleared in the VM-entry interruption-information field. If bit 5 of the IA32_VMX_MISC MSR (index 485H) is read as 1 (see Appendix A.6), the value of IA32_EFER.LMA is stored into the “IA-32e mode guest” VM-entry control.[1]

27.2.1 Basic VM-Exit Information

Section 24.9.1 defines the basic VM-exit information fields. The following items detail their use.

- **Exit reason.**
  - Bits 15:0 of this field contain the basic exit reason. It is loaded with a number indicating the general cause of the VM exit. Appendix C lists the numbers used and their meaning.
  - The remainder of the field (bits 31:16) is cleared to 0 (certain SMM VM exits may set some of these bits; see Section 29.15.2.3).[2]

1. Bit 5 of the IA32_VMX_MISC MSR is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.
2. Bit 13 of this field is set on certain VM-entry failures; see Section 26.7.

- **Exit qualification.** This field is saved for VM exits due to the following causes: debug exceptions; page-fault exceptions; start-up IPIs (SIPIs); system-management interrupts (SMIs) that arrive immediately after the retirement of I/O instructions; task switches; INVEPT; INVLPG; INVPCID; INVVPID; LGDT; LIDT; LLDT; LTR; SGDT; SIDT; SLDT; STR; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; MWAIT; accesses to the APIC-access page (see Section 25.2); and EPT violations. For all other VM exits, this field is cleared. The following items provide details:
  - — For a debug exception, the exit qualification contains information about the debug exception. The information has the format given in Table 27-1.

Table 27-1. Exit Qualification for Debug Exceptions

Bit Position(s)	Contents
3:0	B3 – B0. When set, each of these bits indicates that the corresponding breakpoint condition was met. Any of these bits may be set even if its corresponding enabling bit in DR7 is not set.
12:4	Reserved (cleared to 0).
13	BD. When set, this bit indicates that the cause of the debug exception is "debug register access detected."
14	BS. When set, this bit indicates that the cause of the debug exception is either the execution of a single instruction (if RFLAGS.TF = 1 and IA32_DEBUGCTL.BTF = 0) or a taken branch (if RFLAGS.TF = DEBUGCTL.BTF = 1).
63:15	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

  - — For a page-fault exception, the exit qualification contains the linear address that caused the page fault. On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - — For a start-up IPI (SIPI), the exit qualification contains the SIPI vector information in bits 7:0. Bits 63:8 of the exit qualification are cleared to 0.

— For a task switch, the exit qualification contains details about the task switch, encoded as shown in Table 27-2.

Table 27-2. Exit Qualification for Task Switch

Bit Position(s)	Contents
15:0	Selector of task-state segment (TSS) to which the guest attempted to switch
29:16	Reserved (cleared to 0)
31:30	Source of task switch initiation: 0: CALL instruction 1: IRET instruction 2: JMP instruction 3: Task gate in IDT
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

— For INVLPG, the exit qualification contains the linear-address operand of the instruction.

- On processors that support Intel 64 architecture, bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
- If the INVLPG source operand specifies an unusable segment, the linear address specified in the exit qualification will match the linear address that the INVLPG would have used if no VM exit occurred. This address is not architecturally defined and may be implementation-specific.

— For INVEPT, INVPCID, INVVPID, LGDT, LIDT, LLDT, LTR, SGDT, SIDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, and VMXON, the exit qualification receives the value of the instruction's displacement field, which is sign-extended to 64 bits if necessary (32 bits on processors that do not support Intel 64 architecture). If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

On processors that support Intel 64 architecture, an exception is made for RIP-relative addressing (used only in 64-bit mode). Such addressing causes an instruction to use an address that is the sum of the displacement field and the value of RIP that references the following instruction. In this case, the exit qualification is loaded with the sum of the displacement field and the appropriate RIP value.

In all cases, bits of this field beyond the instruction's address size are undefined. For example, suppose that the address-size field in the VM-exit instruction-information field (see Section 24.9.4 and Section 27.2.4) reports an *n*-bit address size. Then bits 63:*n* (bits 31:*n* on processors that do not support Intel 64 architecture) of the instruction displacement are undefined.

— For a control-register access, the exit qualification contains information about the access and has the format given in Table 27-3.

Table 27-3. Exit Qualification for Control-Register Accesses

Bit Positions	Contents
3:0	Number of control register (0 for CLTS and LMSW). Bit 3 is always 0 on processors that do not support Intel 64 architecture as they do not support CR8.
5:4	Access type: 0 = MOV to CR 1 = MOV from CR 2 = CLTS 3 = LMSW
6	LMSW operand type: 0 = register 1 = memory For CLTS and MOV CR, cleared to 0
7	Reserved (cleared to 0)
11:8	For MOV CR, the general-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) For CLTS and LMSW, cleared to 0
15:12	Reserved (cleared to 0)
31:16	For LMSW, the LMSW source data For CLTS and MOV CR, cleared to 0
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

— For MOV DR, the exit qualification contains information about the instruction and has the format given in Table 27-4.

Table 27-4. Exit Qualification for MOV DR

Bit Position(s)	Contents
2:0	Number of debug register
3	Reserved (cleared to 0)
4	Direction of access (0 = MOV to DR; 1 = MOV from DR)
7:5	Reserved (cleared to 0)
11:8	General-purpose register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8 –15 = R8 – R15, respectively
63:12	Reserved (cleared to 0)

— For an I/O instruction, the exit qualification contains information about the instruction and has the format given in Table 27-5.

Table 27-5. Exit Qualification for I/O Instructions

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)

Table 27-5. Exit Qualification for I/O Instructions (Contd.)

Bit Position(s)	Contents
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in DX or in an immediate operand)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

— For MWAIT, the exit qualification contains a value that indicates whether address-range monitoring hardware was armed. The exit qualification is set either to 0 (if address-range monitoring hardware is not armed) or to 1 (if address-range monitoring hardware is armed).

— For an APIC-access VM exit resulting from a linear access or a guest-physical access to the APIC-access page (see Section 25.2.1 and Section 25.2.2), the exit qualification contains information about the access and has the format given in Table 27-6.[1]

Table 27-6. Exit Qualification for APIC-Access VM Exits from Linear Accesses and Guest-Physical Accesses

Bit Position(s)	Contents
11:0	▪ If the APIC-access VM exit is due to a linear access, the offset of access within the APIC page. ▪ Undefined if the APIC-access VM exit is due a guest-physical access
15:12	Access type: 0 = linear access for a data read during instruction execution 1 = linear access for a data write during instruction execution 2 = linear access for an instruction fetch 3 = linear access (read or write) during event delivery 10 = guest-physical access during event delivery 15 = guest-physical access for an instruction fetch or during instruction execution Other values not used
63:16	Reserved (cleared to 0). Bits 63:32 exist only on processors that support Intel 64 architecture.

Such a VM exit that set bits 15:12 of the exit qualification to 0000b (data read during instruction execution) or 0001b (data write during instruction execution) set bit 12—which distinguishes data read from data write—to that which would have been stored in bit 1—W/R—of the page-fault error code had the access caused a page fault instead of an APIC-access VM exit. This implies the following:

- For an APIC-access VM exit caused by the CLFLUSH instruction, the access type is "data read during instruction execution."
- For an APIC-access VM exit caused by the ENTER instruction, the access type is "data write during instruction execution."
- For an APIC-access VM exit caused by the MASKMOVQ instruction or the MASKMOVDQU instruction, the access type is "data write during instruction execution."
- For an APIC-access VM exit caused by the MONITOR instruction, the access type is "data read during instruction execution."

Such a VM exit stores 1 for bit 31 for IDT-vectoring information field (see Section 27.2.3) if and only if it sets bits 15:12 of the exit qualification to 0011b (linear access during event delivery) or 1010b (guest-physical access during event delivery).

See Section 25.2.1.3 for further discussion of these instructions and APIC-access VM exits.

For APIC-access VM exits resulting from physical accesses, the APIC-access page (see Section 25.2.3), the exit qualification is undefined.

— For an EPT violation, the exit qualification contains information about the access causing the EPT violation and has the format given in Table 27-5.

Table 27-7. Exit Qualification for EPT Violations

Bit Position(s)	Contents
0	Set if the access causing the EPT violation was a data read.
1	Set if the access causing the EPT violation was a data write.
2	Set if the access causing the EPT violation was an instruction fetch.

1. The exit qualification is undefined if the access was part of the logging of a branch record or a precise-event-based-sampling (PEBS) record to the DS save area. It is recommended that software configure the paging structures so that no address in the DS save area translates to an address on the APIC-access page.

Table 27-7. Exit Qualification for EPT Violations (Contd.)

Bit Position(s)	Contents
3	The logical-AND of bit 0 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation (indicates that the guest-physical address was readable).[1]
4	The logical-AND of bit 1 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation (indicates that the guest-physical address was writeable).
5	The logical-AND of bit 2 in the EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation (indicates that the guest-physical address was executable).
6	Reserved (cleared to 0).
7	Set if the guest linear-address field is valid. The guest linear-address field is valid for all EPT violations except those resulting from an attempt to load the guest PDPTEs as part of the execution of the MOV CR instruction.
8	If bit 7 is 1: ▪ Set if the access causing the EPT violation is to a guest-physical address that is the translation of a linear address. ▪ Clear if the access causing the EPT violation is to a paging-structure entry as part of a page walk or the update of an accessed or dirty bit. Reserved if bit 7 is 0 (cleared to 0).
11:9	Reserved (cleared to 0).
12	NMI unblocking due to IRET
63:13	Reserved (cleared to 0).

NOTES:

1. Bits 5:3 are cleared to 0 if any of EPT paging-structures entries used to translate the guest-physical address of the access causing the EPT violation is not present (see Section 28.2.2).

An EPT violation that occurs during as a result of execution of a read-modify-write operation sets bit 1 (data write). Whether it also sets bit 0 (data read) is implementation-specific and, for a given implementation, may differ for different kinds of read-modify-write operations.

Bit 12 is undefined in any of the following cases:

- If the "NMI exiting" VM-execution control is 1 and the "virtual NMIs" VM-execution control is 0.

    - If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 27.2.3).

  Otherwise, bit 12 is defined as follows:

    - If the "virtual NMIs" VM-execution control is 0, the EPT violation was caused by a memory access as part of execution of the IRET instruction, and blocking by NMI (see Table 24-3) was in effect before execution of IRET, bit 12 is set to 1.
    - If the "virtual NMIs" VM-execution control is 1,the EPT violation was caused by a memory access as part of execution of the IRET instruction, and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.
    - For all other relevant VM exits, bit 12 is cleared to 0.

- **Guest-linear address.** For some VM exits, this field receives a linear address that pertains to the VM exit. The field is set for different VM exits as follows:
  - VM exits due to attempts to execute LMSW with a memory operand. In these cases, this field receives the linear address of that operand. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - VM exits due to attempts to execute INS or OUTS for which the relevant segment is usable (if the relevant segment is not usable, the value is undefined). (ES is always the relevant segment for INS; for OUTS, the relevant segment is DS unless overridden by an instruction prefix.) The linear address is the base address of relevant segment plus (E)DI (for INS) or (E)SI (for OUTS). Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - VM exits due to EPT violations that set bit 7 of the exit qualification (see Table 27-7; these are all EPT violations except those resulting from an attempt to load the PDPTEs as of execution of the MOV CR instruction). The linear address may translate to the guest-physical address whose access caused the EPT violation. Alternatively, translation of the linear address may reference a paging-structure entry whose access caused the EPT violation. Bits 63:32 are cleared if the logical processor was not in 64-bit mode before the VM exit.
  - For all other VM exits, the field is undefined.
- **Guest-physical address.** For a VM exit due to an EPT violation or an EPT misconfiguration, this field receives the guest-physical address that caused the EPT violation or EPT misconfiguration. For all other VM exits, the field is undefined.

27.2.2 Information for VM Exits Due to Vectored Events

Section 24.9.2 defines fields containing information for VM exits due to the following events: exceptions (including those generated by the instructions INT3, INTO, BOUND, and UD2); external interrupts that occur while the "acknowledge interrupt on exit" VM-exit control is 1; and non-maskable interrupts (NMIs). Such VM exits include those that occur on an attempt at a task switch that causes an exception before generating the VM exit due to the task switch that causes the VM exit.

The following items detail the use of these fields:

- **VM-exit interruption information** (format given in Table 24-15). The following items detail how this field is established for VM exits due to these events:
  - For an exception, bits 7:0 receive the exception vector (at most 31). For an NMI, bits 7:0 are set to 2. For an external interrupt, bits 7:0 receive the interrupt number.
  - Bits 10:8 are set to 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), or 6 (software exception). Hardware exceptions comprise all exceptions except breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD2 are hardware exceptions.
  - Bit 11 is set to 1 if the VM exit is caused by a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).[1] If bit 11 is set to 1, the error code is placed in the VM-exit interruption error code (see below).
  - Bit 12 is undefined in any of the following cases:
    - If the "NMI exiting" VM-execution control is 1 and the "virtual NMIs" VM-execution control is 0.
    - If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 27.2.3).
    - If the VM exit is due to a double fault (the interruption type is hardware exception and the vector is 8).

    Otherwise, bit 12 is defined as follows:

    - If the "virtual NMIs" VM-execution control is 0, the VM exit is due to a fault on the IRET instruction (other than a debug exception for an instruction breakpoint), and blocking by NMI (see Table 24-3) was in effect before execution of IRET, bit 12 is set to 1.
    - If the "virtual NMIs" VM-execution control is 1, the VM exit is due to a fault on the IRET instruction (other than a debug exception for an

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

instruction breakpoint), and virtual-NMI blocking was in effect before execution of IRET, bit 12 is set to 1.

- For all other relevant VM exits, bit 12 is cleared to 0.[1]

— Bits 30:13 are always set to 0.

— Bit 31 is always set to 1.

For other VM exits (including those due to external interrupts when the “acknowledge interrupt on exit” VM-exit control is 0), the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- VM-exit interruption error code.

  — For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the VM-exit interruption-information field, this field receives the error code that would have been pushed on the stack had the event causing the VM exit been delivered normally through the IDT. The EXT bit is set in this field exactly when it would be set normally. For exceptions that occur during the delivery of double fault (if the IDT-vectoring information field indicates a double fault), the EXT bit is set to 1, assuming that (1) that the exception would produce an error code normally (if not incident to double-fault delivery) and (2) that the error code uses the EXT bit (not for page faults, which use a different format).

  — For other VM exits, the value of this field is undefined.

27.2.3 Information for VM Exits During Event Delivery

Section 24.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT and as a result of any of the following cases:[2]

- A fault occurs during event delivery and causes a VM exit (because the bit associated with the fault is set to 1 in the exception bitmap).
- A task switch is invoked through a task gate in the IDT. The VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 25.6.2).
- Event delivery causes an APIC-access VM exit (see Section 25.2).
- An EPT violation or EPT misconfiguration that occurs during event delivery.

These fields are used for VM exits that occur during delivery of events injected as part of VM entry (see Section 26.5.1.2).

1. The conditions imply that, if the “NMI exiting” VM-execution control is 0 or the “virtual NMIs” VM-execution control is 1, bit 12 is always cleared to 0 by VM exits due to debug exceptions.

2. This includes the case in which a VM exit occurs while delivering a software interrupt (INT *n*) through the 16-bit IVT (interrupt vector table) that is used in virtual-8086 mode with virtual-machine extensions (if RFLAGS.VM = CR4.VME = 1).

A VM exit is not considered to occur during event delivery in any of the following circumstances:

- The original event causes the VM exit directly (for example, because the original event is a non-maskable interrupt (NMI) and the "NMI exiting" VM-execution control is 1).
- The original event results in a double-fault exception that causes the VM exit directly.
- The VM exit occurred as a result of fetching the first instruction of the handler invoked by the event delivery.
- The VM exit is caused by a triple fault.

The following items detail the use of these fields:

- IDT-vectoring information (format given in Table 24-16). The following items detail how this field is established for VM exits that occur during event delivery:
  - If the VM exit occurred during delivery of an exception, bits 7:0 receive the exception vector (at most 31). If the VM exit occurred during delivery of an NMI, bits 7:0 are set to 2. If the VM exit occurred during delivery of an external interrupt, bits 7:0 receive the interrupt number.
  - Bits 10:8 are set to indicate the type of event that was being delivered when the VM exit occurred: 0 (external interrupt), 2 (non-maskable interrupt), 3 (hardware exception), 4 (software interrupt), 5 (privileged software interrupt), or 6 (software exception).

    Hardware exceptions comprise all exceptions except breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO); these are software exceptions. BOUND-range exceeded exceptions (#BR; generated by BOUND) and invalid opcode exceptions (#UD) generated by UD2 are hardware exceptions.

    Bits 10:8 may indicate privileged software interrupt if such an event was injected as part of VM entry.
  - Bit 11 is set to 1 if the VM exit occurred during delivery of a hardware exception that would have delivered an error code on the stack. This bit is always 0 if the VM exit occurred while the logical processor was in real-address mode (CR0.PE=0).[1] If bit 11 is set to 1, the error code is placed in the IDT-vectoring error code (see below).
  - Bit 12 is undefined.
  - Bits 30:13 are always set to 0.
  - Bit 31 is always set to 1.

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PE must be 1 in VMX operation, a logical processor cannot be in real-address mode unless the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

For other VM exits, the field is marked invalid (by clearing bit 31) and the remainder of the field is undefined.

- IDT-vectoring error code.
  - For VM exits that set both bit 31 (valid) and bit 11 (error code valid) in the IDT-vectoring information field, this field receives the error code that would have been pushed on the stack by the event that was being delivered through the IDT at the time of the VM exit. The EXT bit is set in this field when it would be set normally.
  - For other VM exits, the value of this field is undefined.

27.2.4 Information for VM Exits Due to Instruction Execution

Section 24.9.4 defined fields containing information for VM exits that occur due to instruction execution. (The VM-exit instruction length is also used for VM exits that occur during the delivery of a software interrupt or software exception.) The following items detail their use.

- **VM-exit instruction length.** This field is used in the following cases:
  - For fault-like VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 25.1.2) or based on the settings of VM-execution controls (see Section 25.1.3): CLTS, CPUID, GETSEC, HLT, IN, INS, INVD, INVEPT, INVLPG, INVPCID, INVVPID, LGDT, LIDT, LLDT, LMSW, LTR, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPMC, RDRAND, RDTSC, RDTSCP, RSM, SGDT, SIDT, SLDT, STR, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, WBINVD, WRMSR, and XSETBV.[1]
  - For VM exits due to software exceptions (those generated by executions of INT3 or INTO).
  - For VM exits due to faults encountered during delivery of a software interrupt, privileged software exception, or software exception.
  - For VM exits due to attempts to effect a task switch via instruction execution. These are VM exits that produce an exit reason indicating task switch and either of the following:
    - An exit qualification indicating execution of CALL, IRET, or JMP instruction.
    - An exit qualification indicating a task gate in the IDT and an IDT-vectoring information field indicating that the task gate was encountered during

1. This item applies only to fault-like VM exits. It does not apply to trap-like VM exits following executions of the MOV to CR8 instruction when the “use TPR shadow” VM-execution control is 1 or to those following executions of the WRMSR instruction when the “virtualize x2APIC mode” VM-execution control is 1.

delivery of a software interrupt, privileged software exception, or software exception.

— For APIC-access VM exits resulting from linear accesses (see Section 25.2.1) and encountered during delivery of a software interrupt, privileged software exception, or software exception.[1]

— For VM exits resulting from executions of VMFUNC with EAX = 0 that fail either because ECX ≥ 512 or because the value of ECX selected an invalid tentative EPTP value.

In all the above cases, this field receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit (see the next paragraph for one exception).

The cases of VM exits encountered during delivery of a software interrupt, privileged software exception, or software exception include those encountered during delivery of events injected as part of VM entry (see Section 26.5.1.2). If the original event was injected as part of VM entry, this field receives the value of the VM-entry instruction length.

All VM exits other than those listed in the above items leave this field undefined.

- **VM-exit instruction information**. For VM exits due to attempts to execute INS, INVEPT, INVPCID, INVVPID, LIDT, LGDT, LLDT, LTR, OUTS, RDRAND, SIDT, SGDT, SLDT, STR, VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON, this field receives information about the instruction that caused the VM exit. The format of the field depends on the identity of the instruction causing the VM exit:

  — For VM exits due to attempts to execute INS or OUTS, the field has the format is given in Table 27-8.[2]

Table 27-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS

Bit Position(s)	Content
6:0	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.

1. The VM-exit instruction-length field is not defined following APIC-access VM exits resulting from physical accesses (see Section 25.2.3) even if encountered during delivery of a software interrupt, privileged software exception, or software exception.

2. The format of the field was undefined for these VM exits on the first processors to support the virtual-machine extensions. Software can determine whether the format specified in Table 27-8 is used by consulting the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

Table 27-8. Format of the VM-Exit Instruction-Information Field as Used for INS and OUTS (Contd.)

Bit Position(s)	Content
14:10	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for VM exits due to execution of INS.
31:18	Undefined.

— For VM exits due to attempts to execute INVEPT, INVPCID, or INVVPID, the field has the format is given in Table 27-9.

Table 27-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
14:11	Undefined.

Table 27-9. Format of the VM-Exit Instruction-Information Field as Used for INVEPT, INVPCID, and INVVPID (Contd.)

Bit Position(s)	Content
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for memory instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Reg2 (same encoding as IndexReg above)

— For VM exits due to attempts to execute LIDT, LGDT, SIDT, or SGDT, the field has the format is given in Table 27-10.

Table 27-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).

Table 27-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.
11	Operand size: 0: 16-bit 1: 32-bit Undefined for VM exits from 64-bit mode.
14:12	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)

Table 27-10. Format of the VM-Exit Instruction-Information Field as Used for LIDT, LGDT, SIDT, or SGDT (Contd.)

Bit Position(s)	Content
29:28	Instruction identity: 0: SGDT 1: SIDT 2: LGDT 3: LIDT
31:30	Undefined.

— For VM exits due to attempts to execute LLDT, LTR, SLDT, or STR, the field has the format is given in Table 27-11.

Table 27-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).

Table 27-11. Format of the VM-Exit Instruction-Information Field as Used for LLDT, LTR, SLDT, and STR (Contd.)

Bit Position(s)	Content
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
29:28	Instruction identity: 0: SLDT 1: STR 2: LLDT 3: LTR
31:30	Undefined.

— For VM exits due to attempts to execute RDRAND, the field has the format is given in Table 27-12.

Table 27-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND

Bit Position(s)	Content
2:0	Undefined.

Table 27-12. Format of the VM-Exit Instruction-Information Field as Used for RDRAND (Contd.)

Bit Position(s)	Content
6:3	Destination register: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture)
10:7	Undefined.
12:11	Operand size: 0: 16-bit 1: 32-bit 2: 64-bit The value 3 is not used.
31:13	Undefined.

— For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, or VMXON, the field has the format is given in Table 27-13.

Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, and VMXON

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
6:2	Undefined.
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used.
10	Cleared to 0.

Table 27-13. Format of the VM-Exit Instruction-Information Field as Used for VMCLEAR, VMPTRLD, VMPTRST, and VMXON (Contd.)

Bit Position(s)	Content
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used.
21:18	IndexReg: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for instructions with no index register (bit 22 is set).
22	IndexReg invalid (0 = valid; 1 = invalid)
26:23	BaseReg (encoded as IndexReg above) Undefined for instructions with no base register (bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid)
31:28	Undefined.

— For VM exits due to attempts to execute VMREAD or VMWRITE, the field has the format is given in Table 27-14.

Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE

Bit Position(s)	Content
1:0	Scaling: 0: no scaling 1: scale by 2 2: scale by 4 3: scale by 8 (used only on processors that support Intel 64 architecture) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).
2	Undefined.
6:3	Reg1: 0 = RAX 1 = RCX 2 = RDX 3 = RBX 4 = RSP 5 = RBP 6 = RSI 7 = RDI 8–15 represent R8–R15, respectively (used only on processors that support Intel 64 architecture) Undefined for memory instructions (bit 10 is clear).
9:7	Address size: 0: 16-bit 1: 32-bit 2: 64-bit (used only on processors that support Intel 64 architecture) Other values not used. Undefined for register instructions (bit 10 is set).
10	Mem/Reg (0 = memory; 1 = register).
14:11	Undefined.
17:15	Segment register: 0: ES 1: CS 2: SS 3: DS 4: FS 5: GS Other values not used. Undefined for register instructions (bit 10 is set).
21:18	IndexReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no index register (bit 10 is clear and bit 22 is set).

Table 27-14. Format of the VM-Exit Instruction-Information Field as Used for VMREAD and VMWRITE (Contd.)

Bit Position(s)	Content
22	IndexReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
26:23	BaseReg (encoded as Reg1 above) Undefined for register instructions (bit 10 is set) and for memory instructions with no base register (bit 10 is clear and bit 27 is set).
27	BaseReg invalid (0 = valid; 1 = invalid) Undefined for register instructions (bit 10 is set).
31:28	Reg2 (same encoding as Reg1 above)

For all other VM exits, the field is undefined.

- **I/O RCX, I/O RSI, I/O RDI, I/O RIP**. These fields are undefined except for SMM VM exits due to system-management interrupts (SMIs) that arrive immediately after retirement of I/O instructions. See Section 29.15.2.3.

27.3 SAVING GUEST STATE

Each field in the guest-state area of the VMCS (see Section 24.4) is written with the corresponding component of processor state. On processors that support Intel 64 architecture, the full values of each natural-width field (see Section 24.10.2) is saved regardless of the mode of the logical processor before and after the VM exit.

In general, the state saved is that which was in the logical processor at the time the VM exit commences. See Section 27.1 for a discussion of which architectural updates occur at that time.

Section 27.3.1 through Section 27.3.4 provide details for how certain components of processor state are saved. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

27.3.1 Saving Control Registers, Debug Registers, and MSRs

Contents of certain control registers, debug registers, and MSRs is saved as follows:

- The contents of CR0, CR3, CR4, and the IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are saved into the corresponding fields. Bits 63:32 of the IA32_SYSENTER_CS MSR are not saved. On processors that do not support Intel 64 architecture, bits 63:32 of the IA32_SYSENTER_ESP and IA32_SYSENTER_EIP MSRs are not saved.

- If the “save debug controls” VM-exit control is 1, the contents of DR7 and the IA32_DEBUGCTL MSR are saved into the corresponding fields. The first processors to support the virtual-machine extensions supported only the 1-setting of this control and thus always saved data into these fields.
- If the “save IA32_PAT” VM-exit control is 1, the contents of the IA32_PAT MSR are saved into the corresponding field.
- If the “save IA32_EFER” VM-exit control is 1, the contents of the IA32_EFER MSR are saved into the corresponding field.
- The value of the SMBASE field is undefined after all VM exits except SMM VM exits. See Section 29.15.2.

27.3.2 Saving Segment Registers and Descriptor-Table Registers

For each segment register (CS, SS, DS, ES, FS, GS, LDTR, or TR), the values saved for the base-address, segment-limit, and access rights are based on whether the register was unusable (see Section 24.4.1) before the VM exit:

- If the register was unusable, the values saved into the following fields are undefined: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in the access-rights field. The following exceptions apply:
  - CS.
    - The base-address and segment-limit fields are saved.
    - The L, D, and G bits are saved in the access-rights field.
  - SS.
    - DPL is saved in the access-rights field.
    - On processors that support Intel 64 architecture, bits 63:32 of the value saved for the base address are always zero.
  - DS and ES. On processors that support Intel 64 architecture, bits 63:32 of the values saved for the base addresses are always zero.
  - FS and GS. The base-address field is saved.
  - LDTR. The value saved for the base address is always canonical.
- If the register was not unusable, the values saved into the following fields are those which were in the register before the VM exit: (1) base address; (2) segment limit; and (3) bits 7:0 and bits 15:12 in access rights.
- Bits 31:17 and 11:8 in the access-rights field are always cleared. Bit 16 is set to 1 if and only if the segment is unusable.

The contents of the GDTR and IDTR registers are saved into the corresponding base-address and limit fields.

27.3.3 Saving RIP, RSP, and RFLAGS

The contents of the RIP, RSP, and RFLAGS registers are saved as follows:

- The value saved in the RIP field is determined by the nature and cause of the VM exit:
  - If the VM exit occurs due to by an attempt to execute an instruction that causes VM exits unconditionally or that has been configured to cause a VM exit via the VM-execution controls, the value saved references that instruction.
  - If the VM exit is caused by an occurrence of an INIT signal, a start-up IPI (SIPI), or system-management interrupt (SMI), the value saved is that which was in RIP before the event occurred.
  - If the VM exit occurs due to the 1-setting of either the “interrupt-window exiting” VM-execution control or the “NMI-window exiting” VM-execution control, the value saved is that which would be in the register had the VM exit not occurred.
  - If the VM exit is due to an external interrupt, non-maskable interrupt (NMI), or hardware exception (as defined in Section 27.2.2), the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate,[1] or into the old task-state segment had the event been delivered through a task gate).
  - If the VM exit is due to a triple fault, the value saved is the return pointer that would have been saved (either on the stack had the event been delivered through a trap or interrupt gate, or into the old task-state segment had the event been delivered through a task gate) had delivery of the double fault not encountered the nested exception that caused the triple fault.
  - If the VM exit is due to a software exception (due to an execution of INT3 or INTO), the value saved references the INT3 or INTO instruction that caused that exception.
  - Suppose that the VM exit is due to a task switch that was caused by execution of CALL, IRET, or JMP or by execution of a software interrupt (INT *n*) or software exception (due to execution of INT3 or INTO) that encountered a task gate in the IDT. The value saved references the instruction that caused the task switch (CALL, IRET, JMP, INT *n*, INT3, or INTO).
  - Suppose that the VM exit is due to a task switch that was caused by a task gate in the IDT that was encountered for any reason except the direct access by a software interrupt or software exception. The value saved is that which would have been saved in the old task-state segment had the task switch completed normally.

1. The reference here is to the full value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

 - — If the VM exit is due to an execution of MOV to CR8 or WRMSR that reduced the value of the TPR shadow[1] below that of TPR threshold VM-execution control field, the value saved references the instruction following the MOV to CR8 or WRMSR.
 - — If the VM exit was caused by a TPR-shadow update (see Section 21.5.3.3) that results from an APIC access as part of instruction execution, the value saved references the instruction following the one whose execution caused the VTPR access.

- The contents of the RSP register are saved into the RSP field.
- With the exception of the resume flag (RF; bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. RFLAGS.RF is saved as follows:
 - — If the VM exit is caused directly by an event that would normally be delivered through the IDT, the value saved is that which would appear in the saved RFLAGS image (either that which would be saved on the stack had the event been delivered through a trap or interrupt gate[2] or into the old task-state segment had the event been delivered through a task gate) had the event been delivered through the IDT. See below for VM exits due to task switches caused by task gates in the IDT.
 - — If the VM exit is caused by a triple fault, the value saved is that which the logical processor would have in RF in the RFLAGS register had the triple fault taken the logical processor to the shutdown state.
 - — If the VM exit is caused by a task switch (including one caused by a task gate in the IDT), the value saved is that which would have been saved in the RFLAGS image in the old task-state segment (TSS) had the task switch completed normally without exception.
 - — If the VM exit is caused by an attempt to execute an instruction that unconditionally causes VM exits or one that was configured to do with a VM-execution control, the value saved is 0.[3]
 - — For APIC-access VM exits and for VM exits caused by EPT violations and EPT misconfigurations, the value saved depends on whether the VM exit occurred during delivery of an event through the IDT:

1. The TPR shadow is bits 7:4 of the byte at offset 80H of the virtual-APIC page (see Section 24.6.8).
2. The reference here is to the full value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.
3. This is true even if RFLAGS.RF was 1 before the instruction was executed. If, in response to such a VM exit, a VM monitor re-enters the guest to re-execute the instruction that caused the VM exit (for example, after clearing the VM-execution control that caused the VM exit), the instruction may encounter a code breakpoint that has already been processed. A VM monitor can avoid this by setting the guest value of RFLAGS.RF to 1 before resuming guest software.

    - If the VM exit stored 0 for bit 31 for IDT-vectoring information field (because the VM exit did not occur during delivery of an event through the IDT; see Section 27.2.3), the value saved is 1.
    - If the VM exit stored 1 for bit 31 for IDT-vectoring information field (because the VM exit did occur during delivery of an event through the IDT), the value saved is the value that would have appeared in the saved RFLAGS image had the event been delivered through the IDT (see above).
  - — For all other VM exits, the value saved is the value RFLAGS.RF had before the VM exit occurred.

27.3.4 Saving Non-Register State

Information corresponding to guest non-register state is saved as follows:

- The activity-state field is saved with the logical processor's activity state before the VM exit.[1] See Section 27.1 for details of how events leading to a VM exit may affect the activity state.
- The interruptibility-state field is saved to reflect the logical processor's interruptibility before the VM exit. See Section 27.1 for details of how events leading to a VM exit may affect this state. VM exits that end outside system-management mode (SMM) save bit 2 (blocking by SMI) as 0 regardless of the state of such blocking before the VM exit.

  Bit 3 (blocking by NMI) is treated specially if the "virtual NMIs" VM-execution control is 1. In this case, the value saved for this field does not indicate the blocking of NMIs but rather the state of virtual-NMI blocking.
- The pending debug exceptions field is saved as clear for all VM exits except the following:
  - — A VM exit caused by an INIT signal, a machine-check exception, or a system-management interrupt (SMI).
  - — A VM exit with basic exit reason either "TPR below threshold."[2]
  - — A VM exit with basic exit reason "monitor trap flag."
  - — VM exits that are not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

  For VM exits that do not clear the field, the value saved is determined as follows:
  - — Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.

---

1. If this activity state was an inactive state resulting from execution of a specific instruction (HLT or MWAIT), the value saved for RIP by that VM exit will reference the following instruction.
2. This item includes VM exits that occur after executions of MOV to CR8 or WRMSR (Section 25.1.3), TPR-shadow updates (Section 25.5.3.3), and certain VM entries (Section 26.6.7).

- — Suppose that a VM exit is due to an INIT signal, a machine-check exception, or an SMI; or that a VM exit has basic exit reason "TPR below threshold" or "monitor trap flag." In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit.

  If the VM exit occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 26.6.3). Otherwise, the following items apply:

  - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 26.6.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
  - Bit 14 (BS) is set if RFLAGS.TF = 1 in either of the following cases:
    - IA32_DEBUGCTL.BTF = 0 and the cause of a pending debug exception was the execution of a single instruction.
    - IA32_DEBUGCTL.BTF = 1 and the cause of a pending debug exception was a taken branch.
- — Suppose that a VM exit is due to another reason (but not a debug exception) and occurs while there is MOV-SS blocking of debug exceptions. In this case, the value saved sets bits corresponding to the causes of any debug exceptions that were pending at the time of the VM exit. If the VM exit occurs immediately after VM entry (no instructions were executed in VMX non-root operation), the value saved may match that which was loaded on VM entry (see Section 26.6.3). Otherwise, the following items apply:
  - Bit 12 (enabled breakpoint) is set to 1 if there was at least one matched data or I/O breakpoint that was enabled in DR7. Bit 12 is also set if it had been set on VM entry, causing there to be valid pending debug exceptions (see Section 26.6.3) and the VM exit occurred before those exceptions were either delivered or lost. In other cases, bit 12 is cleared to 0.
  - The setting of bit 14 (BS) is implementation-specific. However, it is not set if RFLAGS.TF = 0 or IA32_DEBUGCTL.BTF = 1.
- — The reserved bits in the field are cleared.

- If the "save VMX-preemption timer value" VM-exit control is 1, the value of timer is saved into the VMX-preemption timer-value field. This is the value loaded from this field on VM entry as subsequently decremented (see Section 25.7.1). VM exits due to timer expiration save the value 0. Other VM exits may also save the value 0 if the timer expired during VM exit. (If the "save VMX-preemption timer value" VM-exit control is 0, VM exit does not modify the value of the VMX-preemption timer-value field.)
- If the logical processor supports the 1-setting of the "enable EPT" VM-execution control, values are saved into the four (4) PDPTE fields as follows:

- If the "enable EPT" VM-execution control is 1 and the logical processor was using PAE paging at the time of the VM exit, the PDPTE values currently in use are saved:[1]
  - The values saved into bits 11:9 of each of the fields is undefined.
  - If the value saved into one of the fields has bit 0 (present) clear, the value saved into bits 63:1 of that field is undefined. That value need not correspond to the value that was loaded by VM entry or to any value that might have been loaded in VMX non-root operation.
  - If the value saved into one of the fields has bit 0 (present) set, the value saved into bits 63:12 of the field is a guest-physical address.
- If the "enable EPT" VM-execution control is 0 or the logical processor was not using PAE paging at the time of the VM exit, the values saved are undefined.

27.4 SAVING MSRS

After processor state is saved to the guest-state area, values of MSRs may be stored into the VM-exit MSR-store area (see Section 24.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-store count) is processed in order by storing the value of the MSR indexed by bits 31:0 (as they would be read by RDMSR) into bits 127:64. Processing of an entry fails in either of the following cases:

- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be read only in system-management mode (SMM) and the VM exit will not end in SMM.
- The value of bits 31:0 indicates an MSR that cannot be saved on VM exits for model-specific reasons. A processor may prevent certain MSRs (based on the value of bits 31:0) from being stored on VM exits, even if they can normally be read by RDMSR. Such model-specific behavior is documented in Chapter 34.
- Bits 63:32 of the entry are not all 0.
- An attempt to read the MSR indexed by bits 31:0 would cause a general-protection exception if executed via RDMSR with CPL = 0.

A VMX abort occurs if processing fails for any entry. See Section 27.7.

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM exit functions as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

27.5 LOADING HOST STATE

Processor state is updated on VM exits in the following ways:

- Some state is loaded from or otherwise determined by the contents of the host-state area.
- Some state is determined by VM-exit controls.
- Some state is established in the same way on every VM exit.
- The page-directory pointers are loaded based on the values of certain control registers.

This loading may be performed in any order.

On processors that support Intel 64 architecture, the full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of the logical processor before and after the VM exit.

The loading of host state is detailed in Section 27.5.1 to Section 27.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

A logical processor is in IA-32e mode after a VM exit only if the “host address-space size” VM-exit control is 1. If the logical processor was in IA-32e mode before the VM exit and this control is 0, a VMX abort occurs. See Section 27.7.

In addition to loading host state, VM exits clear address-range monitoring (Section 27.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 27.6). This loading occurs only after the state loading described in this section.

27.5.1 Loading Host Control Registers, Debug Registers, MSRs

VM exits load new values for controls registers, debug registers, and some MSRs:

- CR0, CR3, and CR4 are loaded from the CR0 field, the CR3 field, and the CR4 field, respectively, with the following exceptions:
  - The following bits are not modified:

    - For CR0, ET, CD, NW; bits 63:32 (on processors that support Intel 64 architecture), 28:19, 17, and 15:6; and any bits that are fixed in VMX operation (see Section 23.8).[1]
    - For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).[2] (This item applies only to processors that support Intel 64 architecture.)
    - For CR4, any bits that are fixed in VMX operation (see Section 23.8).
  - CR4.PAE is set to 1 if the "host address-space size" VM-exit control is 1.
  - CR4.PCIDE is set to 0 if the "host address-space size" VM-exit control is 0.
- DR7 is set to 400H.
- The following MSRs are established as follows:
  - The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
  - The IA32_SYSENTER_CS MSR is loaded from the IA32_SYSENTER_CS field. Since that field has only 32 bits, bits 63:32 of the MSR are cleared to 0.
  - IA32_SYSENTER_ESP MSR and IA32_SYSENTER_EIP MSR are loaded from the IA32_SYSENTER_ESP field and the IA32_SYSENTER_EIP field, respectively.

    If the processor does not support the Intel 64 architecture, these fields have only 32 bits; bits 63:32 of the MSRs are cleared to 0.

    If the processor does support the Intel 64 architecture and the processor supports N < 64 linear-address bits, each of bits 63:N is set to the value of bit N–1.[3]
  - The following steps are performed on processors that support Intel 64 architecture:
    - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 27.5.2).
    - The LMA and LME bits in the IA32_EFER MSR are each loaded with the setting of the "host address-space size" VM-exit control.
  - If the "load IA32_PERF_GLOBAL_CTRL" VM-exit control is 1, the IA32_PERF_GLOBAL_CTRL MSR is loaded from the IA32_PERF_GLOBAL_CTRL field. Bits that are reserved in that MSR are maintained with their reserved values.

---

1. Bits 28:19, 17, and 15:6 of CR0 and CR0.ET are unchanged by executions of MOV to CR0. CR0.ET is always 1 and the other bits are always 0.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
3. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

- — If the “load IA32_PAT” VM-exit control is 1, the IA32_PAT MSR is loaded from the IA32_PAT field. Bits that are reserved in that MSR are maintained with their reserved values.
- — If the “load IA32_EFER” VM-exit control is 1, the IA32_EFER MSR is loaded from the IA32_EFER field. Bits that are reserved in that MSR are maintained with their reserved values.

With the exception of FS.base and GS.base, any of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 27.6.

27.5.2 Loading Host Segment and Descriptor-Table Registers

Each of the registers CS, SS, DS, ES, FS, GS, and TR is loaded as follows (see below for the treatment of LDTR):

- The selector is loaded from the selector field. The segment is unusable if its selector is loaded with zero. The checks specified Section 26.3.1.2 limit the selector values that may be loaded. In particular, CS and TR are never loaded with zero and are thus never unusable. SS can be loaded with zero only on processors that support Intel 64 architecture and only if the VM exit is to 64-bit mode (64-bit mode allows use of segments marked unusable).
- The base address is set as follows:
  - — CS. Cleared to zero.
  - — SS, DS, and ES. Undefined if the segment is unusable; otherwise, cleared to zero.
  - — FS and GS. Undefined (but, on processors that support Intel 64 architecture, canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field.

    If the processor supports the Intel 64 architecture and the processor supports N < 64 linear-address bits, each of bits 63:N is set to the value of bit N–1.[1] The values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.
  - — TR. Loaded from the host-state area. If the processor supports the Intel 64 architecture and the processor supports N < 64 linear-address bits, each of bits 63:N is set to the value of bit N–1.
- The segment limit is set as follows:
  - — CS. Set to FFFFFFFFH (corresponding to a descriptor limit of FFFFFH and a G-bit setting of 1).
  - — SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to FFFFFFFFH.

1. Software can determine the number N by executing CPUID with 80000008H in EAX. The number of linear-address bits supported is returned in bits 15:8 of EAX.

   — TR. Set to 00000067H.

- The type field and S bit are set as follows:
   — CS. Type set to 11 and S set to 1 (execute/read, accessed, non-conforming code segment).
   — SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, type set to 3 and S set to 1 (read/write, accessed, expand-up data segment).
   — TR. Type set to 11 and S set to 0 (busy 32-bit task-state segment).
- The DPL is set as follows:
   — CS, SS, and TR. Set to 0. The current privilege level (CPL) will be 0 after the VM exit completes.
   — DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 0.
- The P bit is set as follows:
   — CS, TR. Set to 1.
   — SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
- On processors that support Intel 64 architecture, CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because the value of this control is also loaded into IA32_EFER.LMA (see Section 27.5.1), no VM exit is ever to compatibility mode (which requires IA32_EFER.LMA = 1 and CS.L = 0).
- D/B.
   — CS. Loaded with the inverse of the setting of the “host address-space size” VM-exit control. For example, if that control is 0, indicating a 32-bit guest, CS.D/B is set to 1.
   — SS. Set to 1.
   — DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
   — TR. Set to 0.
- G.
   — CS. Set to 1.
   — SS, DS, ES, FS, and GS. Undefined if the segment is unusable; otherwise, set to 1.
   — TR. Set to 0.

The host-state area does not contain a selector field for LDTR. LDTR is established as follows on all VM exits: the selector is cleared to 0000H, the segment is marked unusable and is otherwise undefined (although the base address is always canonical).

The base addresses for GDTR and IDTR are loaded from the GDTR base-address field and the IDTR base-address field, respectively. If the processor supports the Intel 64 architecture and the processor supports N < 64 linear-address bits, each of bits 63:N of each base address is set to the value of bit N–1 of that base address. The GDTR and IDTR limits are each set to FFFFH.

27.5.3 Loading Host RIP, RSP, and RFLAGS

RIP and RSP are loaded from the RIP field and the RSP field, respectively. RFLAGS is cleared, except bit 1, which is always set.

27.5.4 Checking and Loading Host Page-Directory-Pointer-Table Entries

If CR0.PG = 1, CR4.PAE = 1, and IA32_EFER.LMA = 0, the logical processor uses **PAE paging**. See Section 4.4 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.[1] When in PAE paging is in use, the physical address in CR3 references a table of **page-directory-pointer-table entries** (PDPTEs). A MOV to CR3 when PAE paging is in use checks the validity of the PDPTEs and, if they are valid, loads them into the processor (into internal, non-architectural registers).

A VM exit is to a VMM that uses PAE paging if (1) bit 5 (corresponding to CR4.PAE) is set in the CR4 field in the host-state area of the VMCS; and (2) the "host address-space size" VM-exit control is 0. Such a VM exit may check the validity of the PDPTEs referenced by the CR3 field in the host-state area of the VMCS. Such a VM exit must check their validity if either (1) PAE paging was not in use before the VM exit; or (2) the value of CR3 is changing as a result of the VM exit. A VM exit to a VMM that does not use PAE paging must not check the validity of the PDPTEs.

A VM exit that checks the validity of the PDPTEs uses the same checks that are used when CR3 is loaded with MOV to CR3 when PAE paging is in use. If MOV to CR3 would cause a general-protection exception due to the PDPTEs that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs (see Section 27.7). If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTEs are loaded into the processor as would MOV to CR3, using the value of CR3 being load by the VM exit.

27.5.5 Updating Non-Register State

VM exits affect the non-register state of a logical processor as follows:

- A logical processor is always in the active state after a VM exit.

1. On processors that support Intel 64 architecture, the physical-address extension may support more than 36 physical-address bits. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

- Event blocking is affected as follows:
  - There is no blocking by STI or by MOV SS after a VM exit.
  - VM exits caused directly by non-maskable interrupts (NMIs) cause blocking by NMI (see Table 24-3). Other VM exits do not affect blocking by NMI. (See Section 27.1 for the case in which an NMI causes a VM exit indirectly.)
- There are no pending debug exceptions after a VM exit.

Section 28.3 describes how the VMX architecture controls how a logical processor manages information in the TLBs and paging-structure caches. The following items detail how VM exits invalidate cached mappings:

- If the “enable VPID” VM-execution control is 0, the logical processor invalidates linear mappings and combined mappings associated with VPID 0000H (for all PCIDs); combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP).
- VM exits are not required to invalidate any guest-physical mappings, nor are they required to invalidate any linear mappings or combined mappings if the “enable VPID” VM-execution control is 1.

27.5.6 Clearing Address-Range Monitoring

The Intel 64 and IA-32 architectures allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 8.10.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*. VM exits clear any address-range monitoring that may be in effect.

27.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 24.7.2). Specifically each entry in that area (up to the number specified in the VM-exit MSR-load count) is processed in order by loading the MSR indexed by bits 31:0 with the contents of bits 127:64 as they would be written by WRMSR.

Processing of an entry fails in any of the following cases:

- The value of bits 31:0 is either C0000100H (the IA32_FS_BASE MSR) or C0000101H (the IA32_GS_BASE MSR).
- The value of bits 31:8 is 000008H, meaning that the indexed MSR is one that allows access to an APIC register when the local APIC is in x2APIC mode.
- The value of bits 31:0 indicates an MSR that can be written only in system-management mode (SMM) and the VM exit will not end in SMM. (IA32_SMM_MONITOR_CTL is an MSR that can be written only in SMM.)
- The value of bits 31:0 indicates an MSR that cannot be loaded on VM exits for model-specific reasons. A processor may prevent loading of certain MSRs even if

they can normally be written by WRMSR. Such model-specific behavior is documented in Chapter 34.

- Bits 63:32 are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 of the entry would cause a general-protection exception if executed via WRMSR with CPL = 0.[1]

If processing fails for any entry, a VMX abort occurs. See Section 27.7.

If any MSR is being loaded in such a way that would architecturally require a TLB flush, the TLBs are updated so that, after VM exit, the logical processor does not use any translations that were cached before the transition.

27.7 VMX ABORTS

A problem encountered during a VM exit leads to a **VMX abort**. A VMX abort takes a logical processor into a shutdown state as described below.

A VMX abort does not modify the VMCS data in the VMCS region of any active VMCS. The contents of these data are thus suspect after the VMX abort.

On a VMX abort, a logical processor saves a nonzero 32-bit VMX-abort indicator field at byte offset 4 in the VMCS region of the VMCS whose misconfiguration caused the failure (see Section 24.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 27.4).
2. Host checking of the page-directory-pointer-table entries (PDPTEs) failed (see Section 27.5.4).
3. The current VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the logical processor cannot complete the VM exit properly.
4. There was a failure on loading host MSRs (see Section 27.6).
5. There was a machine-check event during VM exit (see Section 27.8).
6. The logical processor was in IA-32e mode before the VM exit and the "host address-space size" VM-entry control was 0 (see Section 27.5).

Some of these causes correspond to failures during the loading of state from the host-state area. Because the loading of such state may be done in any order (see Section 27.5) a VM exit that might lead to a VMX abort for multiple reasons (for example, the current VMCS may be corrupt and the host PDPTEs might not be prop-

1. Note the following about processors that support Intel 64 architecture. If CR0.PG = 1, WRMSR to the IA32_EFER MSR causes a general-protection exception if it would modify the LME bit. Since CR0.PG is always 1 in VMX operation, the IA32_EFER MSR should not be included in the VM-exit MSR-load area for the purpose of modifying the LME bit.

erly configured). In such cases, the VMX-abort indicator could correspond to any one of those reasons.

A logical processor never reads the VMX-abort indicator in a VMCS region and writes it only with one of the non-zero values mentioned above. The VMX-abort indicator allows software on one logical processor to diagnose the VMX-abort on another. For this reason, it is recommended that software running in VMX root operation zero the VMX-abort indicator in the VMCS region of any VMCS that it uses.

After saving the VMX-abort indicator, operation of a logical processor experiencing a VMX abort depends on whether the logical processor is in SMX operation:[1]

- If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000DH, indicating "VMX abort." See *Intel® Trusted Execution Technology Measured Launched Environment Programming Guide*.
- If the logical processor is outside SMX operation, it issues a special bus cycle (to notify the chipset) and enters the **VMX-abort shutdown state**. RESET is the only event that wakes a logical processor from the VMX-abort shutdown state. The following events do not affect a logical processor in this state: machine-check events; INIT signals; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

27.8 MACHINE-CHECK EVENTS DURING VM EXIT

If a machine-check event occurs during VM exit, one of the following occurs:

- The machine-check event is handled as if it occurred before the VM exit:
  - If CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:[2]
    - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs. The error code used is 000CH, indicating "unrecoverable machine-check condition."
    - If the logical processor is outside SMX operation, it goes to the shutdown state.

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

2. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

- — If CR4.MCE = 1, a machine-check exception (#MC) is generated:
    - If bit 18 (#MC) of the exception bitmap is 0, the exception is delivered through the guest IDT.
    - If bit 18 of the exception bitmap is 1, the exception causes a VM exit.
- The machine-check event is handled after VM exit completes:
  - — If the VM exit ends with CR4.MCE = 0, operation of the logical processor depends on whether the logical processor is in SMX operation:
    - If the logical processor is in SMX operation, an Intel® TXT shutdown condition occurs with error code 000CH (unrecoverable machine-check condition).
    - If the logical processor is outside SMX operation, it goes to the shutdown state.
  - — If the VM exit ends with CR4.MCE = 1, a machine-check exception (#MC) is delivered through the host IDT.
- A VMX abort is generated (see Section 27.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for "machine-check event during VM exit."

The first option is not used if the machine-check event occurs after any host state has been loaded. The second option is used only if VM entry is able to load all host state.

CHAPTER 28
VMX SUPPORT FOR ADDRESS TRANSLATION

The architecture for VMX operation includes two features that support address translation: virtual-processor identifiers (VPIDs) and the extended page-table mechanism (EPT). VPIDs are a mechanism for managing translations of linear addresses. EPT defines a layer of address translation that augments the translation of linear addresses.

Section 28.1 details the architecture of VPIDs. Section 28.2 provides the details of EPT. Section 28.3 explains how a logical processor may cache information from the paging structures, how it may use that cached information, and how software can managed the cached information.

28.1 VIRTUAL PROCESSOR IDENTIFIERS (VPIDS)

The original architecture for VMX operation required VMX transitions to flush the TLBs and paging-structure caches. This ensured that translations cached for the old linear-address space would not be used after the transition.

Virtual-processor identifiers (**VPIDs**) introduce to VMX operation a facility by which a logical processor may cache information for multiple linear-address spaces. When VPIDs are used, VMX transitions may retain cached information and the logical processor switches to a different linear-address space.

Section 28.3 details the mechanisms by which a logical processor manages information cached for multiple address spaces. A logical processor may tag some cached information with a 16-bit VPID. This section specifies how the current VPID is determined at any point in time:

- The current VPID is 0000H in the following situations:
  - Outside VMX operation. (This includes operation in system-management mode under the default treatment of SMIs and SMM with VMX operation; see Section 29.14.)
  - In VMX root operation.
  - In VMX non-root operation when the “enable VPID” VM-execution control is 0.
- If the logical processor is in VMX non-root operation and the “enable VPID” VM-execution control is 1, the current VPID is the value of the VPID VM-execution control field in the VMCS. (VM entry ensures that this value is never 0000H; see Section 26.2.1.1.)

VPIDs and PCIDs (see Section 4.10.1) can be used concurrently. When this is done, the processor associates cached information with both a VPID and a PCID. Such information is used only if the current VPID and PCID **both** match those associated with the cached information.

28.2 THE EXTENDED PAGE TABLE MECHANISM (EPT)

The extended page-table mechanism (**EPT**) is a feature that can be used to support the virtualization of physical memory. When EPT is in use, certain addresses that would normally be treated as physical addresses (and used to access memory) are instead treated as **guest-physical addresses**. Guest-physical addresses are translated by traversing a set of **EPT paging structures** to produce physical addresses that are used to access memory.

- Section 28.2.1 gives an overview of EPT.
- Section 28.2.2 describes operation of EPT-based address translation.
- Section 28.2.3 discusses VM exits that may be caused by EPT.
- Section 28.2.4 describes interactions between EPT and memory typing.

28.2.1 EPT Overview

EPT is used when the "enable EPT" VM-execution control is 1.[1] It translates the guest-physical addresses used in VMX non-root operation and those used by VM entry for event injection.

The translation from guest-physical addresses to physical addresses is determined by a set of **EPT paging structures**. The EPT paging structures are similar to those used to translate linear addresses while the processor is in IA-32e mode. Section 28.2.2 gives the details of the EPT paging structures.

If CR0.PG = 1, linear addresses are translated through paging structures referenced through control register CR3 . While the "enable EPT" VM-execution control is 1, these are called **guest paging structures**. There are no guest paging structures if CR0.PG = 0.[2]

1. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, the logical processor operates as if the "enable EPT" VM-execution control were 0. See Section 24.6.2.

2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

When the “enable EPT” VM-execution control is 1, the identity of **guest-physical addresses** depends on the value of CR0.PG:

- If CR0.PG = 0, each linear address is treated as a guest-physical address.
- If CR0.PG = 1, guest-physical addresses are those derived from the contents of control register CR3 and the guest paging structures. (This includes the values of the PDPTEs, which logical processors store in internal, non-architectural registers.) The latter includes (in page-table entries and in other paging-structure entries for which bit 7—PS—is 1) the addresses to which linear addresses are translated by the guest paging structures.

If CR0.PG = 1, the translation of a linear address to a physical address requires multiple translations of guest-physical addresses using EPT. Assume, for example, that CR4.PAE = CR4.PSE = 0. The translation of a 32-bit linear address then operates as follows:

- Bits 31:22 of the linear address select an entry in the guest page directory located at the guest-physical address in CR3. The guest-physical address of the guest page-directory entry (PDE) is translated through EPT to determine the guest PDE’s physical address.
- Bits 21:12 of the linear address select an entry in the guest page table located at the guest-physical address in the guest PDE. The guest-physical address of the guest page-table entry (PTE) is translated through EPT to determine the guest PTE’s physical address.
- Bits 11:0 of the linear address is the offset in the page frame located at the guest-physical address in the guest PTE. The guest-physical address determined by this offset is translated through EPT to determine the physical address to which the original linear address translates.

In addition to translating a guest-physical address to a physical address, EPT specifies the privileges that software is allowed when accessing the address. Attempts at disallowed accesses are called **EPT violations** and cause VM exits. See Section 28.2.3.

A logical processor uses EPT to translate guest-physical addresses only when those addresses are used to access memory. This principle implies the following:

- The MOV to CR3 instruction loads CR3 with a guest-physical address. Whether that address is translated through EPT depends on whether PAE paging is being used.[1]
  - If PAE paging is not being used, the instruction does not use that address to access memory and does **not** cause it to be translated through EPT. (If CR0.PG = 1, the address will be translated through EPT on the next memory accessing using a linear address.)

1. A logical processor uses PAE paging if CR0.PG = 1, CR4.PAE = 1 and IA32_EFER.LMA = 0. See Section 4.4 in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

  - If PAE paging is being used, the instruction loads the four (4) page-directory-pointer-table entries (PDPTEs) from that address and it **does** cause the address to be translated through EPT.
- Section 4.4.1 identifies executions of MOV to CR0 and MOV to CR4 that load the PDPTEs from the guest-physical address in CR3. Such executions cause that address to be translated through EPT.
- The PDPTEs contain guest-physical addresses. The instructions that load the PDPTEs (see above) do not use those addresses to access memory and do **not** cause them to be translated through EPT. The address in a PDPTE will be translated through EPT on the next memory accessing using a linear address that uses that PDPTE.

28.2.2 EPT Translation Mechanism

The EPT translation mechanism uses only bits 47:0 of each guest-physical address.[1] It uses a page-walk length of 4, meaning that at most 4 EPT paging-structure entries are accessed to translate a guest-physical address.[2]

These 48 bits are partitioned by the logical processor to traverse the EPT paging structures:

- A 4-KByte naturally aligned EPT PML4 table is located at the physical address specified in bits 51:12 of the extended-page-table pointer (EPTP), a VM-execution control field (see Table 24-8 in Section 24.6.11). An EPT PML4 table comprises 512 64-bit entries (EPT PML4Es). An EPT PML4E is selected using the physical address defined as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPTP.
  - Bits 11:3 are bits 47:39 of the guest-physical address.
  - Bits 2:0 are all 0.

  Because an EPT PML4E is identified using bits 47:39 of the guest-physical address, it controls access to a 512-GByte region of the guest-physical-address space.
- A 4-KByte naturally aligned EPT page-directory-pointer table is located at the physical address specified in bits 51:12 of the EPT PML4E (see Table 28-1). An

1. No processors supporting the Intel 64 architecture support more than 48 physical-address bits. Thus, no such processor can produce a guest-physical address with more than 48 bits. An attempt to use such an address causes a page fault. An attempt to load CR3 with such an address causes a general-protection fault. If PAE paging is being used, an attempt to load CR3 that would load a PDPTE with such an address causes a general-protection fault.
2. Future processors may include support for other EPT page-walk lengths. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine what EPT page-walk lengths are supported.

EPT page-directory-pointer table comprises 512 64-bit entries (PDPTEs). An EPT PDPTE is selected using the physical address defined as follows:

Table 28-1. Format of an EPT PML4 Entry (PML4E)

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 512-GByte region controlled by this entry
1	Write access; indicates whether writes are allowed to the 512-GByte region controlled by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 512-GByte region controlled by this entry
7:3	Reserved (must be 0)
11:8	Ignored
(N–1):12	Physical address of 4-KByte aligned EPT page-directory-pointer table referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the processor. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

— Bits 63:52 are all 0.

— Bits 51:12 are from the EPT PML4 entry.

— Bits 11:3 are bits 38:30 of the guest-physical address.

— Bits 2:0 are all 0.

Because a PDPTE is identified using bits 47:30 of the guest-physical address, it controls access to a 1-GByte region of the guest-physical-address space. Use of the PDPTE depends on the value of bit 7 in that entry:[1]

- If bit 7 of the EPT PDPTE is 1, the EPT PDPTE maps a 1-GByte page (see Table 28-2). The final physical address is computed as follows:

1. Not all processors allow bit 7 of an EPT PDPTE to be set to 1. Software should read the VMX capability MSR IA32_VMX_EPT_VPID_CAP (see Appendix A.10) to determine whether this is allowed.

Table 28-2. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that Maps a 1-GByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte page referenced by this entry
1	Write access; indicates whether writes are allowed to the 1-GByte page referenced by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 1-GByte page referenced by this entry
5:3	EPT memory type for this 1-GByte page (see Section 28.2.4)
6	Ignore PAT memory type for this 1-GByte page (see Section 28.2.4)
7	Must be 1 (otherwise, this entry references an EPT page directory)
11:8	Ignored
29:12	Reserved (must be 0)
(N–1):30	Physical address of the 1-GByte page referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

- — Bits 63:52 are all 0.
- — Bits 51:30 are from the EPT PDPTE.
- — Bits 29:0 are from the original guest-physical address.

- If bit 7 of the EPT PDPTE is 0, a 4-KByte naturally aligned EPT page directory is located at the physical address specified in bits 51:12 of the EPT PDPTE (see Table 28-3). An EPT page-directory comprises 512 64-bit entries (PDEs). An EPT PDE is selected using the physical address defined as follows:
  - — Bits 63:52 are all 0.
  - — Bits 51:12 are from the EPT PDPTE.
  - — Bits 11:3 are bits 29:21 of the guest-physical address.
  - — Bits 2:0 are all 0.

Table 28-3. Format of an EPT Page-Directory-Pointer-Table Entry (PDPTE) that References an EPT Page Directory

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 1-GByte region controlled by this entry
1	Write access; indicates whether writes are allowed to the 1-GByte region controlled by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 1-GByte region controlled by this entry
7:3	Reserved (must be 0)
11:8	Ignored
(N–1):12	Physical address of 4-KByte aligned EPT page directory referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

Because an EPT PDE is identified using bits 47:21 of the guest-physical address, it controls access to a 2-MByte region of the guest-physical-address space. Use of the EPT PDE depends on the value of bit 7 in that entry:

- If bit 7 of the EPT PDE is 1, the EPT PDE maps a 2-MByte page (see Table 28-4). The final physical address is computed as follows:
  - Bits 63:52 are all 0.
  - Bits 51:21 are from the EPT PDE.
  - Bits 20:0 are from the original guest-physical address.
- If bit 7 of the EPT PDE is 0, a 4-KByte naturally aligned EPT page table is located at the physical address specified in bits 51:12 of the EPT PDE (see Table 28-5). An EPT page table comprises 512 64-bit entries (PTEs). An EPT PTE is selected using a physical address defined as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPT PDE.
  - Bits 11:3 are bits 20:12 of the guest-physical address.
  - Bits 2:0 are all 0.

Table 28-4. Format of an EPT Page-Directory Entry (PDE) that Maps a 2-MByte Page

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte page referenced by this entry
1	Write access; indicates whether writes are allowed to the 2-MByte page referenced by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 2-MByte page referenced by this entry
5:3	EPT memory type for this 2-MByte page (see Section 28.2.4)
6	Ignore PAT memory type for this 2-MByte page (see Section 28.2.4)
7	Must be 1 (otherwise, this entry references an EPT page table)
11:8	Ignored
20:12	Reserved (must be 0)
(N–1):21	Physical address of the 2-MByte page referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

- Because an EPT PTE is identified using bits 47:12 of the guest-physical address, every EPT PTE maps a 4-KByte page (see Table 28-6). The final physical address is computed as follows:
  - Bits 63:52 are all 0.
  - Bits 51:12 are from the EPT PTE.
  - Bits 11:0 are from the original guest-physical address.

If bits 2:0 of an EPT paging-structure entry are all 0, the entry is **not present**. The processor ignores bits 63:3 and does uses the entry neither to reference another EPT paging-structure entry nor to produce a physical address. A reference using a guest-physical address whose translation encounters an EPT paging-structure that is not present causes an EPT violation (see Section 28.2.3.2).

The discussion above describes how the EPT paging structures reference each other and how the logical processor traverses those structures when translating a guest-physical address. It does not cover all details of the translation process. Additional details are provided as follows:

Table 28-5. Format of an EPT Page-Directory Entry (PDE) that References an EPT Page Table

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 2-MByte region controlled by this entry
1	Write access; indicates whether writes are allowed to the 2-MByte region controlled by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 2-MByte region controlled by this entry
6:3	Reserved (must be 0)
7	Must be 0 (otherwise, this entry maps a 2-MByte page)
11:8	Ignored
(N–1):12	Physical address of 4-KByte aligned EPT page table referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

- Situations in which the translation process may lead to VM exits (sometimes before the process completes) are described in Section 28.2.3.
- Interactions between the EPT translation mechanism and memory typing are described in Section 28.2.4.

Figure 28-1 gives a summary of the formats of the EPTP and the EPT paging-structure entries. For the EPT paging structure entries, it identifies separately the format of entries that map pages, those that reference other EPT paging structures, and those that do neither because they are "not present"; bits 2:0 and bit 7 are highlighted because they determine how a paging-structure entry is used.

28.2.3 EPT-Induced VM Exits

Accesses using guest-physical addresses may cause VM exits due to **EPT misconfigurations** and **EPT violations**. An EPT misconfiguration occurs when, in the course of translation a guest-physical address, the logical processor encounters an EPT paging-structure entry that contains an unsupported value. An EPT violation occurs when there is no EPT misconfiguration but the EPT paging-structure entries disallow an access using the guest-physical address.

Table 28-6. Format of an EPT Page-Table Entry

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 4-KByte page referenced by this entry
1	Write access; indicates whether writes are allowed to the 4-KByte page referenced by this entry
2	Execute access; indicates whether instruction fetches are allowed from the 4-KByte page referenced by this entry
5:3	EPT memory type for this 4-KByte page (see Section 28.2.4)
6	Ignore PAT memory type for this 4-KByte page (see Section 28.2.4)
11:7	Ignored
(N–1):12	Physical address of the 4-KByte page referenced by this entry[1]
51:N	Reserved (must be 0)
63:52	Ignored

NOTES:

1. N is the physical-address width supported by the logical processor.

EPT misconfigurations and EPT violations occur only due to an attempt to access memory with a guest-physical address. Loading CR3 with a guest-physical address with the MOV to CR3 instruction can cause neither an EPT configuration nor an EPT violation until that address is used to access a paging structure.[1]

28.2.3.1 EPT Misconfigurations

AN EPT misconfiguration occurs if any of the following is identified while translating a guest-physical address:

- The value of bits 2:0 of an EPT paging-structure entry is either 010b (write-only) or 110b (write/execute).
- The value of bits 2:0 of an EPT paging-structure entry is 100b (execute-only) and this value is not supported by the logical processor. Software should read the

1. If the logical processor is using PAE paging—because CR0.PG = CR4.PAE = 1 and IA32_EFER.LMA = 0—the MOV to CR3 instruction loads the PDPTEs from memory using the guest-physical address being loaded into CR3. In this case, therefore, the MOV to CR3 instruction may cause an EPT misconfiguration or an EPT violation.

63 … 52	51 … M[1]	M-1 … 32, 31 … 12	11 … 8	7	6	5 … 3	2	1	0	
Reserved		Address of EPT PML4 table	Reserved			EPT PWL-1	EPT PS MT			EPTP[2]
Ignored	Rsvd.	Address of EPT page-directory-pointer table	Ign.	Reserved			X	W	R	PML4E: present
Ignored							0	0	0	PML4E: not present
Ignored	Rsvd.	Physical address of 1GB page / Reserved	Ign.	1	IPAT	EPT MT	X	W	R	PDPTE: 1GB page
Ignored	Rsvd.	Address of EPT page directory	Ign.	0	Rsvd.		X	W	R	PDPTE: page directory
Ignored							0	0	0	PDTPE: not present
Ignored	Rsvd.	Physical address of 2MB page / Reserved	Ign.	1	IPAT	EPT MT	X	W	R	PDE: 2MB page
Ignored	Rsvd.	Address of EPT page table	Ign.	0	Rsvd.		X	W	R	PDE: page table
Ignored							0	0	0	PDE: not present
Ignored	Rsvd.	Physical address of 4KB page	Ign.		IPAT	EPT MT	X	W	R	PTE: 4KB page
Ignored							0	0	0	PTE: not present

Figure 28-1. Formats of EPTP and EPT Paging-Structure Entries

NOTES:

1. M is an abbreviation for MAXPHYADDR.
2. See Section 24.6.11 for details of the EPTP.

VMX capability MSR IA32_VMX_EPT_VPID_CAP to determine whether this value is supported (see Appendix A.10).

- The value of bits 2:0 of an EPT paging-structure entry is not 000b (the entry is present) **and** one of the following holds:
  - A reserved bit is set. This includes the setting of a bit in the range 51:12 that is beyond the logical processor's physical-address width.[1] See Section 28.2.2 for details of which bits are reserved in which EPT paging-structure entries.
  - The entry is the last one used to translate a guest physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE) and the value of bits 5:3 (EPT memory type) is 2, 3, or 7 (these values are reserved).

EPT misconfigurations result when an EPT paging-structure entry is configured with settings reserved for future functionality. Software developers should be aware that such settings may be used in the future and that an EPT paging-structure entry that causes an EPT misconfiguration on one processor might not do so in the future.

28.2.3.2 EPT Violations

An EPT violation may occur during an access using a guest-physical address whose translation does not cause an EPT misconfiguration. An EPT violation occurs in any of the following situations:

- Translation of the guest-physical address encounters an EPT paging-structure entry that is not present (see Section 28.2.2).
- The access is a data read and bit 0 was clear in any of the EPT paging-structure entries used to translate the guest-physical address. Reads by the logical processor of guest paging structures to translate a linear address are considered to be data reads.
- The access is a data write and bit 1 was clear in any of the EPT paging-structure entries used to translate the guest-physical address. Writes by the logical processor to guest paging structures to update accessed and dirty flags are considered to be data writes.
- The access is an instruction fetch and bit 2 was clear in any of the EPT paging-structure entries used to translate the guest-physical address.

28.2.3.3 Prioritization of EPT-Induced VM Exits

The translation of a linear address to a physical address requires one or more translations of guest-physical addresses using EPT (see Section 28.2.1). This section specifies the relative priority of EPT-induced VM exits with respect to each other and to other events that may be encountered when accessing memory using a linear address.

For an access to a guest-physical address, determination of whether an EPT misconfiguration or an EPT violation occurs is based on an iterative process:[2]

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

1. An EPT paging-structure entry is read (initially, this is an EPT PML4 entry):
    a. If the entry is not present (bits 2:0 are all 0), an EPT violation occurs.
    b. If the entry is present but its contents are not configured properly (see Section 28.2.3.1), an EPT misconfiguration occurs.
    c. If the entry is present and its contents are configured properly, operation depends on whether the entry references another EPT paging structure (whether it is an EPT PDE with bit 7 set to 1 or an EPT PTE):
        i) If the entry does reference another EPT paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
        ii) Otherwise, the entry is used to produce the ultimate physical address (the translation of the original guest-physical address); step 2 is executed.
2. Once the ultimate physical address is determined, the privileges determined by the EPT paging-structure entries are evaluated:
    a. If the access to the guest-physical address is not allowed by these privileges (see Section 28.2.3.2), an EPT violation occurs.
    b. If the access to the guest-physical address is allowed by these privileges, memory is accessed using the ultimate physical address.

If CR0.PG = 1, the translation of a linear address is also an iterative process, with the processor first accessing an entry in the guest paging structure referenced by the guest-physical address in CR3 (or, if PAE paging is in use, the guest-physical address in the appropriate PDPTE register), then accessing an entry in another guest paging structure referenced by the guest-physical address in the first guest paging-structure entry, etc. Each guest-physical address is itself translated using EPT and may cause an EPT-induced VM exit. The following items detail how page faults and EPT-induced VM exits are recognized during this iterative process:

1. An attempt is made to access a guest paging-structure entry with a guest-physical address (initially, the address in CR3 or PDPTE register).
    a. If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs.
    b. If the access does not cause an EPT-induced VM exit, bit 0 (the present flag) of the entry is consulted:
        i) If the present flag is 0 or any reserved bit is set, a page fault occurs.
        ii) If the present flag is 1, no reserved bit is set, operation depends on whether the entry references another guest paging structure (whether it is a guest PDE with PS = 1 or a guest PTE):

2. This is a simplification of the more detailed description given in Section 28.2.2.

- If the entry does reference another guest paging structure, an entry from that structure is accessed; step 1 is executed for that other entry.
- Otherwise, the entry is used to produce the ultimate guest-physical address (the translation of the original linear address); step 2 is executed.

2. Once the ultimate guest-physical address is determined, the privileges determined by the guest paging-structure entries are evaluated:
   a. If the access to the linear address is not allowed by these privileges (e.g., it was a write to a read-only page), a page fault occurs.
   b. If the access to the linear address is allowed by these privileges, an attempt is made to access memory at the ultimate guest-physical address:
      i) If the access fails because of an EPT misconfiguration or an EPT violation (see above), an EPT-induced VM exit occurs.
      ii) If the access does not cause an EPT-induced VM exit, memory is accessed using the ultimate physical address (the translation, using EPT, of the ultimate guest-physical address).

If CR0.PG = 0, a linear address is treated as a guest-physical address and is translated using EPT (see above). This process, if it completes without an EPT violation or EPT misconfiguration, produces a physical address and determines the privileges allowed by the EPT paging-structure entries. If these privileges do not allow the access to the physical address (see Section 28.2.3.2), an EPT violation occurs. Otherwise, memory is accessed using the physical address.

28.2.4 EPT and Memory Typing

This section specifies how a logical processor determines the memory type use for a memory access while EPT is in use. (See Chapter 11, “Memory Cache Control” of *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A* for details of memory typing in the Intel 64 architecture.) Section 28.2.4.1 explains how the memory type is determined for accesses to the EPT paging structures. Section 28.2.4.2 explains how the memory type is determined for an access using a guest-physical address that is translated using EPT.

28.2.4.1 Memory Type Used for Accessing EPT Paging Structures

This section explains how the memory type is determined for accesses to the EPT paging structures. The determination is based first on the value of bit 30 (cache disable—CD) in control register CR0:

- If CR0.CD = 0, the memory type used for any such reference is the EPT paging-structure memory type, which is specified in bits 2:0 of the extended-page-table pointer (EPTP), a VM-execution control field (see Section 24.6.11). A value of 0 indicates the uncacheable type (UC), while a value of 6 indicates the write-back type (WB). Other values are reserved.

- If CR0.CD = 1, the memory type used for any such reference is uncacheable (UC).

The MTRRs have no effect on the memory type used for an access to an EPT paging structure.

28.2.4.2 Memory Type Used for Translated Guest-Physical Addresses

The **effective memory type** of a memory access using a guest-physical address (an access that is translated using EPT) is the memory type that is used to access memory. The effective memory type is based on the value of bit 30 (cache disable—CD) in control register CR0; the **last** EPT paging-structure entry used to translate the guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE); and the PAT memory type (see below):

- The **PAT memory type** depends on the value of CR0.PG:
  - If CR0.PG = 0, the PAT memory type is WB (writeback).[1]
  - If CR0.PG = 1, the PAT memory type is the memory type selected from the IA32_PAT MSR as specified in Section 11.12.3, “Selecting a Memory Type from the PAT”.[2]
- The **EPT memory type** is specified in bits 5:3 of the last EPT paging-structure entry: 0 = UC; 1 = WC; 4 = WT; 5 = WP; and 6 = WB. Other values are reserved and cause EPT misconfigurations (see Section 28.2.3).
- If CR0.CD = 0, the effective memory type depends upon the value of bit 6 of the last EPT paging-structure entry:
  - If the value is 0, the effective memory type is the combination of the EPT memory type and the PAT memory type specified in Table 11-7 in Section 11.5.2.2, using the EPT memory type in place of the MTRR memory type.
  - If the value is 1, the memory type used for the access is the EPT memory type. The PAT memory type is ignored.
- If CR0.CD = 1, the effective memory type is UC.

---

1. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the “unrestricted guest” VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.
2. Table 11-11 in Section 11.12.3, “Selecting a Memory Type from the PAT” illustrates how the PAT memory type is selected based on the values of the PAT, PCD, and PWT bits in a page-table entry (or page-directory entry with PS = 1). For accesses to a guest paging-structure entry X, the PAT memory type is selected from the table by using a value of 0 for the PAT bit with the values of PCD and PWT from the paging-structure entry Y that references X (or from CR3 if X is in the root paging structure). With PAE paging, the PAT memory type for accesses to the PDPTEs is WB.

The MTRRs have no effect on the memory type used for an access to a guest-physical address.

28.3 CACHING TRANSLATION INFORMATION

Processors supporting Intel® 64 and IA-32 architectures may accelerate the address-translation process by caching on the processor data from the structures in memory that control that process. Such caching is discussed in Section 4.10, "Caching Translation Information" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A.* The current section describes how this caching interacts with the VMX architecture.

The VPID and EPT features of the architecture for VMX operation augment this caching architecture. EPT defines the guest-physical address space and defines translations to that address space (from the linear-address space) and from that address space (to the physical-address space). Both features control the ways in which a logical processor may create and use information cached from the paging structures.

Section 28.3.1 describes the different kinds of information that may be cached. Section 28.3.2 specifies when such information may be cached and how it may be used. Section 28.3.3 details how software can invalidate cached information.

28.3.1 Information That May Be Cached

Section 4.10, "Caching Translation Information" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* identifies two kinds of translation-related information that may be cached by a logical processor: **translations**, which are mappings from linear page numbers to physical page frames, and **paging-structure caches**, which map the upper bits of a linear page number to information from the paging-structure entries used to translate linear addresses matching those upper bits.

The same kinds of information may be cached when VPIDs and EPT are in use. A logical processor may cache and use such information based on its function. Information with different functionality is identified as follows:

- **Linear mappings**.[1] There are two kinds:
  - Linear translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - Linear paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address

1. Earlier versions of this manual used the term "VPID-tagged" to identify linear mappings.

space, along with information about access privileges. For example, bits 47:39 of a linear address would map to the address of the relevant page-directory-pointer table.

Linear mappings do not contain information from any EPT paging structure.

- **Guest-physical mappings.**[1] There are two kinds:
  - — Guest-physical translations. Each of these is a mapping from a guest-physical page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - — Guest-physical paging-structure-cache entries. Each of these is a mapping from the upper portion of a guest-physical address to the physical address of the EPT paging structure used to translate the corresponding region of the guest-physical address space, along with information about access privileges.

  The information in guest-physical mappings about access privileges and memory typing is derived from EPT paging structures.
- **Combined mappings.**[2] There are two kinds:
  - — Combined translations. Each of these is a mapping from a linear page number to the physical page frame to which it translates, along with information about access privileges and memory typing.
  - — Combined paging-structure-cache entries. Each of these is a mapping from the upper portion of a linear address to the physical address of the paging structure used to translate the corresponding region of the linear-address space, along with information about access privileges.

  The information in combined mappings about access privileges and memory typing is derived from both guest paging structures and EPT paging structures.

28.3.2 Creating and Using Cached Translation Information

The following items detail the creation of the mappings described in the previous section:[3]

- The following items describe the creation of mappings while EPT is not in use (including execution outside VMX non-root operation):

1. Earlier versions of this manual used the term “EPTP-tagged” to identify guest-physical mappings.
2. Earlier versions of this manual used the term “dual-tagged” to identify combined mappings.
3. This section associated cached information with the current VPID and PCID. If PCIDs are not supported or are not being used (e.g., because CR4.PCIDE = 0), all the information is implicitly associated with PCID 000H; see Section 4.10.1, “Process-Context Identifiers (PCIDs),” in *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

  - Linear mappings may be created. They are derived from the paging structures referenced (directly or indirectly) by the current value of CR3 and are associated with the current VPID and the current PCID.
  - No linear mappings are created with information derived from paging-structure entries that are not present (bit 0 is 0) or that set reserved bits. For example, if a PTE is not present, no linear mapping are created for any linear page number whose translation would use that PTE.
  - No guest-physical or combined mappings are created while EPT is not in use.
- The following items describe the creation of mappings while EPT is in use:
  - Guest-physical mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by bits 51:12 of the current EPTP. These 40 bits contain the address of the EPT-PML4-table. (the notation **EP4TA** refers to those 40 bits). Newly created guest-physical mappings are associated with the current EP4TA.
  - Combined mappings may be created. They are derived from the EPT paging structures referenced (directly or indirectly) by the current EP4TA. If CR0.PG = 1, they are also derived from the paging structures referenced (directly or indirectly) by the current value of CR3. They are associated with the current VPID, the current PCID, and the current EP4TA.[1] No combined paging-structure-cache entries are created if CR0.PG = 0.[2]
  - No guest-physical mappings or combined mappings are created with information derived from EPT paging-structure entries that are not present (bits 2:0 are all 0) or that are misconfigured (see Section 28.2.3.1).
  - No combined mappings are created with information derived from guest paging-structure entries that are not present or that set reserved bits.
  - No linear mappings are created while EPT is in use.

The following items detail the use of the various mappings:

- If EPT is not in use (e.g., when outside VMX non-root operation), a logical processor may use cached mappings as follows:
  - For accesses using linear addresses, it may use linear mappings associated with the current VPID and the current PCID. It may also use global TLB entries (linear mappings) associated with the current VPID and any PCID.
  - No guest-physical or combined mappings are used while EPT is not in use.
- If EPT is in use, a logical processor may use cached mappings as follows:

1. At any given time, a logical processor may be caching combined mappings for a VPID and a PCID that are associated with different EP4TAs. Similarly, it may be caching combined mappings for an EP4TA that are associated with different VPIDs and PCIDs.
2. If the capability MSR IA32_VMX_CR0_FIXED0 reports that CR0.PG must be 1 in VMX operation, CR0.PG can be 0 in VMX non-root operation only if the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls are both 1.

- — For accesses using linear addresses, it may use combined mappings associated with the current VPID, the current PCID, and the current EP4TA. It may also use global TLB entries (combined mappings) associated with the current VPID, the current EP4TA, and any PCID.
- — For accesses using guest-physical addresses, it may use guest-physical mappings associated with the current EP4TA.
- — No linear mappings are used while EPT is in use.

28.3.3 Invalidating Cached Translation Information

Software modifications of paging structures (including EPT paging structures) may result in inconsistencies between those structures and the mappings cached by a logical processor. Certain operations invalidate information cached by a logical processor and can be used to eliminate such inconsistencies.

28.3.3.1 Operations that Invalidate Cached Mappings

The following operations invalidate cached mappings as indicated:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation (e.g., the INVLPG and INVPCID instructions) invalidate linear mappings and combined mappings.[1] They are required to do so only for the current VPID (but, for combined mappings, all EP4TAs). Linear mappings for the current VPID are invalidated even if EPT is in use.[2] Combined mappings for the current VPID are invalidated even if EPT is not in use.[3]
- An EPT violation invalidates any guest-physical mappings (associated with the current EP4TA) that would be used to translate the guest-physical address that caused the EPT violation. If that guest-physical address was the translation of a linear address, the EPT violation also invalidates any combined mappings for that linear address associated with the current PCID, the current VPID and the current EP4TA.
- If the “enable VPID” VM-execution control is 0, VM entries and VM exits invalidate linear mappings and combined mappings associated with VPID 0000H

1. See Section 4.10.4, “Invalidation of TLBs and Paging-Structure Caches,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A* for an enumeration of operations that architecturally invalidate entries in the TLBs and paging-structure caches independent of VMX operation.
2. While no linear mappings are created while EPT is in use, a logical processor may retain, while EPT is in use, linear mappings (for the same VPID as the current one) there were created earlier, when EPT was not in use.
3. While no combined mappings are created while EPT is not in use, a logical processor may retain, while EPT is in not use, combined mappings (for the same VPID as the current one) there were created earlier, when EPT was in use.

(for all PCIDs). Combined mappings for VPID 0000H are invalidated for all EP4TAs.

- Execution of the INVVPID instruction invalidates linear mappings and combined mappings. Invalidation is based on instruction operands, called the INVVPID type and the INVVPID descriptor. Four INVVPID types are currently defined:
  - **Individual-address.** If the INVVPID type is 0, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor and that would be used to translate the linear address specified in of the INVVPID descriptor. Linear mappings and combined mappings for that VPID and linear address are invalidated for all PCIDs and, for combined mappings, all EP4TAs. (The instruction may also invalidate mappings associated with other VPIDs and for other linear addresses.)
  - **Single-context.** If the INVVPID type is 1, the logical processor invalidates all linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EP4TAs. (The instruction may also invalidate mappings associated with other VPIDs.)
  - **All-context.** If the INVVPID type is 2, the logical processor invalidates linear mappings and combined mappings associated with all VPIDs except VPID 0000H and with all PCIDs. (The instruction may also invalidate linear mappings with VPID 0000H.) Combined mappings are invalidated for all EP4TAs.
  - **Single-context-retaining-globals.** If the INVVPID type is 3, the logical processor invalidates linear mappings and combined mappings associated with the VPID specified in the INVVPID descriptor. Linear mappings and combined mappings for that VPID are invalidated for all PCIDs and, for combined mappings, all EP4TAs. The logical processor is **not** required to invalidate information that was used for **global** translations (although it may do so). See Section 4.10, "Caching Translation Information" for details regarding global translations. (The instruction may also invalidate mappings associated with other VPIDs.)

  See Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* for details of the INVVPID instruction. See Section 28.3.3.3 for guidelines regarding use of this instruction.
- Execution of the INVEPT instruction invalidates guest-physical mappings and combined mappings. Invalidation is based on instruction operands, called the INVEPT type and the INVEPT descriptor. Two INVEPT types are currently defined:
  - **Single-context.** If the INVEPT type is 1, the logical processor invalidates all guest-physical mappings and combined mappings associated with the EP4TA specified in the INVEPT descriptor. Combined mappings for that EP4TA are invalidated for all VPIDs and all PCIDs. (The instruction may invalidate mappings associated with other EP4TAs.)

  — **All-context.** If the INVEPT type is 2, the logical processor invalidates guest-physical mappings and combined mappings associated with all EP4TAs (and, for combined mappings, for all VPIDs and PCIDs).

  See Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* for details of the INVEPT instruction. See Section 28.3.3.4 for guidelines regarding use of this instruction.

- A power-up or a reset invalidates all linear mappings, guest-physical mappings, and combined mappings.

28.3.3.2 Operations that Need Not Invalidate Cached Mappings

The following items detail cases of operations that are not required to invalidate certain cached mappings:

- Operations that architecturally invalidate entries in the TLBs or paging-structure caches independent of VMX operation are not required to invalidate any guest-physical mappings.
- The INVVPID instruction is not required to invalidate any guest-physical mappings.
- The INVEPT instruction is not required to invalidate any linear mappings.
- VMX transitions are not required to invalidate any guest-physical mappings. If the "enable VPID" VM-execution control is 1, VMX transitions are not required to invalidate any linear mappings or combined mappings.
- The VMXOFF and VMXON instructions are not required to invalidate any linear mappings, guest-physical mappings, or combined mappings.

A logical processor may invalidate any cached mappings at any time. For this reason, the operations identified above may invalidate the indicated mappings despite the fact that doing so is not required.

28.3.3.3 Guidelines for Use of the INVVPID Instruction

The need for VMM software to use the INVVPID instruction depends on how that software is virtualizing memory (e.g., see Section 31.3, "Memory Virtualization").

If EPT is not in use, it is likely that the VMM is virtualizing the guest paging structures. Such a VMM may configure the VMCS so that all or some of the operations that invalidate entries the TLBs and the paging-structure caches (e.g., the INVLPG instruction) cause VM exits. If VMM software is emulating these operations, it may be necessary to use the INVVPID instruction to ensure that the logical processor's TLBs and the paging-structure caches are appropriately invalidated.

Requirements of when software should use the INVVPID instruction depend on the specific algorithm being used for page-table virtualization. The following items provide guidelines for software developers:

- Emulation of the INVLPG instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is individual-address (0).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
  - The linear address in the INVVPID descriptor is that of the operand of the INVLPG instruction being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—except for global translations. An example is the MOV to CR3 instruction. (See Section 4.10, "Caching Translation Information" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A* for details regarding global translations.) Emulation of such an instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context-retaining-globals (3).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.
- Some instructions invalidate all entries in the TLBs and paging-structure caches—including for global translations. An example is the MOV to CR4 instruction if the value of value of bit 4 (page global enable—PGE) is changing. Emulation of such an instruction may require execution of the INVVPID instruction as follows:
  - The INVVPID type is single-context (1).
  - The VPID in the INVVPID descriptor is the one assigned to the virtual processor whose execution is being emulated.

If EPT is not in use, the logical processor associates all mappings it creates with the current VPID, and it will use such mappings to translate linear addresses. For that reason, a VMM should not use the same VPID for different non-EPT guests that use different page tables. Doing so may result in one guest using translations that pertain to the other.

If EPT is in use, the instructions enumerated above might not be configured to cause VM exits and the VMM might not be emulating them. In that case, executions of the instructions by guest software properly invalidate the required entries in the TLBs and paging-structure caches (see Section 28.3.3.1); execution of the INVVPID instruction is not required.

If EPT is in use, the logical processor associates all mappings it creates with the value of bits 51:12 of current EPTP. If a VMM uses different EPTP values for different guests, it may use the same VPID for those guests. Doing so cannot result in one guest using translations that pertain to the other.

The following guidelines apply more generally and are appropriate even if EPT is in use:

- As detailed in Section 25.2.1.1, an access to the APIC-access page might not cause an APIC-access VM exit if software does not properly invalidate information that may be cached from the paging structures. If, at one time, the current VPID on a logical processor was a non-zero value X, it is recommended that software use the INVVPID instruction with the "single-context" INVVPID type and with VPID X in the INVVPID descriptor before a VM entry on the same logical processor that establishes VPID X and either (a) the "virtualize APIC accesses" VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.
- Software can use the INVVPID instruction with the "all-context" INVVPID type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from paging structures between separate uses of VMX operation.

28.3.3.4 Guidelines for Use of the INVEPT Instruction

The following items provide guidelines for use of the INVEPT instruction to invalidate information cached from the EPT paging structures.

- Software should use the INVEPT instruction with the "single-context" INVEPT type after making any of the following changes to an EPT paging-structure entry (the INVEPT descriptor should contain an EPTP value that references — directly or indirectly — the modified EPT paging structure):
  - Changing any of the privilege bits 2:0 from 1 to 0.
  - Changing the physical address in bits 51:12.
  - For an EPT PDPTE or an EPT PDE, changing bit 7 (which determines whether the entry maps a page).
  - For the **last** EPT paging-structure entry used to translate a guest-physical address (either an EPT PDE with bit 7 set to 1 or an EPT PTE), changing either bits 5:3 or bit 6. (These bits determine the effective memory type of accesses using that EPT paging-structure entry; see Section 28.2.4.)
- Software may use the INVEPT instruction after modifying a present EPT paging-structure entry to change any of the privilege bits 2:0 from 0 to 1. Failure to do so may cause an EPT violation that would not otherwise occur. Because an EPT violation invalidates any mappings that would be used by the access that caused the EPT violation (see Section 28.3.3.1), an EPT violation will not recur if the original access is performed again, even if the INVEPT instruction is not executed.
- Because a logical processor does not cache any information derived from EPT paging-structure entries that are not present or misconfigured (see Section 28.2.3.1), it is not necessary to execute INVEPT following modification of an EPT paging-structure entry that had been not present or misconfigured.
- As detailed in Section 25.2.1.1 and Section 25.2.2.1, an access to the APIC-access page might not cause an APIC-access VM exit if software does not

properly invalidate information that may be cached from the EPT paging structures. If EPT was in use on a logical processor at one time with EPTP X, it is recommended that software use the INVEPT instruction with the “single-context” INVEPT type and with EPTP X in the INVEPT descriptor before a VM entry on the same logical processor that enables EPT with EPTP X and either (a) the “virtualize APIC accesses” VM-execution control was changed from 0 to 1; or (b) the value of the APIC-access address was changed.

- Software can use the INVEPT instruction with the “all-context” INVEPT type immediately after execution of the VMXON instruction or immediately prior to execution of the VMXOFF instruction. Either prevents potentially undesired retention of information cached from EPT paging structures between separate uses of VMX operation.

In a system containing more than one logical processor, software must account for the fact that information from an EPT paging-structure entry may be cached on logical processors other than the one that modifies that entry. The process of propagating the changes to a paging-structure entry is commonly referred to as “TLB shootdown.” A discussion of TLB shootdown appears in Section 4.10.5, “Propagation of Paging-Structure Changes to Multiple Processors,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.

CHAPTER 29
SYSTEM MANAGEMENT MODE

This chapter describes aspects of IA-64 and IA-32 architecture used in system management mode (SMM).

SMM provides an alternate operating environment that can be used to monitor and manage various system resources for more efficient energy usage, to control system hardware, and/or to run proprietary code. It was introduced into the IA-32 architecture in the Intel386 SL processor (a mobile specialized version of the Intel386 processor). It is also available in the Pentium M, Pentium 4, Intel Xeon, P6 family, and Pentium and Intel486 processors (beginning with the enhanced versions of the Intel486 SL and Intel486 processors).

29.1 SYSTEM MANAGEMENT MODE OVERVIEW

SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary OEM-designed code. It is intended for use only by system firmware, not by applications software or general-purpose systems software. The main benefit of SMM is that it offers a distinct and easily isolated processor environment that operates transparently to the operating system or executive and software applications.

When SMM is invoked through a system management interrupt (SMI), the processor saves the current state of the processor (the processor's context), then switches to a separate operating environment contained in system management RAM (SMRAM). While in SMM, the processor executes SMI handler code to perform operations such as powering down unused disk drives or monitors, executing proprietary code, or placing the whole system in a suspended state. When the SMI handler has completed its operations, it executes a resume (RSM) instruction. This instruction causes the processor to reload the saved context of the processor, switch back to protected or real mode, and resume executing the interrupted application or operating-system program or task.

The following SMM mechanisms make it transparent to applications programs and operating systems:

- The only way to enter SMM is by means of an SMI.
- The processor executes SMM code in a separate address space (SMRAM) that can be made inaccessible from the other operating modes.
- Upon entering SMM, the processor saves the context of the interrupted program or task.

- All interrupts normally handled by the operating system are disabled upon entry into SMM.
- The RSM instruction can be executed only in SMM.

SMM is similar to real-address mode in that there are no privilege levels or address mapping. An SMM program can address up to 4 GBytes of memory and can execute all I/O and applicable system instructions. See Section 29.5 for more information about the SMM execution environment.

NOTES

Software developers should be aware that, even if a logical processor was using the physical-address extension (PAE) mechanism (introduced in the P6 family processors) or was in IA-32e mode before an SMI, this will not be the case after the SMI is delivered. This is because delivery of an SMI disables paging (see Table 29-4). (This does not apply if the dual-monitor treatment of SMIs and SMM is active; see Section 29.15.)

29.1.1 System Management Mode and VMX Operation

Traditionally, SMM services system management interrupts and then resumes program execution (back to the software stack consisting of executive and application software; see Section 29.2 through Section 29.13).

A virtual machine monitor (VMM) using VMX can act as a host to multiple virtual machines and each virtual machine can support its own software stack of executive and application software. On processors that support VMX, virtual-machine extensions may use system-management interrupts (SMIs) and system-management mode (SMM) in one of two ways:

- **Default treatment.** System firmware handles SMIs. The processor saves architectural states and critical states relevant to VMX operation upon entering SMM. When the firmware completes servicing SMIs, it uses RSM to resume VMX operation.
- **Dual-monitor treatment.** Two VM monitors collaborate to control the servicing of SMIs: one VMM operates outside of SMM to provide basic virtualization in support for guests; the other VMM operates inside SMM (while in VMX operation) to support system-management functions. The former is referred to as **executive monitor**, the latter **SMM-transfer monitor** (**STM**).[1]

The default treatment is described in Section 29.14, "Default Treatment of SMIs and SMM with VMX Operation and SMX Operation". Dual-monitor treatment of SMM is described in Section 29.15, "Dual-Monitor Treatment of SMIs and SMM".

1. The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

29.2 SYSTEM MANAGEMENT INTERRUPT (SMI)

The only way to enter SMM is by signaling an SMI through the SMI# pin on the processor or through an SMI message received through the APIC bus. The SMI is a nonmaskable external interrupt that operates independently from the processor's interrupt- and exception-handling mechanism and the local APIC. The SMI takes precedence over an NMI and a maskable interrupt. SMM is non-reentrant; that is, the SMI is disabled while the processor is in SMM.

NOTES

In the Pentium 4, Intel Xeon, and P6 family processors, when a processor that is designated as an application processor during an MP initialization sequence is waiting for a startup IPI (SIPI), it is in a mode where SMIs are masked. However if a SMI is received while an application processor is in the wait for SIPI mode, the SMI will be pended. The processor then responds on receipt of a SIPI by immediately servicing the pended SMI and going into SMM before handling the SIPI.

An SMI may be blocked for one instruction following execution of STI, MOV to SS, or POP into SS.

29.3 SWITCHING BETWEEN SMM AND THE OTHER PROCESSOR OPERATING MODES

Figure 2-3 shows how the processor moves between SMM and the other processor operating modes (protected, real-address, and virtual-8086). Signaling an SMI while the processor is in real-address, protected, or virtual-8086 modes always causes the processor to switch to SMM. Upon execution of the RSM instruction, the processor always returns to the mode it was in when the SMI occurred.

29.3.1 Entering SMM

The processor always handles an SMI on an architecturally defined "interruptible" point in program execution (which is commonly at an IA-32 architecture instruction boundary). When the processor receives an SMI, it waits for all instructions to retire and for all stores to complete. The processor then saves its current context in SMRAM (see Section 29.4), enters SMM, and begins to execute the SMI handler.

Upon entering SMM, the processor signals external hardware that SMM handling has begun. The signaling mechanism used is implementation dependent. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is asserted each time a bus transaction is generated while the processor is in SMM. For the Pentium and Intel486 processors, the SMIACT# pin is asserted.

An SMI has a greater priority than debug exceptions and external interrupts. Thus, if an NMI, maskable hardware interrupt, or a debug exception occurs at an instruction boundary along with an SMI, only the SMI is handled. Subsequent SMI requests are not acknowledged while the processor is in SMM. The first SMI interrupt request that occurs while the processor is in SMM (that is, after SMM has been acknowledged to external hardware) is latched and serviced when the processor exits SMM with the RSM instruction. The processor will latch only one SMI while in SMM.

See Section 29.5 for a detailed description of the execution environment when in SMM.

29.3.2 Exiting From SMM

The only way to exit SMM is to execute the RSM instruction. The RSM instruction is only available to the SMI handler; if the processor is not in SMM, attempts to execute the RSM instruction result in an invalid-opcode exception (#UD) being generated.

The RSM instruction restores the processor’s context by loading the state save image from SMRAM back into the processor’s registers. The processor then returns an SMIACK transaction on the system bus and returns program control back to the interrupted program.

Upon successful completion of the RSM instruction, the processor signals external hardware that SMM has been exited. For the P6 family processors, an SMI acknowledge transaction is generated on the system bus and the multiplexed status signal EXF4 is no longer generated on bus cycles. For the Pentium and Intel486 processors, the SMIACT# pin is deserted.

If the processor detects invalid state information saved in the SMRAM, it enters the shutdown state and generates a special bus cycle to indicate it has entered shutdown state. Shutdown happens only in the following situations:

- A reserved bit in control register CR4 is set to 1 on a write to CR4. This error should not happen unless SMI handler code modifies reserved areas of the SMRAM saved state map (see Section 29.4.1). CR4 is saved in the state map in a reserved location and cannot be read or modified in its saved state.
- An illegal combination of bits is written to control register CR0, in particular PG set to 1 and PE set to 0, or NW set to 1 and CD set to 0.
- CR4.PCIDE would be set to 1 and IA32_EFER.LMA to 0.
- (For the Pentium and Intel486 processors only.) If the address stored in the SMBASE register when an RSM instruction is executed is not aligned on a 32-KByte boundary. This restriction does not apply to the P6 family processors.

In the shutdown state, Intel processors stop executing instructions until a RESET#, INIT# or NMI# is asserted. While Pentium family processors recognize the SMI# signal in shutdown state, P6 family and Intel486 processors do not. Intel does not support using SMI# to recover from shutdown states for any processor family; the response of processors in this circumstance is not well defined. On Pentium 4 and later processors, shutdown will inhibit INTR and A20M but will not change any of the other inhibits. On these processors, NMIs will be inhibited if no action is taken in the SMM handler to uninhibit them (see Section 29.8).

If the processor is in the HALT state when the SMI is received, the processor handles the return from SMM slightly differently (see Section 29.10). Also, the SMBASE address can be changed on a return from SMM (see Section 29.11).

29.4 SMRAM

While in SMM, the processor executes code and stores data in the SMRAM space. The SMRAM space is mapped to the physical address space of the processor and can be up to 4 GBytes in size. The processor uses this space to save the context of the processor and to store the SMI handler code, data and stack. It can also be used to store system management information (such as the system configuration and specific information about powered-down devices) and OEM-specific information.

The default SMRAM size is 64 KBytes beginning at a base physical address in physical memory called the SMBASE (see Figure 29-1). The SMBASE default value following a hardware reset is 30000H. The processor looks for the first instruction of the SMI handler at the address [SMBASE + 8000H]. It stores the processor's state in the area from [SMBASE + FE00H] to [SMBASE + FFFFH]. See Section 29.4.1 for a description of the mapping of the state save area.

The system logic is minimally required to decode the physical address range for the SMRAM from [SMBASE + 8000H] to [SMBASE + FFFFH]. A larger area can be decoded if needed. The size of this SMRAM can be between 32 KBytes and 4 GBytes.

The location of the SMRAM can be changed by changing the SMBASE value (see Section 29.11). It should be noted that all processors in a multiple-processor system are initialized with the same SMBASE value (30000H). Initialization software must sequentially place each processor in SMM and change its SMBASE so that it does not overlap those of other processors.

The actual physical location of the SMRAM can be in system memory or in a separate RAM memory. The processor generates an SMI acknowledge transaction (P6 family processors) or asserts the SMIACT# pin (Pentium and Intel486 processors) when the processor receives an SMI (see Section 29.3.1).

System logic can use the SMI acknowledge transaction or the assertion of the SMIACT# pin to decode accesses to the SMRAM and redirect them (if desired) to specific SMRAM memory. If a separate RAM memory is used for SMRAM, system logic should provide a programmable method of mapping the SMRAM into system memory space when the processor is not in SMM. This mechanism will enable start-up procedures to initialize the SMRAM space (that is, load the SMI handler) before executing the SMI handler during SMM.

29.4.1 SMRAM State Save Map

When an IA-32 processor that does not support Intel 64 architecture initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area begins at [SMBASE + 8000H + 7FFFH] and extends down to [SMBASE + 8000H + 7E00H]. Table 29-1 shows the state save map. The offset in column 1 is relative to the SMBASE value plus 8000H. Reserved spaces should not be used by software.

Some of the registers in the SMRAM state save area (marked YES in column 3) may be read and changed by the SMI handler, with the changed values restored to the processor registers by the RSM instruction. Some register images are read-only, and must not be modified (modifying these registers will result in unpredictable behavior). An SMI handler should not rely on any values stored in an area that is marked as reserved.

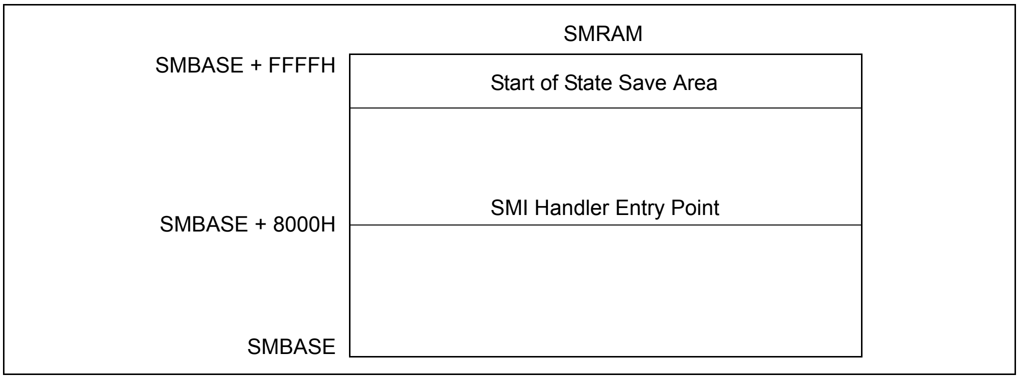

Figure 29-1. SMRAM Usage

Table 29-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes

Table 29-1. SMRAM State Save Map (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR[1]	No
7FC0H	Reserved	No
7FBCH	GS[1]	No
7FB8H	FS[1]	No
7FB4H	DS[1]	No
7FB0H	SS[1]	No
7FACH	CS[1]	No
7FA8H	ES[1]	No
7FA4H	I/O State Field, see Section 29.7	No
7FA0H	I/O Memory Address Field, see Section 29.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

The following registers are saved (but not readable) and restored upon exiting SMM:

- Control register CR4. (This register is cleared to all 0s when entering SMM).
- The hidden segment descriptor information stored in segment registers CS, DS, ES, FS, GS, and SS.

If an SMI request is issued for the purpose of powering down the processor, the values of all reserved locations in the SMM state save must be saved to nonvolatile memory.

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The x87 FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium processors) or test registers TR3 through TR7 (for the Pentium and Intel486 processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The microcode update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor, it must also save and restore them.

NOTES

A small subset of the MSRs (such as, the time-stamp counter and performance-monitoring counters) are not arbitrarily writable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers.

Operating system developers should be aware of this fact and insure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

29.4.1.1 SMRAM State Save Map and Intel 64 Architecture

When the processor initially enters SMM, it writes its state to the state save area of the SMRAM. The state save area on an Intel 64 processor at [SMBASE + 8000H + 7FFFH] and extends to [SMBASE + 8000H + 7C00H].

Support for Intel 64 architecture is reported by CPUID.80000001:EDX[29] = 1. The layout of the SMRAM state save map is shown in Table 29-3.

Additionally, the SMRAM state save map shown in Table 29-3 also applies to processors with the following CPUID signatures listed in Table 29-2, irrespective of the value in CPUID.80000001:EDX[29].

Table 29-2. Processor Signatures and 64-bit SMRAM State Save Map Format

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_17H	Intel Xeon Processor 5200, 5400 series, Intel Core 2 Quad processor Q9xxx, Intel Core 2 Duo processors E8000, T9000,
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad, Intel Core 2 Extreme, Intel Core 2 Duo processors, Intel Pentium dual-core processors
06_1CH	Intel® Atom™ processors

Table 29-3. SMRAM State Save Map for Intel 64 Architecture

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FF8H	CR0	No
7FF0H	CR3	No
7FE8H	RFLAGS	Yes
7FE0H	IA32_EFER	Yes
7FD8H	RIP	Yes
7FD0H	DR6	No
7FC8H	DR7	No
7FC4H	TR SEL[1]	No
7FC0H	LDTR SEL[1]	No
7FBCH	GS SEL[1]	No
7FB8H	FS SEL[1]	No
7FB4H	DS SEL[1]	No
7FB0H	SS SEL[1]	No
7FACH	CS SEL[1]	No
7FA8H	ES SEL[1]	No
7FA4H	IO_MISC	No
7F9CH	IO_MEM_ADDR	No

Table 29-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7F94H	RDI	Yes
7F8CH	RSI	Yes
7F84H	RBP	Yes
7F7CH	RSP	Yes
7F74H	RBX	Yes
7F6CH	RDX	Yes
7F64H	RCX	Yes
7F5CH	RAX	Yes
7F54H	R8	Yes
7F4CH	R9	Yes
7F44H	R10	Yes
7F3CH	R11	Yes
7F34H	R12	Yes
7F2CH	R13	Yes
7F24H	R14	Yes
7F1CH	R15	Yes
7F1BH-7F04H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7EE4H	Reserved	No
7EE0H	Setting of “enable EPT” VM-execution control	No
7ED8H	Value of EPTP VM-execution control field	No
7ED7H - 7EA0H	Reserved	No
7E9CH	LDT Base (lower 32 bits)	No
7E98H	Reserved	No
7E94H	IDT Base (lower 32 bits)	No
7E90H	Reserved	No

Table 29-3. SMRAM State Save Map for Intel 64 Architecture (Contd.)

Offset (Added to SMBASE + 8000H)	Register	Writable?
7E8CH	GDT Base (lower 32 bits)	No
7E8BH - 7E44H	Reserved	No
7E40H	CR4	No
7E3FH - 7DF0H	Reserved	No
7DE8H	IO_EIP	Yes
7DE7H - 7DDCH	Reserved	No
7DD8H	IDT Base (Upper 32 bits)	No
7DD4H	LDT Base (Upper 32 bits)	No
7DD0H	GDT Base (Upper 32 bits)	No
7DCFH - 7C00H	Reserved	No

NOTE:

1. The two most significant bytes are reserved.

29.4.2 SMRAM Caching

An IA-32 processor does not automatically write back and invalidate its caches before entering SMM or before exiting SMM. Because of this behavior, care must be taken in the placement of the SMRAM in system memory and in the caching of the SMRAM to prevent cache incoherence when switching back and forth between SMM and protected mode operation. Either of the following three methods of locating the SMRAM in system memory will guarantee cache coherency:

- Place the SRAM in a dedicated section of system memory that the operating system and applications are prevented from accessing. Here, the SRAM can be designated as cacheable (WB, WT, or WC) for optimum processor performance, without risking cache incoherence when entering or exiting SMM.
- Place the SRAM in a section of memory that overlaps an area used by the operating system (such as the video memory), but designate the SMRAM as uncacheable (UC). This method prevents cache access when in SMM to maintain cache coherency, but the use of uncacheable memory reduces the performance of SMM code.
- Place the SRAM in a section of system memory that overlaps an area used by the operating system and/or application code, but explicitly flush (write back and invalidate) the caches upon entering and exiting SMM mode. This method maintains cache coherency, but the incurs the overhead of two complete cache flushes.

For Pentium 4, Intel Xeon, and P6 family processors, a combination of the first two methods of locating the SMRAM is recommended. Here the SMRAM is split between an overlapping and a dedicated region of memory. Upon entering SMM, the SMRAM space that is accessed overlaps video memory (typically located in low memory). This SMRAM section is designated as UC memory. The initial SMM code then jumps to a second SMRAM section that is located in a dedicated region of system memory (typically in high memory). This SMRAM section can be cached for optimum processor performance.

For systems that explicitly flush the caches upon entering SMM (the third method described above), the cache flush can be accomplished by asserting the FLUSH# pin at the same time as the request to enter SMM (generally initiated by asserting the SMI# pin). The priorities of the FLUSH# and SMI# pins are such that the FLUSH# is serviced first. To guarantee this behavior, the processor requires that the following constraints on the interaction of FLUSH# and SMI# be met. In a system where the FLUSH# and SMI# pins are synchronous and the set up and hold times are met, then the FLUSH# and SMI# pins may be asserted in the same clock. In asynchronous systems, the FLUSH# pin must be asserted at least one clock before the SMI# pin to guarantee that the FLUSH# pin is serviced first.

Upon leaving SMM (for systems that explicitly flush the caches), the WBINVD instruction should be executed prior to leaving SMM to flush the caches.

NOTES

In systems based on the Pentium processor that use the FLUSH# pin to write back and invalidate cache contents before entering SMM, the processor will prefetch at least one cache line in between when the Flush Acknowledge cycle is run and the subsequent recognition of SMI# and the assertion of SMIACT#.

It is the obligation of the system to ensure that these lines are not cached by returning KEN# inactive to the Pentium processor.

29.4.2.1 System Management Range Registers (SMRR)

SMI handler code and data stored by SMM code resides in SMRAM. The SMRR interface is an enhancement in Intel 64 architecture to limit cacheable reference of addresses in SMRAM to code running in SMM. The SMRR interface can be configured only by code running in SMM. Details of SMRR is described in Section 11.11.2.4.

29.5 SMI HANDLER EXECUTION ENVIRONMENT

After saving the current context of the processor, the processor initializes its core registers to the values shown in Table 29-4. Upon entering SMM, the PE and PG flags in control register CR0 are cleared, which places the processor is in an environment similar to real-address mode. The differences between the SMM execution environment and the real-address mode execution environment are as follows:

- The addressable SMRAM address space ranges from 0 to FFFFFFFFH (4 GBytes). (The physical address extension — enabled with the PAE flag in control register CR4 — is not supported in SMM.)
- The normal 64-KByte segment limit for real-address mode is increased to 4 GBytes.
- The default operand and address sizes are set to 16 bits, which restricts the addressable SMRAM address space to the 1-MByte real-address mode limit for native real-address-mode code. However, operand-size and address-size override prefixes can be used to access the address space beyond the 1-MByte.

Table 29-4. Processor Register Initialization in SMM

Register	Contents
General-purpose registers	Undefined
EFLAGS	00000002H
EIP	00008000H
CS selector	SMM Base shifted right 4 bits (default 3000H)
CS base	SMM Base (default 30000H)
DS, ES, FS, GS, SS Selectors	0000H
DS, ES, FS, GS, SS Bases	000000000H
DS, ES, FS, GS, SS Limits	0FFFFFFFFH
CR0	PE, EM, TS, and PG flags set to 0; others unmodified
CR4	Cleared to zero
DR6	Undefined
DR7	00000400H

- Near jumps and calls can be made to anywhere in the 4-GByte address space if a 32-bit operand-size override prefix is used. Due to the real-address-mode style of base-address formation, a far call or jump cannot transfer control to a segment with a base address of more than 20 bits (1 MByte). However, since the segment limit in SMM is 4 GBytes, offsets into a segment that go beyond the 1-MByte limit are allowed when using 32-bit operand-size override prefixes. Any program control transfer that does not have a 32-bit operand-size override prefix truncates the EIP value to the 16 low-order bits.
- Data and the stack can be located anywhere in the 4-GByte address space, but can be accessed only with a 32-bit address-size override if they are located above 1 MByte. As with the code segment, the base address for a data or stack segment cannot be more than 20 bits.

The value in segment register CS is automatically set to the default of 30000H for the SMBASE shifted 4 bits to the right; that is, 3000H. The EIP register is set to 8000H. When the EIP value is added to shifted CS value (the SMBASE), the resulting linear address points to the first instruction of the SMI handler.

The other segment registers (DS, SS, ES, FS, and GS) are cleared to 0 and their segment limits are set to 4 GBytes. In this state, the SMRAM address space may be treated as a single flat 4-GByte linear address space. If a segment register is loaded with a 16-bit value, that value is then shifted left by 4 bits and loaded into the segment base (hidden part of the segment register). The limits and attributes are not modified.

Maskable hardware interrupts, exceptions, NMI interrupts, SMI interrupts, A20M interrupts, single-step traps, breakpoint traps, and INIT operations are inhibited when the processor enters SMM. Maskable hardware interrupts, exceptions, single-step traps, and breakpoint traps can be enabled in SMM if the SMM execution environment provides and initializes an interrupt table and the necessary interrupt and exception handlers (see Section 29.6).

29.6 EXCEPTIONS AND INTERRUPTS WITHIN SMM

When the processor enters SMM, all hardware interrupts are disabled in the following manner:

- The IF flag in the EFLAGS register is cleared, which inhibits maskable hardware interrupts from being generated.
- The TF flag in the EFLAGS register is cleared, which disables single-step traps.
- Debug register DR7 is cleared, which disables breakpoint traps. (This action prevents a debugger from accidentally breaking into an SMM handler if a debug breakpoint is set in normal address space that overlays code or data in SMRAM.)
- NMI, SMI, and A20M interrupts are blocked by internal SMM logic. (See Section 29.8 for more information about how NMIs are handled in SMM.)

Software-invoked interrupts and exceptions can still occur, and maskable hardware interrupts can be enabled by setting the IF flag. Intel recommends that SMM code be written in so that it does not invoke software interrupts (with the INT *n*, INTO, INT 3, or BOUND instructions) or generate exceptions.

If the SMM handler requires interrupt and exception handling, an SMM interrupt table and the necessary exception and interrupt handlers must be created and initialized from within SMM. Until the interrupt table is correctly initialized (using the LIDT instruction), exceptions and software interrupts will result in unpredictable processor behavior.

The following restrictions apply when designing SMM interrupt and exception-handling facilities:

- The interrupt table should be located at linear address 0 and must contain real-address mode style interrupt vectors (4 bytes containing CS and IP).
- Due to the real-address mode style of base address formation, an interrupt or exception cannot transfer control to a segment with a base address of more that 20 bits.

- An interrupt or exception cannot transfer control to a segment offset of more than 16 bits (64 KBytes).
- When an exception or interrupt occurs, only the 16 least-significant bits of the return address (EIP) are pushed onto the stack. If the offset of the interrupted procedure is greater than 64 KBytes, it is not possible for the interrupt/exception handler to return control to that procedure. (One solution to this problem is for a handler to adjust the return address on the stack.)
- The SMBASE relocation feature affects the way the processor will return from an interrupt or exception generated while the SMI handler is executing. For example, if the SMBASE is relocated to above 1 MByte, but the exception handlers are below 1 MByte, a normal return to the SMI handler is not possible. One solution is to provide the exception handler with a mechanism for calculating a return address above 1 MByte from the 16-bit return address on the stack, then use a 32-bit far call to return to the interrupted procedure.
- If an SMI handler needs access to the debug trap facilities, it must insure that an SMM accessible debug handler is available and save the current contents of debug registers DR0 through DR3 (for later restoration). Debug registers DR0 through DR3 and DR7 must then be initialized with the appropriate values.
- If an SMI handler needs access to the single-step mechanism, it must insure that an SMM accessible single-step handler is available, and then set the TF flag in the EFLAGS register.
- If the SMI design requires the processor to respond to maskable hardware interrupts or software-generated interrupts while in SMM, it must ensure that SMM accessible interrupt handlers are available and then set the IF flag in the EFLAGS register (using the STI instruction). Software interrupts are not blocked upon entry to SMM, so they do not need to be enabled.

29.7 MANAGING SYNCHRONOUS AND ASYNCHRONOUS SYSTEM MANAGEMENT INTERRUPTS

When coding for a multiprocessor system or a system with Intel HT Technology, it was not always possible for an SMI handler to distinguish between a synchronous SMI (triggered during an I/O instruction) and an asynchronous SMI. To facilitate the discrimination of these two events, incremental state information has been added to the SMM state save map.

Processors that have an SMM revision ID of 30004H or higher have the incremental state information described below.

29.7.1 I/O State Implementation

Within the extended SMM state save map, a bit (IO_SMI) is provided that is set only when an SMI is either taken immediately after a *successful* I/O instruction or is taken after a *successful* iteration of a REP I/O instruction (the *successful* notion pertains to the processor point of view; not necessarily to the corresponding platform function). When set, the IO_SMI bit provides a strong indication that the corresponding SMI was synchronous. In this case, the SMM State Save Map also supplies the port address of the I/O operation. The IO_SMI bit and the I/O Port Address may be used in conjunction with the information logged by the platform to confirm that the SMI was indeed synchronous.

The IO_SMI bit by itself is a strong indication, not a guarantee, that the SMI is synchronous. This is because an asynchronous SMI might coincidentally be taken after an I/O instruction. In such a case, the IO_SMI bit would still be set in the SMM state save map.

Information characterizing the I/O instruction is saved in two locations in the SMM State Save Map (Table 29-5). The IO_SMI bit also serves as a valid bit for the rest of the I/O information fields. The contents of these I/O information fields are not defined when the IO_SMI bit is not set.

Table 29-5. I/O Instruction Information in the SMM State Save Map

State (SMM Rev. ID: 30004H or higher)	Format				
	31 16	15 8	7 4	3 1	0
I/O State Field SMRAM offset 7FA4	I/O Port	Reserved	I/O Type	I/O Length	IO_SMI
	31				0
I/O Memory Address Field SMRAM offset 7FA0	I/O Memory Address				

When IO_SMI is set, the other fields may be interpreted as follows:

- I/O length:
    - 001 – Byte
    - 010 – Word
    - 100 – Dword
- I/O instruction type (Table 29-6)

Table 29-6. I/O Instruction Type Encodings

Instruction	Encoding
IN Immediate	1001
IN DX	0001
OUT Immediate	1000

Table 29-6. I/O Instruction Type Encodings (Contd.)

Instruction	Encoding
OUT DX	0000
INS	0011
OUTS	0010
REP INS	0111
REP OUTS	0110

29.8 NMI HANDLING WHILE IN SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence. This assumes that NMIs were not blocked before the SMI occurred. If NMIs were blocked before the SMI occurred, they are blocked after execution of RSM.

Although NMI requests are blocked when the processor enters SMM, they may be enabled through software by executing an IRET instruction. If the SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET instruction. Once an IRET instruction is executed, NMI interrupt requests are serviced in the same “real mode” manner in which they are handled outside of SMM.

A special case can occur if an SMI handler nests inside an NMI handler and then another NMI occurs. During NMI interrupt handling, NMI interrupts are disabled, so normally NMI interrupts are serviced and completed with an IRET instruction one at a time. When the processor enters SMM while executing an NMI handler, the processor saves the SMRAM state save map but does not save the attribute to keep NMI interrupts disabled. Potentially, an NMI could be latched (while in SMM or upon exit) and serviced upon exit of SMM even though the previous NMI handler has still not completed. One or more NMIs could thus be nested inside the first NMI handler. The NMI interrupt handler should take this possibility into consideration.

Also, for the Pentium processor, exceptions that invoke a trap or fault handler will enable NMI interrupts from inside of SMM. This behavior is implementation specific for the Pentium processor and is not part of the IA-32 architecture.

29.9 SMM REVISION IDENTIFIER

The SMM revision identifier field is used to indicate the version of SMM and the SMM extensions that are supported by the processor (see Figure 29-2). The SMM revision identifier is written during SMM entry and can be examined in SMRAM space at offset

7EFCH. The lower word of the SMM revision identifier refers to the version of the base SMM architecture.

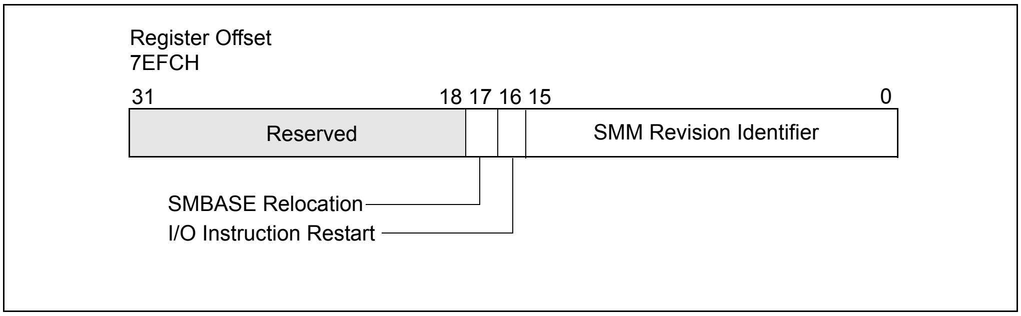

Figure 29-2. SMM Revision Identifier

The upper word of the SMM revision identifier refers to the extensions available. If the I/O instruction restart flag (bit 16) is set, the processor supports the I/O instruction restart (see Section 29.12); if the SMBASE relocation flag (bit 17) is set, SMRAM base address relocation is supported (see Section 29.11).

29.10 AUTO HALT RESTART

If the processor is in a HALT state (due to the prior execution of a HLT instruction) when it receives an SMI, the processor records the fact in the auto HALT restart flag in the saved processor state (see Figure 29-3). (This flag is located at offset 7F02H and bit 0 in the state save area of the SMRAM.)

If the processor sets the auto HALT restart flag upon entering SMM (indicating that the SMI occurred when the processor was in the HALT state), the SMI handler has two options:

- It can leave the auto HALT restart flag set, which instructs the RSM instruction to return program control to the HLT instruction. This option in effect causes the processor to re-enter the HALT state after handling the SMI. (This is the default operation.)
- It can clear the auto HALT restart flag, with instructs the RSM instruction to return program control to the instruction following the HLT instruction.

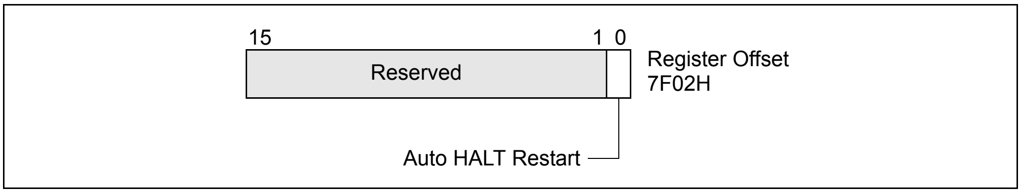

Figure 29-3. Auto HALT Restart Field

These options are summarized in Table 29-7. If the processor was not in a HALT state when the SMI was received (the auto HALT restart flag is cleared), setting the flag to 1 will cause unpredictable behavior when the RSM instruction is executed.

Table 29-7. Auto HALT Restart Flag Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
0	0	Returns to next instruction in interrupted program or task.
0	1	Unpredictable.
1	0	Returns to next instruction after HLT instruction.
1	1	Returns to HALT state.

If the HLT instruction is restarted, the processor will generate a memory access to fetch the HLT instruction (if it is not in the internal cache), and execute a HLT bus transaction. This behavior results in multiple HLT bus transactions for the same HLT instruction.

29.10.1 Executing the HLT Instruction in SMM

The HLT instruction should not be executed during SMM, unless interrupts have been enabled by setting the IF flag in the EFLAGS register. If the processor is halted in SMM, the only event that can remove the processor from this state is a maskable hardware interrupt or a hardware reset.

29.11 SMBASE RELOCATION

The default base address for the SMRAM is 30000H. This value is contained in an internal processor register called the SMBASE register. The operating system or executive can relocate the SMRAM by setting the SMBASE field in the saved state map (at offset 7EF8H) to a new value (see Figure 29-4). The RSM instruction reloads the internal SMBASE register with the value in the SMBASE field each time it exits SMM. All subsequent SMI requests will use the new SMBASE value to find the starting address for the SMI handler (at SMBASE + 8000H) and the SMRAM state save area (from SMBASE + FE00H to SMBASE + FFFFH). (The processor resets the value in its internal SMBASE register to 30000H on a RESET, but does not change it on an INIT.)

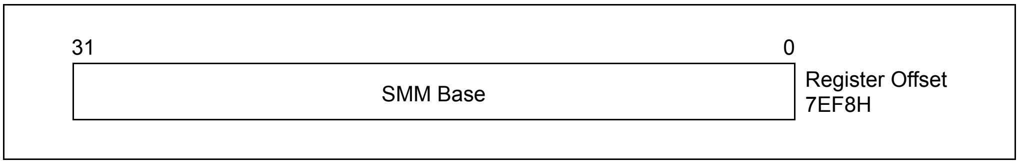

Figure 29-4. SMBASE Relocation Field

In multiple-processor systems, initialization software must adjust the SMBASE value for each processor so that the SMRAM state save areas for each processor do not overlap. (For Pentium and Intel486 processors, the SMBASE values must be aligned on a 32-KByte boundary or the processor will enter shutdown state during the execution of a RSM instruction.)

If the SMBASE relocation flag in the SMM revision identifier field is set, it indicates the ability to relocate the SMBASE (see Section 29.9).

29.11.1 Relocating SMRAM to an Address Above 1 MByte

In SMM, the segment base registers can only be updated by changing the value in the segment registers. The segment registers contain only 16 bits, which allows only 20 bits to be used for a segment base address (the segment register is shifted left 4 bits to determine the segment base address). If SMRAM is relocated to an address above 1 MByte, software operating in real-address mode can no longer initialize the segment registers to point to the SMRAM base address (SMBASE).

The SMRAM can still be accessed by using 32-bit address-size override prefixes to generate an offset to the correct address. For example, if the SMBASE has been relocated to FFFFFFH (immediately below the 16-MByte boundary) and the DS, ES, FS, and GS registers are still initialized to 0H, data in SMRAM can be accessed by using 32-bit displacement registers, as in the following example:

```
mov    esi,00FFxxxxH; 64K segment immediately below 16M
mov    ax,ds:[esi]
```

A stack located above the 1-MByte boundary can be accessed in the same manner.

29.12 I/O INSTRUCTION RESTART

If the I/O instruction restart flag in the SMM revision identifier field is set (see Section 29.9), the I/O instruction restart mechanism is present on the processor. This mechanism allows an interrupted I/O instruction to be re-executed upon returning from

SMM mode. For example, if an I/O instruction is used to access a powered-down I/O device, a chip set supporting this device can intercept the access and respond by asserting SMI#. This action invokes the SMI handler to power-up the device. Upon returning from the SMI handler, the I/O instruction restart mechanism can be used to re-execute the I/O instruction that caused the SMI.

The I/O instruction restart field (at offset 7F00H in the SMM state-save area, see Figure 29-5) controls I/O instruction restart. When an RSM instruction is executed, if this field contains the value FFH, then the EIP register is modified to point to the I/O instruction that received the SMI request. The processor will then automatically re-execute the I/O instruction that the SMI trapped. (The processor saves the necessary machine state to insure that re-execution of the instruction is handled coherently.)

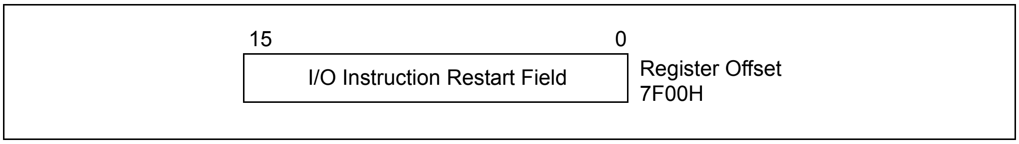

Figure 29-5. I/O Instruction Restart Field

If the I/O instruction restart field contains the value 00H when the RSM instruction is executed, then the processor begins program execution with the instruction following the I/O instruction. (When a repeat prefix is being used, the next instruction may be the next I/O instruction in the repeat loop.) Not re-executing the interrupted I/O instruction is the default behavior; the processor automatically initializes the I/O instruction restart field to 00H upon entering SMM. Table 29-8 summarizes the states of the I/O instruction restart field.

Table 29-8. I/O Instruction Restart Field Values

Value of Flag After Entry to SMM	Value of Flag When Exiting SMM	Action of Processor When Exiting SMM
00H	00H	Does not re-execute trapped I/O instruction.
00H	FFH	Re-executes trapped I/O instruction.

The I/O instruction restart mechanism does not indicate the cause of the SMI. It is the responsibility of the SMI handler to examine the state of the processor to determine the cause of the SMI and to determine if an I/O instruction was interrupted and should be restarted upon exiting SMM. If an SMI interrupt is signaled on a non-I/O instruction boundary, setting the I/O instruction restart field to FFH prior to executing the RSM instruction will likely result in a program error.

29.12.1 Back-to-Back SMI Interrupts When I/O Instruction Restart Is Being Used

If an SMI interrupt is signaled while the processor is servicing an SMI interrupt that occurred on an I/O instruction boundary, the processor will service the new SMI request before restarting the originally interrupted I/O instruction. If the I/O instruction restart field is set to FFH prior to returning from the second SMI handler, the EIP will point to an address different from the originally interrupted I/O instruction, which will likely lead to a program error. To avoid this situation, the SMI handler must be able to recognize the occurrence of back-to-back SMI interrupts when I/O instruction restart is being used and insure that the handler sets the I/O instruction restart field to 00H prior to returning from the second invocation of the SMI handler.

29.13 SMM MULTIPLE-PROCESSOR CONSIDERATIONS

The following should be noted when designing multiple-processor systems:

- Any processor in a multiprocessor system can respond to an SMM.
- Each processor needs its own SMRAM space. This space can be in system memory or in a separate RAM.
- The SMRAMs for different processors can be overlapped in the same memory space. The only stipulation is that each processor needs its own state save area and its own dynamic data storage area. (Also, for the Pentium and Intel486 processors, the SMBASE address must be located on a 32-KByte boundary.) Code and static data can be shared among processors. Overlapping SMRAM spaces can be done more efficiently with the P6 family processors because they do not require that the SMBASE address be on a 32-KByte boundary.
- The SMI handler will need to initialize the SMBASE for each processor.
- Processors can respond to local SMIs through their SMI# pins or to SMIs received through the APIC interface. The APIC interface can distribute SMIs to different processors.
- Two or more processors can be executing in SMM at the same time.
- When operating Pentium processors in dual processing (DP) mode, the SMIACT# pin is driven only by the MRM processor and should be sampled with ADS#. For additional details, see Chapter 14 of the *Pentium Processor Family User's Manual, Volume 1*.

SMM is not re-entrant, because the SMRAM State Save Map is fixed relative to the SMBASE. If there is a need to support two or more processors in SMM mode at the same time then each processor should have dedicated SMRAM spaces. This can be done by using the SMBASE Relocation feature (see Section 29.11).

29.14 DEFAULT TREATMENT OF SMIS AND SMM WITH VMX OPERATION AND SMX OPERATION

Under the default treatment, the interactions of SMIs and SMM with VMX operation are few. This section details those interactions. It also explains how this treatment affects SMX operation.

29.14.1 Default Treatment of SMI Delivery

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on architectural definitions. Under the default treatment, processors that support VMX operation perform SMI delivery as follows:

```
enter SMM;
save the following internal to the processor:
    CR4.VMXE
    an indication of whether the logical processor was in VMX operation (root or non-root)
IF the logical processor is in VMX operation
    THEN
        save current VMCS pointer internal to the processor;
        leave VMX operation;
        save VMX-critical state defined below;
FI;
IF the logical processor supports SMX operation
    THEN
        save internal to the logical processor an indication of whether the Intel® TXT private space
is locked;
        IF the TXT private space is unlocked
            THEN lock the TXT private space;
        FI;
FI;
CR4.VMXE ← 0;
perform ordinary SMI delivery:
    save processor state in SMRAM;
    set processor state to standard SMM values;[1]
    invalidate linear mappings and combined mappings associated with VPID 0000H (for all PCIDs);
combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits
51:12 of EPTP; see Section 28.3);
```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state consists of the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM[2]; (3) the state of blocking by STI and by MOV SS (see Table 24-3 in

1. This causes the logical processor to block INIT signals, NMIs, and SMIs.

Section 24.4.2); (4) the state of virtual-NMI blocking (only if the processor is in VMX non-root operation and the "virtual NMIs" VM-execution control is 1); and (5) an indication of whether an MTF VM exit is pending (see Section 25.7.2). These data may be saved internal to the processor or in the VMCS region of the current VMCS. Processors that do not support SMI recognition while there is blocking by STI or by MOV SS need not save the state of such blocking.

If the logical processor supports the 1-setting of the "enable EPT" VM-execution control and the logical processor was in VMX non-root operation at the time of an SMI, it saves the value of that control into bit 0 of the 32-bit field at offset SMBASE + 8000H + 7EE0H (SMBASE + FEE0H; see Table 29-3).[1] If the logical processor was not in VMX non-root operation at the time of the SMI, it saves 0 into that bit. If the logical processor saves 1 into that bit (it was in VMX non-root operation and the "enable EPT" VM-execution control was 1), it saves the value of the EPT pointer (EPTP) into the 64-bit field at offset SMBASE + 8000H + 7ED8H (SMBASE + FED8H).

Because SMI delivery causes a logical processor to leave VMX operation, all the controls associated with VMX non-root operation are disabled in SMM and thus cannot cause VM exits while the logical processor in SMM.

29.14.2 Default Treatment of RSM

Ordinary execution of RSM restores processor state from SMRAM. Under the default treatment, processors that support VMX operation perform RSM as follows:

```
IF VMXE = 1 in CR4 image in SMRAM
   THEN fail and enter shutdown state;
   ELSE
       restore state normally from SMRAM;
       invalidate linear mappings and combined mappings associated with all VPIDs and all PCIDs;
combined mappings are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP;
see Section 28.3);
       IF the logical processor supports SMX operation andthe Intel® TXT private space was
unlocked at the time of the last SMI (as saved)
           THEN unlock the TXT private space;
       FI;
       CR4.VMXE ← value stored internally;
```

2. Section 29.14 and Section 29.15 use the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of these registers (EAX, EIP, ESP, EFLAGS, etc.). In a few places, notation such as EAX is used to refer specifically to the lower 32 bits of the register.

1. "Enable EPT" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, SMI functions as the "enable EPT" VM-execution control were 0. See Section 24.6.2.

```
        IF internal storage indicates that the logical processor
        had been in VMX operation (root or non-root)
            THEN
                enter VMX operation (root or non-root);
                restore VMX-critical state as defined in Section 29.14.1;
                set to their fixed values any bits in CR0 and CR4 whose values must be fixed in
VMX operation (see Section 23.8);[1]
                IF RFLAGS.VM = 0 AND (in VMX root operation OR the "unrestricted guest" VM-
execution control is 0)[2]
                    THEN
                        CS.RPL ← SS.DPL;
                        SS.RPL ← SS.DPL;
                FI;
                restore current VMCS pointer;
        FI;
        leave SMM;
        IF logical processor will be in VMX operation or in SMX operation after RSM
            THEN block A20M and leave A20M mode;
        FI;
FI;
```

RSM unblocks SMIs. It restores the state of blocking by NMI (see Table 24-3 in Section 24.4.2) as follows:

- If the RSM is not to VMX non-root operation or if the "virtual NMIs" VM-execution control will be 0, the state of NMI blocking is restored normally.
- If the RSM is to VMX non-root operation and the "virtual NMIs" VM-execution control will be 1, NMIs are not blocked after RSM. The state of virtual-NMI blocking is restored as part of VMX-critical state.

INIT signals are blocked after RSM if and only if the logical processor will be in VMX root operation.

If RSM returns a logical processor to VMX non-root operation, it re-establishes the controls associated with the current VMCS. If the "interrupt-window exiting" VM-execution control is 1, a VM exit occurs immediately after RSM if the enabling conditions apply. The same is true for the "NMI-window exiting" VM-execution control. Such VM exits occur with their normal priority. See Section 25.3.

1. If the RSM is to VMX non-root operation and both the "unrestricted guest" VM-execution control and bit 31 of the primary processor-based VM-execution controls will be 1, CR0.PE and CR0.PG retain the values that were loaded from SMRAM regardless of what is reported in the capability MSR IA32_VMX_CR0_FIXED0.

2. "Unrestricted guest" is a secondary processor-based VM-execution control. If bit 31 of the primary processor-based VM-execution controls is 0, VM entry functions as if the "unrestricted guest" VM-execution control were 0. See Section 24.6.2.

If an MTF VM exit was pending at the time of the previous SMI, an MTF VM exit is pending on the instruction boundary following execution of RSM. The following items detail the treatment of MTF VM exits that may be pending following RSM:

- System-management interrupts (SMIs), INIT signals, and higher priority events take priority over these MTF VM exits. These MTF VM exits take priority over debug-trap exceptions and lower priority events.
- These MTF VM exits wake the logical processor if RSM caused the logical processor to enter the HLT state (see Section 29.10). They do not occur if the logical processor just entered the shutdown state.

29.14.3 Protection of CR4.VMXE in SMM

Under the default treatment, CR4.VMXE is treated as a reserved bit while a logical processor is in SMM. Any attempt by software running in SMM to set this bit causes a general-protection exception. In addition, software cannot use VMX instructions or enter VMX operation while in SMM.

29.14.4 VMXOFF and SMI Unblocking

The VMXOFF instruction can be executed only with the default treatment (see Section 29.15.1) and only outside SMM. If SMIs are blocked when VMXOFF is executed, VMXOFF unblocks them unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 29.15.5 for details regarding this MSR).[1] Section 29.15.7 identifies a case in which SMIs may be blocked when VMXOFF is executed.

Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine whether this is allowed.

29.15 DUAL-MONITOR TREATMENT OF SMIs AND SMM

Dual-monitor treatment is activated through the cooperation of the **executive monitor** (the VMM that operates outside of SMM to provide basic virtualization) and the **SMM-transfer monitor** (**STM**; the VMM that operates inside SMM—while in VMX operation—to support system-management functions). Control is transferred to the STM through VM exits; VM entries are used to return from SMM.

The dual-monitor treatment may not be supported by all processors. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether it is supported.

1. Setting IA32_SMM_MONITOR_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register's valid bit (bit 0).

29.15.1 Dual-Monitor Treatment Overview

The dual-monitor treatment uses an executive monitor and an SMM-transfer monitor (STM). Transitions from the executive monitor or its guests to the STM are called **SMM VM exits** and are discussed in Section 29.15.2. SMM VM exits are caused by SMIs as well as executions of VMCALL in VMX root operation. The latter allow the executive monitor to call the STM for service.

The STM runs in VMX root operation and uses VMX instructions to establish a VMCS and perform VM entries to its own guests. This is done all inside SMM (see Section 29.15.3). The STM returns from SMM, not by using the RSM instruction, but by using a VM entry that returns from SMM. Such VM entries are described in Section 29.15.4.

Initially, there is no STM and the default treatment (Section 29.14) is used. The dual-monitor treatment is not used until it is enabled and activated. The steps to do this are described in Section 29.15.5 and Section 29.15.6.

It is not possible to leave VMX operation under the dual-monitor treatment; VMXOFF will fail if executed. The dual-monitor treatment must be deactivated first. The STM deactivates dual-monitor treatment using a VM entry that returns from SMM with the “deactivate dual-monitor treatment” VM-entry control set to 1 (see Section 29.15.7).

The executive monitor configures any VMCS that it uses for VM exits to the executive monitor. SMM VM exits, which transfer control to the STM, use a different VMCS. Under the dual-monitor treatment, each logical processor uses a separate VMCS called the **SMM-transfer VMCS**. When the dual-monitor treatment is active, the logical processor maintains another VMCS pointer called the **SMM-transfer VMCS pointer**. The SMM-transfer VMCS pointer is established when the dual-monitor treatment is activated.

29.15.2 SMM VM Exits

An SMM VM exit is a VM exit that begins outside SMM and that ends in SMM.

Unlike other VM exits, SMM VM exits can begin in VMX root operation. SMM VM exits result from the arrival of an SMI outside SMM or from execution of VMCALL in VMX root operation outside SMM. Execution of VMCALL in VMX root operation causes an SMM VM exit only if the valid bit is set in the IA32_SMM_MONITOR_CTL MSR (see Section 29.15.5).

Execution of VMCALL in VMX root operation causes an SMM VM exit even under the default treatment. This SMM VM exit activates the dual-monitor treatment (see Section 29.15.6).

Differences between SMM VM exits and other VM exits are detailed in Sections 29.15.2.1 through 29.15.2.5. Differences between SMM VM exits that activate the dual-monitor treatment and other SMM VM exits are described in Section 29.15.6.

29.15.2.1 Architectural State Before a VM Exit

System-management interrupts (SMIs) that cause SMM VM exits always do so directly. They do not save state to SMRAM as they do under the default treatment.

29.15.2.2 Updating the Current-VMCS and Executive-VMCS Pointers

SMM VM exits begin by performing the following steps:

1. The executive-VMCS pointer field in the SMM-transfer VMCS is loaded as follows:
   - If the SMM VM exit commenced in VMX non-root operation, it receives the current-VMCS pointer.
   - If the SMM VM exit commenced in VMX root operation, it receives the VMXON pointer.
2. The current-VMCS pointer is loaded with the value of the SMM-transfer VMCS pointer.

The last step ensures that the current VMCS is the SMM-transfer VMCS. VM-exit information is recorded in that VMCS, and VM-entry control fields in that VMCS are updated. State is saved into the guest-state area of that VMCS. The VM-exit controls and host-state area of that VMCS determine how the VM exit operates.

29.15.2.3 Recording VM-Exit Information

SMM VM exits differ from other VM exit with regard to the way they record VM-exit information. The differences follow.

- **Exit reason**.
  - Bits 15:0 of this field contain the basic exit reason. The field is loaded with the reason for the SMM VM exit: I/O SMI (an SMI arrived immediately after retirement of an I/O instruction), other SMI, or VMCALL. See Appendix C, “VMX Basic Exit Reasons”.
  - SMM VM exits are the only VM exits that may occur in VMX root operation. Because the SMM-transfer monitor may need to know whether it was invoked from VMX root or VMX non-root operation, this information is stored in bit 29 of the exit-reason field (see Table 24-14 in Section 24.9.1). The bit is set by SMM VM exits from VMX root operation.
  - If the SMM VM exit occurred in VMX non-root operation and an MTF VM exit was pending, bit 28 of the exit-reason field is set; otherwise, it is cleared.
  - Bits 27:16 and bits 31:30 are cleared.
- **Exit qualification.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, the exit qualification contains information about the I/O instruction that retired immediately before the SMI.It has the format given in Table 29-9.

Table 29-9. Exit Qualification for SMIs That Arrive Immediately After the Retirement of an I/O Instruction

Bit Position(s)	Contents
2:0	Size of access: 0 = 1-byte 1 = 2-byte 3 = 4-byte Other values not used.
3	Direction of the attempted access (0 = OUT, 1 = IN)
4	String instruction (0 = not string; 1 = string)
5	REP prefixed (0 = not REP; 1 = REP)
6	Operand encoding (0 = DX, 1 = immediate)
15:7	Reserved (cleared to 0)
31:16	Port number (as specified in the I/O instruction)
63:32	Reserved (cleared to 0). These bits exist only on processors that support Intel 64 architecture.

- **Guest linear address.** This field is used for VM exits due to SMIs that arrive immediately after the retirement of an INS or OUTS instruction for which the relevant segment (ES for INS; DS for OUTS unless overridden by an instruction prefix) is usable. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS; the default segment is DS but can be overridden by a segment override prefix) at the time the instruction started. If the relevant segment is not usable, the value is undefined. On processors that support Intel 64 architecture, bits 63:32 are clear if the logical processor was not in 64-bit mode before the VM exit.
- **I/O RCX, I/O RSI, I/O RDI, and I/O RIP.** For an SMM VM exit due an SMI that arrives immediately after the retirement of an I/O instruction, these fields receive the values that were in RCX, RSI, RDI, and RIP, respectively, before the I/O instruction executed. Thus, the value saved for I/O RIP addresses the I/O instruction.

29.15.2.4 Saving Guest State

SMM VM exits save the contents of the SMBASE register into the corresponding field in the guest-state area.

The value of the VMX-preemption timer is saved into the corresponding field in the guest-state area if the "save VMX-preemption timer value" VM-exit control is 1. That field becomes undefined if, in addition, either the SMM VM exit is from VMX root operation or the SMM VM exit is from VMX non-root operation and the "activate VMX-preemption timer" VM-execution control is 0.

29.15.2.5 Updating Non-Register State

SMM VM exits affect the non-register state of a logical processor as follows:

- SMM VM exits cause non-maskable interrupts (NMIs) to be blocked; they may be unblocked through execution of IRET or through a VM entry (depending on the value loaded for the interruptibility state and the setting of the "virtual NMIs" VM-execution control).
- SMM VM exits cause SMIs to be blocked; they may be unblocked by a VM entry that returns from SMM (see Section 29.15.4).

SMM VM exits invalidate linear mappings and combined mappings associated with VPID 0000H for all PCIDs. Combined mappings for VPID 0000H are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 28.3). (Ordinary VM exits are not required to perform such invalidation if the "enable VPID" VM-execution control is 1; see Section 27.5.5.)

29.15.3 Operation of the SMM-Transfer Monitor

Once invoked, the SMM-transfer monitor (STM) is in VMX root operation and can use VMX instructions to configure VMCSs and to cause VM entries to virtual machines supported by those structures. As noted in Section 29.15.1, the VMXOFF instruction cannot be used under the dual-monitor treatment and thus cannot be used by the STM.

The RSM instruction also cannot be used under the dual-monitor treatment. As noted in Section 25.1.3, it causes a VM exit if executed in SMM in VMX non-root operation. If executed in VMX root operation, it causes an invalid-opcode exception. The STM uses VM entries to return from SMM (see Section 29.15.4).

29.15.4 VM Entries that Return from SMM

The SMM-transfer monitor (STM) returns from SMM using a VM entry with the "entry to SMM" VM-entry control clear. VM entries that return from SMM reverse the effects of an SMM VM exit (see Section 29.15.2).

VM entries that return from SMM may differ from other VM entries in that they do not necessarily enter VMX non-root operation. If the executive-VMCS pointer field in the current VMCS contains the VMXON pointer, the logical processor remains in VMX root operation after VM entry.

For differences between VM entries that return from SMM and other VM entries see Sections 29.15.4.1 through 29.15.4.10.

29.15.4.1 Checks on the Executive-VMCS Pointer Field

VM entries that return from SMM perform the following checks on the executive-VMCS pointer field in the current VMCS:

- Bits 11:0 must be 0.
- The pointer must not set any bits beyond the processor's physical-address width.[1,2]
- The 32 bits located in memory referenced by the physical address in the pointer must contain the processor's VMCS revision identifier (see Section 24.2).

The checks above are performed before the checks described in Section 29.15.4.2 and before any of the following checks:

- 'If the "deactivate dual-monitor treatment" VM-entry control is 0 and the executive-VMCS pointer field does not contain the VMXON pointer, the launch state of the executive VMCS (the VMCS referenced by the executive-VMCS pointer field) must be launched (see Section 24.10.3).
- If the “deactivate dual-monitor treatment” VM-entry control is 1, the executive-VMCS pointer field must contain the VMXON pointer (see Section 29.15.7).[3]

29.15.4.2 Checks on VM-Execution Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-execution control fields specified in Section 26.2.1.1. They do not apply the checks to the current VMCS. Instead, VM-entry behavior depends on whether the executive-VMCS pointer field contains the VMXON pointer:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are not performed at all.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the checks are performed on the VM-execution control fields in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS). These checks are performed after checking the executive-VMCS pointer field itself (for proper alignment).

---

1. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.
2. If IA32_VMX_BASIC[48] is read as 1, this pointer must not set any bits in the range 63:32; see Appendix A.1.
3. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

Other VM entries ensure that, if "activate VMX-preemption timer" VM-execution control is 0, the "save VMX-preemption timer value" VM-exit control is also 0. This check is not performed by VM entries that return from SMM.

29.15.4.3 Checks on VM-Entry Control Fields

VM entries that return from SMM differ from other VM entries with regard to the checks performed on the VM-entry control fields specified in Section 26.2.1.3.

Specifically, if the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the following must **not** all hold for the VM-entry interruption-information field:

- the valid bit (bit 31) in the VM-entry interruption-information field is 1
- the interruption type (bits 10:8) is not 7 (other event); and
- the vector (bits 7:0) is not 0 (pending MTF VM exit).

29.15.4.4 Checks on the Guest State Area

Section 26.3.1 specifies checks performed on fields in the guest-state area of the VMCS. Some of these checks are conditioned on the settings of certain VM-execution controls (e.g., "virtual NMIs" or "unrestricted guest"). VM entries that return from SMM modify these checks based on whether the executive-VMCS pointer field contains the VMXON pointer:[1]

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the checks are performed as all relevant VM-execution controls were 0. (As a result, some checks may not be performed at all.)
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), this check is performed based on the settings of the VM-execution controls in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field in the current VMCS).

For VM entries that return from SMM, the activity-state field must not indicate the wait-for-SIPI state if the executive-VMCS pointer field contains the VMXON pointer (the VM entry is to VMX root operation).

29.15.4.5 Loading Guest State

VM entries that return from SMM load the SMBASE register from the SMBASE field.

VM entries that return from SMM invalidate linear mappings and combined mappings associated with all VPIDs. Combined mappings are invalidated for all EP4TA values (EP4TA is the value of bits 51:12 of EPTP; see Section 28.3). (Ordinary VM entries

1. The STM can determine the VMXON pointer by reading the executive-VMCS pointer field in the current VMCS after the SMM VM exit that activates the dual-monitor treatment.

are required to perform such invalidation only for VPID 0000H and are not required to do even that if the "enable VPID" VM-execution control is 1; see Section 26.3.2.5.)

29.15.4.6 VMX-Preemption Timer

A VM entry that returns from SMM activates the VMX-preemption timer only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation) and the "activate VMX-preemption timer" VM-execution control is 1 in the executive VMCS (the VMCS referenced by the executive-VMCS pointer field). In this case, VM entry starts the VMX-preemption timer with the value in the VMX-preemption timer-value field in the current VMCS.

29.15.4.7 Updating the Current-VMCS and SMM-Transfer VMCS Pointers

Successful VM entries (returning from SMM) load the SMM-transfer VMCS pointer with the current-VMCS pointer. Following this, they load the current-VMCS pointer from a field in the current VMCS:

- If the executive-VMCS pointer field contains the VMXON pointer (the VM entry remains in VMX root operation), the current-VMCS pointer is loaded from the VMCS-link pointer field.
- If the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation), the current-VMCS pointer is loaded with the value of the executive-VMCS pointer field.

If the VM entry successfully enters VMX non-root operation, the VM-execution controls in effect after the VM entry are those from the new current VMCS. This includes any structures external to the VMCS referenced by VM-execution control fields.

The updating of these VMCS pointers occurs before event injection. Event injection is determined, however, by the VM-entry control fields in the VMCS that was current when the VM entry commenced.

29.15.4.8 VM Exits Induced by VM Entry

Section 26.5.1.2 describes how the event-delivery process invoked by event injection may lead to a VM exit. Section 26.6.3 to Section 26.6.7 describe other situations that may cause a VM exit to occur immediately after a VM entry.

Whether these VM exits occur is determined by the VM-execution control fields in the current VMCS. For VM entries that return from SMM, they can occur only if the executive-VMCS pointer field does not contain the VMXON pointer (the VM entry enters VMX non-root operation).

In this case, determination is based on the VM-execution control fields in the VMCS that is current after the VM entry. This is the VMCS referenced by the value of the executive-VMCS pointer field at the time of the VM entry (see Section 29.15.4.7).

This VMCS also controls the delivery of such VM exits. Thus, VM exits induced by a VM entry returning from SMM are to the executive monitor and not to the STM.

29.15.4.9 SMI Blocking

VM entries that return from SMM determine the blocking of system-management interrupts (SMIs) as follows:

- If the "deactivate dual-monitor treatment" VM-entry control is 0, SMIs are blocked after VM entry if and only if the bit 2 in the interruptibility-state field is 1.
- If the "deactivate dual-monitor treatment" VM-entry control is 1, the blocking of SMIs depends on whether the logical processor is in SMX operation:[1]
  - — If the logical processor is in SMX operation, SMIs are blocked after VM entry.
  - — If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

VM entries that return from SMM and that do not deactivate the dual-monitor treatment may leave SMIs blocked. This feature exists to allow the STM to invoke functionality outside of SMM without unblocking SMIs.

29.15.4.10 Failures of VM Entries That Return from SMM

Section 26.7 describes the treatment of VM entries that fail during or after loading guest state. Such failures record information in the VM-exit information fields and load processor state as would be done on a VM exit. The VMCS used is the one that was current before the VM entry commenced. Control is thus transferred to the STM and the logical processor remains in SMM.

29.15.5 Enabling the Dual-Monitor Treatment

Code and data for the SMM-transfer monitor (STM) reside in a region of SMRAM called the **monitor segment** (MSEG). Code running in SMM determines the location of MSEG and establishes its content. This code is also responsible for enabling the dual-monitor treatment.

SMM code enables the dual-monitor treatment and determines the location of MSEG by writing to IA32_SMM_MONITOR_CTL MSR (index 9BH). The MSR has the following format:

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

- Bit 0 is the register's valid bit. The STM may be invoked using VMCALL only if this bit is 1. Because VMCALL is used to activate the dual-monitor treatment (see Section 29.15.6), the dual-monitor treatment cannot be activated if the bit is 0. This bit is cleared when the logical processor is reset.
- Bit 1 is reserved.
- Bit 2 determines whether executions of VMXOFF unblock SMIs under the default treatment of SMIs and SMM. Executions of VMXOFF unblock SMIs unless bit 2 is 1 (the value of bit 0 is irrelevant). See Section 29.14.4.

  Certain leaf functions of the GETSEC instruction clear this bit (see Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*)
- Bits 11:3 are reserved.
- Bits 31:12 contain a value that, when shifted right 12 bits, is the physical address of MSEG (the MSEG base address).
- Bits 63:32 are reserved.

The following items detail use of this MSR:

- The IA32_SMM_MONITOR_CTL MSR is supported only on processors that support the dual-monitor treatment.[1] On other processors, accesses to the MSR using RDMSR or WRMSR generate a general-protection fault (#GP(0)).
- A write to the IA32_SMM_MONITOR_CTL MSR using WRMSR generates a general-protection fault (#GP(0)) if executed outside of SMM or if an attempt is made to set any reserved bit. An attempt to write to IA32_SMM_MONITOR_CTL MSR fails if made as part of a VM exit that does not end in SMM or part of a VM entry that does not begin in SMM.
- Reads from IA32_SMM_MONITOR_CTL MSR using RDMSR are allowed any time RDMSR is allowed. The MSR may be read as part of any VM exit.
- The dual-monitor treatment can be activated only if the valid bit in the MSR is set to 1.

The 32 bytes located at the MSEG base address are called the **MSEG header**. The format of the MSEG header is given in Table 29-10 (each field is 32 bits).

Table 29-10. Format of MSEG Header

Byte Offset	Field
0	MSEG-header revision identifier
4	SMM-transfer monitor features
8	GDTR limit

1. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.

Table 29-10. Format of MSEG Header (Contd.)

Byte Offset	Field
12	GDTR base offset
16	CS selector
20	EIP offset
24	ESP offset
28	CR3 offset

To ensure proper behavior in VMX operation, software should maintain the MSEG header in writeback cacheable memory. Future implementations may allow or require a different memory type.[1] Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1).

SMM code should enable the dual-monitor treatment (by setting the valid bit in IA32_SMM_MONITOR_CTL MSR) only after establishing the content of the MSEG header as follows:

- Bytes 3:0 contain the **MSEG revision identifier**. Different processors may use different MSEG revision identifiers. These identifiers enable software to avoid using an MSEG header formatted for one processor on a processor that uses a different format. Software can discover the MSEG revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_MISC (see Appendix A.6).
- Bytes 7:4 contain the **SMM-transfer monitor features** field. Bits 31:1 of this field are reserved and must be zero. Bit 0 of the field is the **IA-32e mode SMM feature bit**. It indicates whether the logical processor will be in IA-32e mode after the STM is activated (see Section 29.15.6).
- Bytes 31:8 contain fields that determine how processor state is loaded when the STM is activated (see Section 29.15.6.4). SMM code should establish these fields so that activating of the STM invokes the STM's initialization code.

29.15.6 Activating the Dual-Monitor Treatment

The dual-monitor treatment may be enabled by SMM code as described in Section 29.15.5. The dual-monitor treatment is activated only if it is enabled and only by the

1. Alternatively, software may map the MSEG header with the UC memory type; this may be necessary, depending on how memory is organized. Doing so is strongly discouraged unless necessary as it will cause the performance of transitions using those structures to suffer significantly. In addition, the processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with exceptions noted in Appendix A.1.

executive monitor. The executive monitor activates the dual-monitor treatment by executing VMCALL in VMX root operation.

When VMCALL activates the dual-monitor treatment, it causes an SMM VM exit. Differences between this SMM VM exit and other SMM VM exits are discussed in Sections 29.15.6.1 through 29.15.6.5. See also "VMCALL—Call to VM Monitor" in Chapter 6 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

29.15.6.1 Initial Checks

An execution of VMCALL attempts to activate the dual-monitor treatment if (1) the processor supports the dual-monitor treatment;[1] (2) the logical processor is in VMX root operation; (3) the logical processor is outside SMM and the valid bit is set in the IA32_SMM_MONITOR_CTL MSR; (4) the logical processor is not in virtual-8086 mode and not in compatibility mode; (5) CPL = 0; and (6) the dual-monitor treatment is not active.

The VMCS that manages SMM VM exit caused by this VMCALL is the current VMCS established by the executive monitor. The VMCALL performs the following checks on the current VMCS in the order indicated:

1. There must be a current VMCS pointer.
2. The launch state of the current VMCS must be clear.
3. The VM-exit control fields must be valid:
   — Reserved bits in the VM-exit controls must be set properly. Software may consult the VMX capability MSR IA32_VMX_EXIT_CTLS to determine the proper settings (see Appendix A.4).
   — The following checks are performed for the VM-exit MSR-store address if the VM-exit MSR-store count field is non-zero:
      - The lower 4 bits of the VM-exit MSR-store address must be 0. The address should not set any bits beyond the processor's physical-address width.[2]
      - The address of the last byte in the VM-exit MSR-store area should not set any bits beyond the processor's physical-address width. The address of this last byte is VM-exit MSR-store address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

     If IA32_VMX_BASIC[48] is read as 1, neither address should set any bits in the range 63:32; see Appendix A.1.

1. Software should consult the VMX capability MSR IA32_VMX_BASIC (see Appendix A.1) to determine whether the dual-monitor treatment is supported.
2. Software can determine a processor's physical-address width by executing CPUID with 80000008H in EAX. The physical-address width is returned in bits 7:0 of EAX.

If any of these checks fail, subsequent checks are skipped and VMCALL fails. If all these checks succeed, the logical processor uses the IA32_SMM_MONITOR_CTL MSR to determine the base address of MSEG. The following checks are performed in the order indicated:

1. The logical processor reads the 32 bits at the base of MSEG and compares them to the processor's MSEG revision identifier.
2. The logical processor reads the SMM-transfer monitor features field:
   - Bit 0 of the field is the IA-32e mode SMM feature bit, and it indicates whether the logical processor will be in IA-32e mode after the SMM-transfer monitor (STM) is activated.
     - If the VMCALL is executed on a processor that does not support Intel 64 architecture, the IA-32e mode SMM feature bit must be 0.
     - If the VMCALL is executed in 64-bit mode, the IA-32e mode SMM feature bit must be 1.
   - Bits 31:1 of this field are currently reserved and must be zero.

If any of these checks fail, subsequent checks are skipped and the VMCALL fails.

29.15.6.2 MSEG Checking

SMM VM exits that activate the dual-monitor treatment check the following before updating the current-VMCS pointer and the executive-VMCS pointer field (see Section 29.15.2.2):

- The 32 bits at the MSEG base address (used as a physical address) must contain the processor's MSEG revision identifier.
- Bits 31:1 of the SMM-transfer monitor features field in the MSEG header (see Table 29-10) must be 0. Bit 0 of the field (the IA-32e mode SMM feature bit) must be 0 if the processor does not support Intel 64 architecture.

If either of these checks fail, execution of VMCALL fails.

29.15.6.3 Updating the Current-VMCS and Executive-VMCS Pointers

Before performing the steps in Section 29.15.2.2, SMM VM exits that activate the dual-monitor treatment begin by loading the SMM-transfer VMCS pointer with the value of the current-VMCS pointer.

29.15.6.4 Loading Host State

The VMCS that is current during an SMM VM exit that activates the dual-monitor treatment was established by the executive monitor. It does not contain the VM-exit controls and host state required to initialize the STM. For this reason, such SMM VM exits do not load processor state as described in Section 27.5. Instead, state is set to fixed values or loaded based on the content of the MSEG header (see Table 29-10):

- CR0 is set to as follows:
  - — PG, NE, ET, MP, and PE are all set to 1.
  - — CD and NW are left unchanged.
  - — All other bits are cleared to 0.
- CR3 is set as follows:
  - — Bits 63:32 are cleared on processors that supports IA-32e mode.
  - — Bits 31:12 are set to bits 31:12 of the sum of the MSEG base address and the CR3-offset field in the MSEG header.
  - — Bits 11:5 and bits 2:0 are cleared (the corresponding bits in the CR3-offset field in the MSEG header are ignored).
  - — Bits 4:3 are set to bits 4:3 of the CR3-offset field in the MSEG header.
- CR4 is set as follows:
  - — MCE and PGE are cleared.
  - — PAE is set to the value of the IA-32e mode SMM feature bit.
  - — If the IA-32e mode SMM feature bit is clear, PSE is set to 1 if supported by the processor; if the bit is set, PSE is cleared.
  - — All other bits are unchanged.
- DR7 is set to 400H.
- The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.
- The registers CS, SS, DS, ES, FS, and GS are loaded as follows:
  - — All registers are usable.
  - — CS.selector is loaded from the corresponding fields in the MSEG header (the high 16 bits are ignored), with bits 2:0 cleared to 0. If the result is 0000H, CS.selector is set to 0008H.
  - — The selectors for SS, DS, ES, FS, and GS are set to CS.selector+0008H. If the result is 0000H (if the CS selector was 0xFFF8), these selectors are instead set to 0008H.
  - — The base addresses of all registers are cleared to zero.
  - — The segment limits for all registers are set to FFFFFFFFH.
  - — The AR bytes for the registers are set as follows:
    - CS.Type is set to 11 (execute/read, accessed, non-conforming code segment).
    - For SS, DS, FS, and GS, the Type is set to 3 (read/write, accessed, expand-up data segment).

    - The S bits for all registers are set to 1.
    - The DPL for each register is set to 0.
    - The P bits for all registers are set to 1.
    - On processors that support Intel 64 architecture, CS.L is loaded with the value of the IA-32e mode SMM feature bit.
    - CS.D is loaded with the inverse of the value of the IA-32e mode SMM feature bit.
    - For each of SS, DS, FS, and GS, the D/B bit is set to 1.
    - The G bits for all registers are set to 1.
- LDTR is unusable. The LDTR selector is cleared to 0000H, and the register is otherwise undefined (although the base address is always canonical)
- GDTR.base is set to the sum of the MSEG base address and the GDTR base-offset field in the MSEG header (bits 63:32 are always cleared on processors that supports IA-32e mode). GDTR.limit is set to the corresponding field in the MSEG header (the high 16 bits are ignored).
- IDTR.base is unchanged. IDTR.limit is cleared to 0000H.
- RIP is set to the sum of the MSEG base address and the value of the RIP-offset field in the MSEG header (bits 63:32 are always cleared on logical processors that support IA-32e mode).
- RSP is set to the sum of the MSEG base address and the value of the RSP-offset field in the MSEG header (bits 63:32 are always cleared on logical processor that supports IA-32e mode).
- RFLAGS is cleared, except bit 1, which is always set.
- The logical processor is left in the active state.
- Event blocking after the SMM VM exit is as follows:
  — There is no blocking by STI or by MOV SS.
  — There is blocking by non-maskable interrupts (NMIs) and by SMIs.
- There are no pending debug exceptions after the SMM VM exit.
- For processors that support IA-32e mode, the IA32_EFER MSR is modified so that LME and LMA both contain the value of the IA-32e mode SMM feature bit.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs are updated so that, after VM exit, the logical processor does not use translations that were cached before the transition. This is not necessary for changes that would not affect paging due to the settings of other bits (for example, changes to CR4.PSE if IA32_EFER.LMA was 1 before and after the transition).

29.15.6.5 Loading MSRs

The VM-exit MSR-load area is not used by SMM VM exits that activate the dual-monitor treatment. No MSRs are loaded from that area.

29.15.7 Deactivating the Dual-Monitor Treatment

The SMM-transfer monitor may deactivate the dual-monitor treatment and return the processor to default treatment of SMIs and SMM (see Section 29.14). It does this by executing a VM entry with the "deactivate dual-monitor treatment" VM-entry control set to 1.

As noted in Section 26.2.1.3 and Section 29.15.4.1, an attempt to deactivate the dual-monitor treatment fails in the following situations: (1) the processor is not in SMM; (2) the "entry to SMM" VM-entry control is 1; or (3) the executive-VMCS pointer does not contain the VMXON pointer (the VM entry is to VMX non-root operation).

As noted in Section 29.15.4.9, VM entries that deactivate the dual-monitor treatment ignore the SMI bit in the interruptibility-state field of the guest-state area. Instead, the blocking of SMIs following such a VM entry depends on whether the logical processor is in SMX operation:[1]

- If the logical processor is in SMX operation, SMIs are blocked after VM entry. SMIs may later be unblocked by the VMXOFF instruction (see Section 29.14.4) or by certain leaf functions of the GETSEC instruction (see Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*).
- If the logical processor is outside SMX operation, SMIs are unblocked after VM entry.

29.16 SMI AND PROCESSOR EXTENDED STATE MANAGEMENT

On processors that support processor extended states using XSAVE/XRSTOR (see Chapter 13, "System Programming for Instruction Set Extensions and Processor Extended States"), the processor does not save any XSAVE/XRSTOR related state on an SMI. It is the responsibility of the SMM handler code to properly preserve the state information (including CR4.OSXSAVE, XCR0, and possibly processor extended states

1. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B*.

using XSAVE/XRSTOR). Therefore, the SMM handler must follow the rules described in Chapter 13.

CHAPTER 30 VIRTUAL-MACHINE MONITOR PROGRAMMING CONSIDERATIONS

30.1 VMX SYSTEM PROGRAMMING OVERVIEW

The Virtual Machine Monitor (VMM) is a software class used to manage virtual machines (VM). This chapter describes programming considerations for VMMs.

Each VM behaves like a complete physical machine and can run operating system (OS) and applications. The VMM software layer runs at the most privileged level and has complete ownership of the underlying system hardware. The VMM controls creation of a VM, transfers control to a VM, and manages situations that can cause transitions between the guest VMs and host VMM. The VMM allows the VMs to share the underlying hardware and yet provides isolation between the VMs. The guest software executing in a VM is unaware of any transitions that might have occurred between the VM and its host.

30.2 SUPPORTING PROCESSOR OPERATING MODES IN GUEST ENVIRONMENTS

Typically, VMMs transfer control to a VM using VMX transitions referred to as VM entries. The boundary conditions that define what a VM is allowed to execute in isolation are specified in a virtual-machine control structure (VMCS).

As noted in Section 23.8, processors may fix certain bits in CR0 and CR4 to specific values and not support other values. The first processors to support VMX operation require that CR0.PE and CR0.PG be 1 in VMX operation. Thus, a VM entry is allowed only to guests with paging enabled that are in protected mode or in virtual-8086 mode. Guest execution in other processor operating modes need to be specially handled by the VMM.

One example of such a condition is guest execution in real-mode. A VMM could support guest real-mode execution using at least two approaches:

- By using a fast instruction set emulator in the VMM.
- By using the similarity between real-mode and virtual-8086 mode to support real-mode guest execution in a virtual-8086 container. The virtual-8086 container may be implemented as a virtual-8086 container task within a monitor that emulates real-mode guest state and instructions, or by running the guest VM as the virtual-8086 container (by entering the guest with RFLAGS.VM[1] set). Attempts by real-mode code to access privileged state outside the virtual-8086 container would trap to the VMM and would also need to be emulated.

Another example of such a condition is guest execution in protected mode with paging disabled. A VMM could support such guest execution by using “identity” page tables to emulate unpaged protected mode.

30.2.1 Using Unrestricted Guest Mode

Processors which support the “unrestricted guest” VM-execution control allow VM software to run in real-address mode and unpaged protected mode. Since these modes do not use paging, VMM software must virtualize guest memory using EPT.

Special notes for 64-bit VMM software using the 1-setting of the “unrestricted guest” VM-execution control:

- It is recommended that 64-bit VMM software use the 1-settings of the "load IA32_EFER" VM entry control and the "save IA32_EFER" VM-exit control. If VM entry is establishing CR0.PG=0 and if the "IA-32e mode guest" and "load IA32_EFER" VM entry controls are both 0, VM entry leaves IA32_EFER.LME unmodified (i.e., the host value will persist in the guest).
- It is not necessary for VMM software to track guest transitions into and out of IA-32e mode for the purpose of maintaining the correct setting of the "IA-32e mode guest" VM entry control. This is because VM exits on processors supporting the 1-setting of the "unrestricted guest" VM-execution control save the (guest) value of IA32_EFER.LMA into the "IA-32e mode guest" VM entry control.

30.3 MANAGING VMCS REGIONS AND POINTERS

A VMM must observe necessary procedures when working with a VMCS, the associated VMCS pointer, and the VMCS region. It must also not assume the state of persistency for VMCS regions in memory or cache.

Before entering VMX operation, the host VMM allocates a VMXON region. A VMM can host several virtual machines and have many VMCSs active under its management. A unique VMCS region is required for each virtual machine; a VMXON region is required for the VMM itself.

A VMM determines the VMCS region size by reading IA32_VMX_BASIC MSR; it creates VMCS regions of this size using a 4-KByte-aligned area of physical memory. Each VMCS region needs to be initialized with a VMCS revision identifier (at byte offset 0) identical to the revision reported by the processor in the VMX capability MSR.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.).

NOTE

Software must not read or write directly to the VMCS data region as the format is not architecturally defined. Consequently, Intel recommends that the VMM remove any linear-address mappings to VMCS regions before loading.

System software does not need to do special preparation to the VMXON region before entering into VMX operation. The address of the VMXON region for the VMM is provided as an operand to VMXON instruction. Once in VMX root operation, the VMM needs to prepare data fields in the VMCS that control the execution of a VM upon a VM entry. The VMM can make a VMCS the current VMCS by using the VMPTRLD instruction. VMCS data fields must be read or written only through VMREAD and VMWRITE commands respectively.

Every component of the VMCS is identified by a 32-bit encoding that is provided as an operand to VMREAD and VMWRITE. Appendix B provides the encodings. A VMM must properly initialize all fields in a VMCS before using the current VMCS for VM entry.

A VMCS is referred to as a controlling VMCS if it is the current VMCS on a logical processor in VMX non-root operation. A current VMCS for controlling a logical processor in VMX non-root operation may be referred to as a working VMCS if the logical processor is not in VMX non-root operation. The relationship of active, current (i.e. working) and controlling VMCS during VMX operation is shown in Figure 30-1.

NOTE

As noted in Section 24.1, the processor may optimize VMX operation by maintaining the state of an active VMCS (one for which VMPTRLD has been executed) on the processor. Before relinquishing control to other system software that may, without informing the VMM, remove power from the processor (e.g., for transitions to S3 or S4) or leave VMX operation, a VMM must VMCLEAR all active VMCSs. This ensures that all VMCS data cached by the processor are flushed to memory and that no other software can corrupt the current VMM's VMCS data. It is also recommended that the VMM execute VMXOFF after such executions of VMCLEAR.

The VMX capability MSR IA32_VMX_BASIC reports the memory type used by the processor for accessing a VMCS or any data structures referenced through pointers in the VMCS. Software must maintain the VMCS structures in cache-coherent memory. Software must always map the regions hosting the I/O bitmaps, MSR bitmaps, VM-exit MSR-store area, VM-exit MSR-load area, and VM-entry MSR-load area to the write-back (WB) memory type. Mapping these regions to uncacheable (UC) memory type is supported, but strongly discouraged due to negative impact on performance.

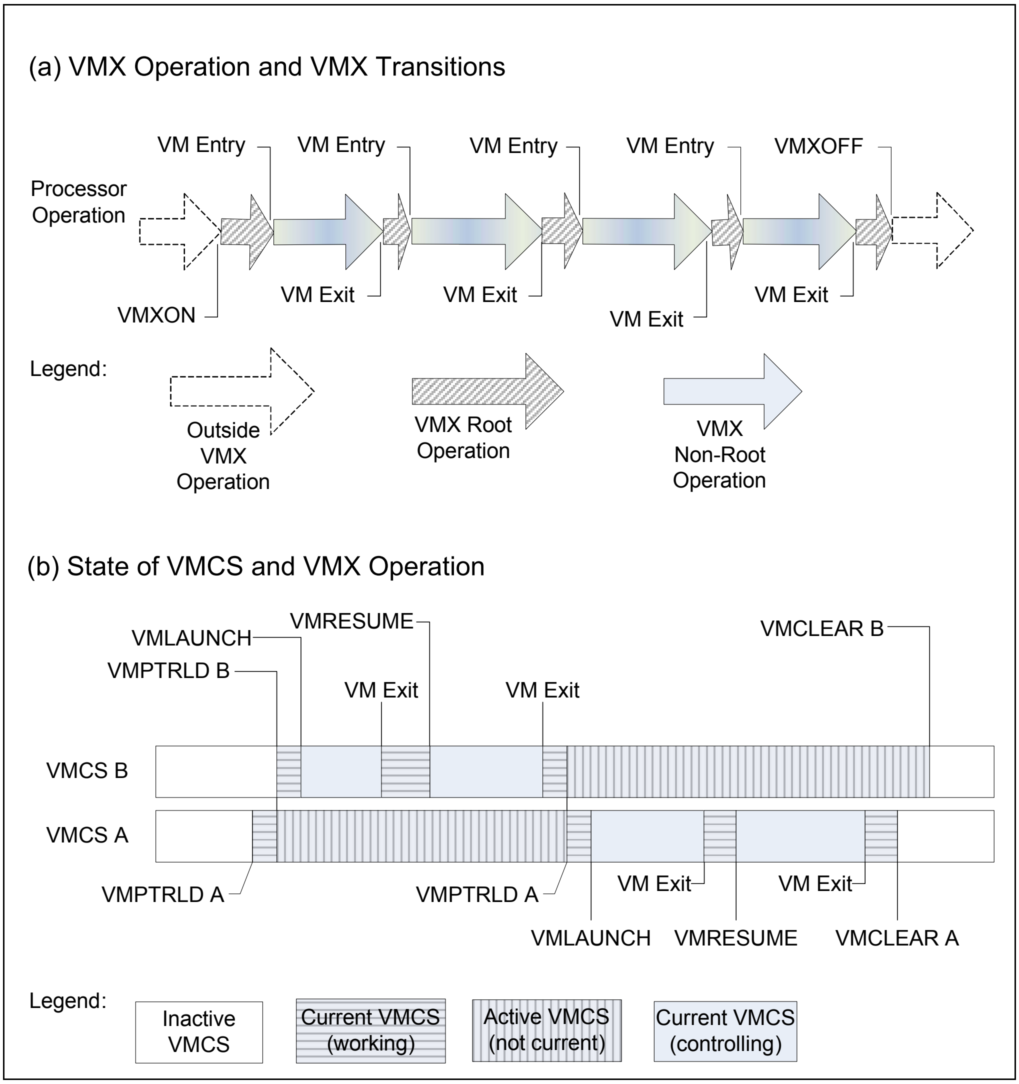

Figure 30-1. VMX Transitions and States of VMCS in a Logical Processor

30.4 USING VMX INSTRUCTIONS

VMX instructions are allowed only in VMX root operation. An attempt to execute a VMX instruction in VMX non-root operation causes a VM exit.

Processors perform various checks while executing any VMX instruction. They follow well-defined error handling on failures. VMX instruction execution failures detected before loading of a guest state are handled by the processor as follows:

- If the working-VMCS pointer is not valid, the instruction fails by setting RFLAGS.CF to 1.
- If the working-VMCS pointer is valid, RFLAGS.ZF is set to 1 and the proper error-code is saved in the VM-instruction error field of the working-VMCS.

Software is required to check RFLAGS.CF and RFLAGS.ZF to determine the success or failure of VMX instruction executions.

The following items provide details regarding use of the VM-entry instructions (VMLAUNCH and VMRESUME):

- If the working-VMCS pointer is valid, the state of the working VMCS may cause the VM-entry instruction to fail. RFLAGS.ZF is set to 1 and one of the following values is saved in the VM-instruction error field:
  - 4: VMLAUNCH with non-clear VMCS.
    If this error occurs, software can avoid the error by executing VMRESUME.
  - 5: VMRESUME with non-launched VMCS.
    If this error occurs, software can avoid the error by executing VMLAUNCH.
  - 6: VMRESUME after VMXOFF.[1]
    If this error occurs, software can avoid the error by executing the following sequence of instructions:

    ```
    VMPTRST ⟨working-VMCS pointer⟩
    VMCLEAR ⟨working-VMCS pointer⟩
    VMPTRLD ⟨working-VMCS pointer⟩
    VMLAUNCH
    ```

    (VMPTRST may not be necessary is software already knows the working-VMCS pointer.)
- If none of the above errors occur, the processor checks on the VMX controls and host-state area. If any of these checks fail, the VM-entry instruction fails. RFLAGS.ZF is set to 1 and either 7 (VM entry with invalid control field(s)) or 8 (VM entry with invalid host-state field(s)) is saved in the VM-instruction error field.
- After a VM-entry instruction (VMRESUME or VMLAUNCH) successfully completes the general checks and checks on VMX controls and the host-state area (see Section 26.2), any errors encountered while loading of guest-state (due to bad guest-state or bad MSR loading) causes the processor to load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 30.7).

1. Earlier versions of this manual described this error as “VMRESUME with a corrupted VMCS”.

This failure behavior differs from that of VM exits in that no guest-state is saved to the guest-state area. A VMM can detect its VM-exit handler was invoked by such a failure by checking bit 31 (for 1) in the exit reason field of the working VMCS and further identify the failure by using the exit qualification field.

See Chapter 26 for more details about the VM-entry instructions.

30.5 VMM SETUP & TEAR DOWN

VMMs need to ensure that the processor is running in protected mode with paging before entering VMX operation. The following list describes the minimal steps required to enter VMX root operation with a VMM running at CPL = 0.

- Check VMX support in processor using CPUID.
- Determine the VMX capabilities supported by the processor through the VMX capability MSRs. See Section 30.5.1 and Appendix A.
- Create a VMXON region in non-pageable memory of a size specified by IA32_VMX_BASIC MSR and aligned to a 4-KByte boundary. Software should read the capability MSRs to determine width of the physical addresses that may be used for the VMXON region and ensure the entire VMXON region can be addressed by addresses with that width. Also, software must ensure that the VMXON region is hosted in cache-coherent memory.
- Initialize the version identifier in the VMXON region (the first 32 bits) with the VMCS revision identifier reported by capability MSRs.
- Ensure the current processor operating mode meets the required CR0 fixed bits (CR0.PE = 1, CR0.PG = 1). Other required CR0 fixed bits can be detected through the IA32_VMX_CR0_FIXED0 and IA32_VMX_CR0_FIXED1 MSRs.
- Enable VMX operation by setting CR4.VMXE = 1. Ensure the resultant CR4 value supports all the CR4 fixed bits reported in the IA32_VMX_CR4_FIXED0 and IA32_VMX_CR4_FIXED1 MSRs.
- Ensure that the IA32_FEATURE_CONTROL MSR (MSR index 3AH) has been properly programmed and that its lock bit is set (Bit 0 = 1). This MSR is generally configured by the BIOS using WRMSR.
- Execute VMXON with the physical address of the VMXON region as the operand. Check successful execution of VMXON by checking if RFLAGS.CF = 0.

Upon successful execution of the steps above, the processor is in VMX root operation.

A VMM executing in VMX root operation and CPL = 0 leaves VMX operation by executing VMXOFF and verifies successful execution by checking if RFLAGS.CF = 0 and RFLAGS.ZF = 0.

If an SMM monitor has been configured to service SMIs while in VMX operation (see Section 29.15), the SMM monitor needs to be torn down before the executive monitor can leave VMX operation (see Section 29.15.7). VMXOFF fails for the executive monitor (a VMM that entered VMX operation by way of issuing VMXON) if SMM monitor is configured.

30.5.1 Algorithms for Determining VMX Capabilities

As noted earlier, a VMM should determine the VMX capabilities supported by the processor by reading the VMX capability MSRs. The architecture for these MSRs is detailed in Appendix A.

As noted in Chapter 24, "Virtual-Machine Control Structures", certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**. Most controls have a default setting of 0; Appendix A.2 identifies those controls that have a default setting of 1. The term **default1** describes the class of controls whose default setting is 1. The are controls in this class from the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls. There are no secondary processor-based VM-execution controls in the default1 class.

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control's new functionality should set the control to its default setting.

The capability MSRs IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS report, respectively, on the allowed settings of most of the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls. However, they will always report that any control in the default1 class must be 1. If a logical processor allows any control in the default1 class to be 0, it indicates this fact by returning 1 for the value of bit 55 of the IA32_VMX_BASIC MSR. If this bit is 1, the logical processor supports the capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. These capability MSRs report, respectively, on the allowed settings of all of the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls.

Software may use one of the following high-level algorithms to determine the correct default control settings:[1]

1. The following algorithm does not use the details given in Appendix A.2:
   a. Ignore bit 55 of the IA32_VMX_BASIC MSR.

1. These algorithms apply only to the pin-based VM-execution controls, the primary processor-based VM-execution controls, the VM-exit controls, and the VM-entry controls. Because there are no secondary processor-based VM-execution controls in the default1 class, a VMM can always set to 0 any such control whose meaning is unknown to it.

   b. Using RDMSR, read the VMX capability MSRs IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS.
   c. Set the VMX controls as follows:
      i) If the relevant VMX capability MSR reports that a control has a single setting, use that setting.
      ii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.
      iii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is not known to the VMM; then set the control to 0.

   A VMM using this algorithm will set to 1 all controls in the default1 class (in step (c)(i)). It will operate correctly even on processors that allow some controls in the default1 class to be 0. However, such a VMM will not be able to use the new features enabled by the 0-setting of such controls. For that reason, this algorithm is not recommended.

2. The following algorithm uses the details given in Appendix A.2. This algorithm requires software to know the identity of the controls in the default1 class:
   a. Using RDMSR, read the IA32_VMX_BASIC MSR.
   b. Use bit 55 of that MSR as follows:
      i) If bit 55 is 0, use RDMSR to read the VMX capability MSRs IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS.
      ii) If bit 55 is 1, use RDMSR to read the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS.
   c. Set the VMX controls as follows:
      i) If the relevant VMX capability MSR reports that a control has a single setting, use that setting.
      ii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.
      iii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the control is not in the default1 class; then set the control to 0.
      iv) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the control is in the default1 class; then set the control to 1.

A VMM using this algorithm will set to 1 all controls in default1 class whose meaning it does not know (either in step (c)(i) or step (c)(iv)). It will operate correctly even on processors that allow some controls in the default1 class to be 0. Unlike a VMM using Algorithm 1, a VMM using Algorithm 2 will be able to use the new features enabled by the 0-setting of such controls.

3. The following algorithm uses the details given in Appendix A.2. This algorithm does not require software to know the identity of the controls in the default1 class:
   a. Using RDMSR, read the VMX capability MSRs IA32_VMX_BASIC, IA32_VMX_PINBASED_CTLS, IA32_VMX_PROCBASED_CTLS, IA32_VMX_EXIT_CTLS, and IA32_VMX_ENTRY_CTLS.
   b. If bit 55 of the IA32_VMX_BASIC MSR is 0, set the VMX controls as follows:
      i) If the relevant VMX capability MSR reports that a control has a single setting, use that setting.
      ii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.
      iii) If (1) the relevant VMX capability MSR reports that a control can be set to 0 or 1; and (2) the control's meaning is not known to the VMM; then set the control to 0.
   c. If bit 55 of the IA32_VMX_BASIC MSR is 1, use RDMSR to read the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. Set the VMX controls as follows:
      i) If the relevant VMX capability MSR just read reports that a control has a single setting, use that setting.
      ii) If (1) the relevant VMX capability MSR just read reports that a control can be set to 0 or 1; and (2) the control's meaning is known to the VMM; then set the control based on functionality desired.
      iii) If (1) the relevant VMX capability MSR just read reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the relevant VMX capability MSR as read in step (a) reports that a control can be set to 0; then set the control to 0.
      iv) If (1) the relevant VMX capability MSR just read reports that a control can be set to 0 or 1; (2) the control's meaning is not known to the VMM; and (3) the relevant VMX capability MSR as read in step (a) reports that a control must be 1; then set the control to 1.

   A VMM using this algorithm will set to 1 all controls in the default1 class whose meaning it does not know (in step (b)(i), step (c)(i), or step (c)(iv)). It will operate correctly even on processors that allow some controls in the default1 class to be 0. Unlike a VMM using Algorithm 1, a VMM using Algorithm 3 will be able to use the new features enabled by the 0-setting of such controls. Unlike a

VMM using Algorithm 2, a VMM using Algorithm 3 need not know the identities of the controls in the default1 class.

30.6 PREPARATION AND LAUNCHING A VIRTUAL MACHINE

The following list describes the minimal steps required by the VMM to set up and launch a guest VM.

- Create a VMCS region in non-pageable memory of size specified by the VMX capability MSR IA32_VMX_BASIC and aligned to 4-KBytes. Software should read the capability MSRs to determine width of the physical addresses that may be used for a VMCS region and ensure the entire VMCS region can be addressed by addresses with that width. The term “guest-VMCS address” refers to the physical address of the new VMCS region for the following steps.
- Initialize the version identifier in the VMCS (first 32 bits) with the VMCS revision identifier reported by the VMX capability MSR IA32_VMX_BASIC.
- Execute the VMCLEAR instruction by supplying the guest-VMCS address. This will initialize the new VMCS region in memory and set the launch state of the VMCS to “clear”. This action also invalidates the working-VMCS pointer register to FFFFFFFF_FFFFFFFFH. Software should verify successful execution of VMCLEAR by checking if RFLAGS.CF = 0 and RFLAGS.ZF = 0.
- Execute the VMPTRLD instruction by supplying the guest-VMCS address. This initializes the working-VMCS pointer with the new VMCS region’s physical address.
- Issue a sequence of VMWRITEs to initialize various host-state area fields in the working VMCS. The initialization sets up the context and entry-points to the VMM upon subsequent VM exits from the guest. Host-state fields include control registers (CR0, CR3 and CR4), selector fields for the segment registers (CS, SS, DS, ES, FS, GS and TR), and base-address fields (for FS, GS, TR, GDTR and IDTR; RSP, RIP and the MSRs that control fast system calls).

  Chapter 25 describes the host-state consistency checking done by the processor for VM entries. The VMM is required to set up host-state that comply with these consistency checks. For example, VMX requires the host-area to have a task register (TR) selector with TI and RPL fields set to 0 and pointing to a valid TSS.
- Use VMWRITEs to set up the various VM-exit control fields, VM-entry control fields, and VM-execution control fields in the VMCS. Care should be taken to make sure the settings of individual fields match the allowed 0 and 1 settings for the respective controls as reported by the VMX capability MSRs (see Appendix A). Any settings inconsistent with the settings reported by the capability MSRs will cause VM entries to fail.
- Use VMWRITE to initialize various guest-state area fields in the working VMCS. This sets up the context and entry-point for guest execution upon VM entry.

Chapter 25 describes the guest-state loading and checking done by the processor for VM entries to protected and virtual-8086 guest execution.

- The VMM is required to set up guest-state that complies with these consistency checks:
  - If the VMM design requires the initial VM launch to cause guest software (typically the guest virtual BIOS) execution from the guest's reset vector, it may need to initialize the guest execution state to reflect the state of a physical processor at power-on reset (described in Chapter 9, *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
  - The VMM may need to initialize additional guest execution state that is not captured in the VMCS guest-state area by loading them directly on the respective processor registers. Examples include general purpose registers, the CR2 control register, debug registers, floating point registers and so forth. VMM may support lazy loading of FPU, MMX, SSE, and SSE2 states with CR0.TS = 1 (described in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).
- Execute VMLAUNCH to launch the guest VM. If VMLAUNCH fails due to any consistency checks before guest-state loading, RFLAGS.CF or RFLAGS.ZF will be set and the VM-instruction error field (see Section 24.9.5) will contain the error-code. If guest-state consistency checks fail upon guest-state loading, the processor loads state from the host-state area as if a VM exit had occurred (see Section 30.6).

VMLAUNCH updates the controlling-VMCS pointer with the working-VMCS pointer and saves the old value of controlling-VMCS as the parent pointer. In addition, the launch state of the guest VMCS is changed to "launched" from "clear". Any programmed exit conditions will cause the guest to VM exit to the VMM. The VMM should execute VMRESUME instruction for subsequent VM entries to guests in a "launched" state.

30.7 HANDLING OF VM EXITS

This section provides examples of software steps involved in a VMM's handling of VM-exit conditions:

- Determine the exit reason through a VMREAD of the exit-reason field in the working-VMCS. Appendix C describes exit reasons and their encodings.
- VMREAD the exit-qualification from the VMCS if the exit-reason field provides a valid qualification. The exit-qualification field provides additional details on the VM-exit condition. For example, in case of page faults, the exit-qualification field provides the guest linear address that caused the page fault.
- Depending on the exit reason, fetch other relevant fields from the VMCS. Appendix C lists the various exit reasons.

- Handle the VM-exit condition appropriately in the VMM. This may involve the VMM emulating one or more guest instructions, programming the underlying host hardware resources, and then re-entering the VM to continue execution.

30.7.1 Handling VM Exits Due to Exceptions

As noted in Section 25.3, an exception causes a VM exit if the bit corresponding to the exception's vector is set in the exception bitmap. (For page faults, the error code also determines whether a VM exit occurs.) This section provide some guidelines of how a VMM might handle such exceptions.

Exceptions result when a logical processor encounters an unusual condition that software may not have expected. When guest software encounters an exception, it may be the case that the condition was caused by the guest software. For example, a guest application may attempt to access a page that is restricted to supervisor access. Alternatively, the condition causing the exception may have been established by the VMM. For example, a guest OS may attempt to access a page that the VMM has chosen to make not present.

When the condition causing an exception was established by guest software, the VMM may choose to **reflect** the exception to guest software. When the condition was established by the VMM itself, the VMM may choose to **resume** guest software after removing the condition.

30.7.1.1 Reflecting Exceptions to Guest Software

If the VMM determines that a VM exit was caused by an exception due to a condition established by guest software, it may reflect that exception to guest software. The VMM would cause the exception to be delivered to guest software, where it can be handled as it would be if the guest were running on a physical machine. This section describes how that may be done.

In general, the VMM can deliver the exception to guest software using VM-entry event injection as described in Section 26.5. The VMM can copy (using VMREAD and VMWRITE) the contents of the VM-exit interruption-information field (which is valid, since the VM exit was caused by an exception) to the VM-entry interruption-information field (which, if valid, will cause the exception to be delivered as part of the next VM entry). The VMM would also copy the contents of the VM-exit interruption error-code field to the VM-entry exception error-code field; this need not be done if bit 11 (error code valid) is clear in the VM-exit interruption-information field. After this, the VMM can execute VMRESUME.

The following items provide details that may qualify the general approach:

- Care should be taken to ensure that reserved bits 30:12 in the VM-entry interruption-information field are 0. In particular, some VM exits may set bit 12 in the VM-exit interruption-information field to indicate NMI unblocking due to IRET. If this bit is copied as 1 into the VM-entry interruption-information field, the next VM entry will fail because that bit should be 0.

- Bit 31 (valid) of the IDT-vectoring information field indicates, if set, that the exception causing the VM exit occurred while another event was being delivered to guest software. If this is the case, it may not be appropriate simply to reflect that exception to guest software. To provide proper virtualization of the exception architecture, a VMM should handle nested events as a physical processor would. Processor handling is described in Chapter 6, "Interrupt 8—Double Fault Exception (#DF)" in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.
  - — The VMM should reflect the exception causing the VM exit to guest software in any of the following cases:
    - The value of bits 10:8 (interruption type) of the IDT-vectoring information field is anything other than 3 (hardware exception).
    - The value of bits 7:0 (vector) of the IDT-vectoring information field indicates a benign exception (1, 2, 3, 4, 5, 6, 7, 9, 16, 17, 18, or 19).
    - The value of bits 7:0 (vector) of the VM-exit interruption-information field indicates a benign exception.
    - The value of bits 7:0 of the IDT-vectoring information field indicates a contributory exception (0, 10, 11, 12, or 13) and the value of bits 7:0 of the VM-exit interruption-information field indicates a page fault (14).
  - — If the value of bits 10:8 of the IDT-vectoring information field is 3 (hardware exception), the VMM should reflect a double-fault exception to guest software in any of the following cases:
    - The value of bits 7:0 of the IDT-vectoring information field and the value of bits 7:0 of the VM-exit interruption-information field each indicates a contributory exception.
    - The value of bits 7:0 of the IDT-vectoring information field indicates a page fault and the value of bits 7:0 of the VM-exit interruption-information field indicates either a contributory exception or a page fault.

    A VMM can reflect a double-fault exception to guest software by setting the VM-entry interruption-information and VM-entry exception error-code fields as follows:

    - Set bits 7:0 (vector) of the VM-entry interruption-information field to 8 (#DF).
    - Set bits 10:8 (interruption type) of the VM-entry interruption-information field to 3 (hardware exception).
    - Set bit 11 (deliver error code) of the VM-entry interruption-information field to 1.
    - Clear bits 30:12 (reserved) of VM-entry interruption-information field.
    - Set bit 31 (valid) of VM-entry interruption-information field.
    - Set the VM-entry exception error-code field to zero.

 — If the value of bits 10:8 of the IDT-vectoring information field is 3 (hardware exception) and the value of bits 7:0 is 8 (#DF), guest software would have encountered a triple fault. Event injection should not be used in this case. The VMM may choose to terminate the guest, or it might choose to enter the guest in the shutdown activity state.

30.7.1.2 Resuming Guest Software after Handling an Exception

If the VMM determines that a VM exit was caused by an exception due to a condition established by the VMM itself, it may choose to resume guest software after removing the condition. The approach for removing the condition may be specific to the VMM's software architecture. and algorithms This section describes how guest software may be resumed after removing the condition.

In general, the VMM can resume guest software simply by executing VMRESUME. The following items provide details of cases that may require special handling:

- If the "NMI exiting" VM-execution control is 0, bit 12 of the VM-exit interruption-information field indicates that the VM exit was due to a fault encountered during an execution of the IRET instruction that unblocked non-maskable interrupts (NMIs). In particular, it provides this indication if the following are both true:
  - — Bit 31 (valid) in the IDT-vectoring information field is 0.
  - — The value of bits 7:0 (vector) of the VM-exit interruption-information field is not 8 (the VM exit is not due to a double-fault exception).

  If both are true and bit 12 of the VM-exit interruption-information field is 1, NMIs were blocked before guest software executed the IRET instruction that caused the fault that caused the VM exit. The VMM should set bit 3 (blocking by NMI) in the interruptibility-state field (using VMREAD and VMWRITE) before resuming guest software.
- If the "virtual NMIs" VM-execution control is 1, bit 12 of the VM-exit interruption-information field indicates that the VM exit was due to a fault encountered during an execution of the IRET instruction that removed virtual-NMI blocking. In particular, it provides this indication if the following are both true:
  - — Bit 31 (valid) in the IDT-vectoring information field is 0.
  - — The value of bits 7:0 (vector) of the VM-exit interruption-information field is not 8 (the VM exit is not due to a double-fault exception).

  If both are true and bit 12 of the VM-exit interruption-information field is 1, there was virtual-NMI blocking before guest software executed the IRET instruction that caused the fault that caused the VM exit. The VMM should set bit 3 (blocking by NMI) in the interruptibility-state field (using VMREAD and VMWRITE) before resuming guest software.
- Bit 31 (valid) of the IDT-vectoring information field indicates, if set, that the exception causing the VM exit occurred while another event was being delivered to guest software. The VMM should ensure that the other event is delivered when

guest software is resumed. It can do so using the VM-entry event injection described in Section 26.5 and detailed in the following paragraphs:

— The VMM can copy (using VMREAD and VMWRITE) the contents of the IDT-vectoring information field (which is presumed valid) to the VM-entry interruption-information field (which, if valid, will cause the exception to be delivered as part of the next VM entry).

  - The VMM should ensure that reserved bits 30:12 in the VM-entry interruption-information field are 0. In particular, the value of bit 12 in the IDT-vectoring information field is undefined after all VM exits. If this bit is copied as 1 into the VM-entry interruption-information field, the next VM entry will fail because the bit should be 0.
  - If the “virtual NMIs” VM-execution control is 1 and the value of bits 10:8 (interruption type) in the IDT-vectoring information field is 2 (indicating NMI), the VM exit occurred during delivery of an NMI that had been injected as part of the previous VM entry. In this case, bit 3 (blocking by NMI) will be 1 in the interruptibility-state field in the VMCS. The VMM should clear this bit; otherwise, the next VM entry will fail (see Section 26.3.1.5).

— The VMM can also copy the contents of the IDT-vectoring error-code field to the VM-entry exception error-code field. This need not be done if bit 11 (error code valid) is clear in the IDT-vectoring information field.

— The VMM can also copy the contents of the VM-exit instruction-length field to the VM-entry instruction-length field. This need be done only if bits 10:8 (interruption type) in the IDT-vectoring information field indicate either software interrupt, privileged software exception, or software exception.

30.8 MULTI-PROCESSOR CONSIDERATIONS

The most common VMM design will be the symmetric VMM. This type of VMM runs the same VMM binary on all logical processors. Like a symmetric operating system, the symmetric VMM is written to ensure all critical data is updated by only one processor at a time, IO devices are accessed sequentially, and so forth. Asymmetric VMM designs are possible. For example, an asymmetric VMM may run its scheduler on one processor and run just enough of the VMM on other processors to allow the correct execution of guest VMs. The remainder of this section focuses on the multi-processor considerations for a symmetric VMM.

A symmetric VMM design does not preclude asymmetry in its operations. For example, a symmetric VMM can support asymmetric allocation of logical processor resources to guests. Multiple logical processors can be brought into a single guest environment to support an MP-aware guest OS. Because an active VMCS can not control more than one logical processor simultaneously, a symmetric VMM must make copies of its VMCS to control the VM allocated to support an MP-aware guest

OS. Care must be taken when accessing data structures shared between these VMCSs. See Section 30.8.4.

Although it may be easier to develop a VMM that assumes a fully-symmetric view of hardware capabilities (with all processors supporting the same processor feature sets, including the same revision of VMX), there are advantages in developing a VMM that comprehends different levels of VMX capability (reported by VMX capability MSRs). One possible advantage of such an approach could be that an existing software installation (VMM and guest software stack) could continue to run without requiring software upgrades to the VMM, when the software installation is upgraded to run on hardware with enhancements in the processor's VMX capabilities. Another advantage could be that a single software installation image, consisting of a VMM and guests, could be deployed to multiple hardware platforms with varying VMX capabilities. In such cases, the VMM could fall back to a common subset of VMX features supported by all VMX revisions, or choose to understand the asymmetry of the VMX capabilities and assign VMs accordingly.

This section outlines some of the considerations to keep in mind when developing an MP-aware VMM.

30.8.1 Initialization

Before enabling VMX, an MP-aware VMM must check to make sure that all processors in the system are compatible and support features required. This can be done by:

- Checking the CPUID on each logical processor to ensure VMX is supported and that the overall feature set of each logical processor is compatible.
- Checking VMCS revision identifiers on each logical processor.
- Checking each of the "allowed-1" or "allowed-0" fields of the VMX capability MSR's on each processor.

30.8.2 Moving a VMCS Between Processors

An MP-aware VMM is free to assign any logical processor to a VM. But for performance considerations, moving a guest VMCS to another logical processor is slower than resuming that guest VMCS on the same logical processor. Certain VMX performance features (such as caching of portions of the VMCS in the processor) are optimized for a guest VMCS that runs on the same logical processor.

The reasons are:

- To restart a guest on the same logical processor, a VMM can use VMRESUME. VMRESUME is expected to be faster than VMLAUNCH in general.
- To migrate a VMCS to another logical processor, a VMM must use the sequence of VMCLEAR, VMPTRLD and VMLAUNCH.
- Operations involving VMCLEAR can impact performance negatively. See Section 24.10.3.

A VMM scheduler should make an effort to schedule a guest VMCS to run on the logical processor where it last ran. Such a scheduler might also benefit from doing lazy VMCLEARs (that is: performing a VMCLEAR on a VMCS only when the scheduler knows the VMCS is being moved to a new logical processor). The remainder of this section describes the steps a VMM must take to move a VMCS from one processor to another.

A VMM must check the VMCS revision identifier in the VMX capability MSR IA32_VMX_BASIC to determine if the VMCS regions are identical between all logical processors. If the VMCS regions are identical (same revision ID) the following sequence can be used to move or copy the VMCS from one logical processor to another:

- Perform a VMCLEAR operation on the source logical processor. This ensures that all VMCS data that may be cached by the processor are flushed to memory.
- Copy the VMCS region from one memory location to another location. This is an optional step assuming the VMM wishes to relocate the VMCS or move the VMCS to another system.
- Perform a VMPTRLD of the physical address of VMCS region on the destination processor to establish its current VMCS pointer.

If the revision identifiers are different, each field must be copied to an intermediate structure using individual reads (VMREAD) from the source fields and writes (VMWRITE) to destination fields. Care must be taken on fields that are hard-wired to certain values on some processor implementations.

30.8.3 Paired Index-Data Registers

A VMM may need to virtualize hardware that is visible to software using paired index-data registers. Paired index-data register interfaces, such as those used in PCI (CF8, CFC), require special treatment in cases where a VM performing writes to these pairs can be moved during execution. In this case, the index (e.g. CF8) should be part of the virtualized state. If the VM is moved during execution, writes to the index should be redone so subsequent data reads/writes go to the right location.

30.8.4 External Data Structures

Certain fields in the VMCS point to external data structures (for example: the MSR bitmap, the I/O bitmaps). If a logical processor is in VMX non-root operation, none of the external structures referenced by that logical processor's current VMCS should be modified by any logical processor or DMA. Before updating one of these structures, the VMM must ensure that no logical processor whose current VMCS references the structure is in VMX non-root operation.

If a VMM uses multiple VMCS with each VMCS using separate external structures, and these structures must be kept synchronized, the VMM must apply the same care to updating these structures.

30.8.5 CPUID Emulation

CPUID reports information that is used by OS and applications to detect hardware features. It also provides multi-threading/multi-core configuration information. For example, MP-aware OSs rely on data reported by CPUID to discover the topology of logical processors in a platform (see Section 8.9, "Programming Considerations for Hardware Multi-Threading Capable Processors," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*).

If a VMM is to support asymmetric allocation of logical processor resources to guest OSs that are MP aware, then the VMM must emulate CPUID for its guests. The emulation of CPUID by the VMM must ensure the guest's view of CPUID leaves are consistent with the logical processor allocation committed by the VMM to each guest OS.

30.9 32-BIT AND 64-BIT GUEST ENVIRONMENTS

For the most part, extensions provided by VMX to support virtualization are orthogonal to the extensions provided by Intel 64 architecture. There are considerations that impact VMM designs. These are described in the following subsections.

30.9.1 Operating Modes of Guest Environments

For Intel 64 processors, VMX operation supports host and guest environments that run in IA-32e mode or without IA-32e mode. VMX operation also supports host and guest environments on IA-32 processors.

A VMM entering VMX operation while IA-32e mode is active is considered to be an IA-32e mode host. A VMM entering VMX operation while IA-32e mode is not activated or not available is referred to as a 32-bit VMM. The type of guest operations such VMMs support are summarized in Table 30-1.

Table 30-1. Operating Modes for Host and Guest Environments

Capability	Guest Operation in IA-32e mode	Guest Operation Not Requiring IA-32e Mode
IA-32e mode VMM	Yes	Yes
32-bit VMM	Not supported	Yes

A VM exit may occur to an IA-32e mode guest in either 64-bit sub-mode or compatibility sub-mode of IA-32e mode. VMMs may resume guests in either mode. The sub-mode in which an IA-32e mode guest resumes VMX non-root operation is determined by the attributes of the code segment which experienced the VM exit. If CS.L = 1, the guest is executing in 64-bit mode; if CS.L = 0, the guest is executing in compatibility mode (see Section 30.9.5).

Not all of an IA-32e mode VMM must run in 64-bit mode. While some parts of an IA-32e mode VMM must run in 64-bit mode, there are only a few restrictions preventing a VMM from executing in compatibility mode. The most notable restriction is that most VMX instructions cause exceptions when executed in compatibility mode.

30.9.2 Handling Widths of VMCS Fields

Individual VMCS control fields must be accessed using VMREAD or VMWRITE instructions. Outside of 64-Bit mode, VMREAD and VMWRITE operate on 32 bits of data. The widths of VMCS control fields may vary depending on whether a processor supports Intel 64 architecture.

Many VMCS fields are architected to extend transparently on processors supporting Intel 64 architecture (64 bits on processors that support Intel 64 architecture, 32 bits on processors that do not). Some VMCS fields are 64-bits wide regardless of whether the processor supports Intel 64 architecture or is in IA-32e mode.

30.9.2.1 Natural-Width VMCS Fields

Many VMCS fields operate using natural width. Such fields return (on reads) and set (on writes) 32-bits when operating in 32-bit mode and 64-bits when operating in 64-bit mode. For the most part, these fields return the naturally expected data widths. The “Guest RIP” field in the VMCS guest-state area is an example of this type of field.

30.9.2.2 64-Bit VMCS Fields

Unlike natural width fields, these fields are fixed to 64-bit width on all processors. When in 64-bit mode, reads of these fields return 64-bit wide data and writes to these fields write 64-bits. When outside of 64-bit mode, reads of these fields return the low 32-bits and writes to these fields write the low 32-bits and zero the upper 32-bits. Should a non-IA-32e mode host require access to the upper 32-bits of these fields, a separate VMCS encoding is used when issuing VMREAD/VMWRITE instructions.

The VMCS control field “MSR bitmap address” (which contains the physical address of a region of memory which specifies which MSR accesses should generate VM-exits) is an example of this type of field. Specifying encoding 00002004H to VMREAD returns the lower 32-bits to non-IA-32e mode hosts and returns 64-bits to 64-bit hosts. The separate encoding 00002005H returns only the upper 32-bits.

30.9.3 IA-32e Mode Hosts

An IA-32e mode host is required to support 64-bit guest environments. Because activating IA-32e mode currently requires that paging be disabled temporarily and VMX entry requires paging to be enabled, IA-32e mode must be enabled before entering

VMX operation. For this reason, it is not possible to toggle in and out of IA-32e mode in a VMM.

Section 30.5 describes the steps required to launch a VMM. An IA-32e mode host is also required to set the “host address-space size” VMCS VM-exit control to 1. The value of this control is then loaded in the IA32_EFER.LME/LMA and CS.L bits on each VM exit. This establishes a 64-bit host environment as execution transfers to the VMM entry point. At a minimum, the entry point is required to be in a 64-bit code segment. Subsequently, the VMM can, if it chooses, switch to 32-bit compatibility mode on a code-segment basis (see Section 30.9.1). Note, however, that VMX instructions other than VMCALL are not supported in compatibility mode; they generate an invalid opcode exception if used.

The following VMCS controls determine the value of IA32_EFER when a VM exit occurs: the “host address-space size” control (described above), the “load IA32_EFER” VM-exit control, the “VM-exit MSR-load count,” and the “VM-exit MSR-load address” (see Section 27.3).

If the “load IA32_EFER” VM-exit control is 1, the value of the LME and LMA bits in the IA32_EFER field in the host-state area must be the value of the “host address-space size” VM-exit control.

The loading of IA32_EFER.LME/LMA and CS.L bits established by the “host address-space size” control precede any loading of the IA32_EFER MSR due from the VM-exit MSR-load area. If IA32_EFER is specified in the VM-exit MSR-load area, the value of the LME bit in the load image of IA32_EFER should match the setting of the “host address-space size” control. Otherwise the attempt to modify the LME bit (while paging is enabled) will lead to a VMX-abort. However, IA32_EFER.LMA is always set by the processor to equal IA32_EFER.LME & CR0.PG; the value specified for LMA in the load image of the IA32_EFER MSR is ignored. For these and performance reasons, VMM writers may choose to not use the VM-exit/entry MSR-load/save areas for IA32_EFER.

On a VMM teardown, VMX operation should be exited before deactivating IA-32e mode if the latter is required.

30.9.4 IA-32e Mode Guests

A 32-bit guest can be launched by either IA-32e-mode hosts or non-IA-32e-mode hosts. A 64-bit guests can only be launched by a IA-32e-mode host.

In addition to the steps outlined in Section 30.6, VMM writers need to:

- Set the “IA-32e-mode guest” VM-entry control to 1 in the VMCS to assure VM-entry (VMLAUNCH or VMRESUME) will establish a 64-bit (or 32-bit compatible) guest operating environment.
- Enable paging (CR0.PG) and PAE mode (CR4.PAE) to assure VM-entry to a 64-bit guest will succeed.

- Ensure that the host to be in IA-32e mode (the IA32_EFER.LMA must be set to 1) and the setting of the VM-exit “host address-space size” control bit in the VMCS must also be set to 1.

If each of the above conditions holds true, then VM-entry will copy the value of the VM-entry “IA-32e-mode guest” control bit into the guests IA32_EFER.LME bit, which will result in subsequent activation of IA-32e mode. If any of the above conditions is false, the VM-entry will fail and load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 26.7).

The following VMCS controls determine the value of IA32_EFER on a VM entry: the “IA-32e-mode guest” VM-entry control (described above), the “load IA32_EFER” VM-entry control, the “VM-entry MSR-load count,” and the “VM-entry MSR-load address” (see Section 26.4).

If the “load IA32_EFER” VM-entry control is 1, the value of the LME and LMA bits in the IA32_EFER field in the guest-state area must be the value of the “IA-32e-mode guest” VM-exit control. Otherwise, the VM entry fails.

The loading of IA32_EFER.LME bit (described above) precedes any loading of the IA32_EFER MSR from the VM-entry MSR-load area of the VMCS. If loading of IA32_EFER is specified in the VM-entry MSR-load area, the value of the LME bit in the load image should be match the setting of the “IA-32e-mode guest” VM-entry control. Otherwise, the attempt to modify the LME bit (while paging is enabled) results in a failed VM entry. However, IA32_EFER.LMA is always set by the processor to equal IA32_EFER.LME & CR0.PG; the value specified for LMA in the load image of the IA32_EFER MSR is ignored. For these and performance reasons, VMM writers may choose to not use the VM-exit/entry MSR-load/save areas for IA32_EFER MSR.

Note that the VMM can control the processor’s architectural state when transferring control to a VM. VMM writers may choose to launch guests in protected mode and subsequently allow the guest to activate IA-32e mode or they may allow guests to toggle in and out of IA-32e mode. In this case, the VMM should require VM exit on accesses to the IA32_EFER MSR to detect changes in the operating mode and modify the VM-entry “IA-32e-mode guest” control accordingly.

A VMM should save/restore the extended (full 64-bit) contents of the guest general-purpose registers, the new general-purpose registers (R8-R15) and the SIMD registers introduced in 64-bit mode should it need to modify these upon VM exit.

30.9.5 32-Bit Guests

To launch or resume a 32-bit guest, VMM writers can follow the steps outlined in Section 30.6, making sure that the “IA-32e-mode guest” VM-entry control bit is set to 0. Then the “IA-32e-mode guest” control bit is copied into the guest IA32_EFER.LME bit, establishing IA32_EFER.LMA as 0.

30.10 HANDLING MODEL SPECIFIC REGISTERS

Model specific registers (MSR) provide a wide range of functionality. They affect processor features, control the programming interfaces, or are used in conjunction with specific instructions. As part of processor virtualization, a VMM may wish to protect some or all MSR resources from direct guest access.

VMX operation provides the following features to virtualize processor MSRs.

30.10.1 Using VM-Execution Controls

Processor-based VM-execution controls provide two levels of support for handling guest access to processor MSRs using RDMSR and WRMSR:

- **MSR bitmaps**: In VMX implementations that support a 1-setting (see Appendix A) of the user-MSR-bitmaps execution control bit, MSR bitmaps can be used to provide flexibility in managing guest MSR accesses. The MSR-bitmap-address in the guest VMCS can be programmed by VMM to point to a bitmap region which specifies VM-exit behavior when reading and writing individual MSRs.

  MSR bitmaps form a 4-KByte region in physical memory and are required to be aligned to a 4-KByte boundary. The first 1-KByte region manages read control of MSRs in the range 00000000H-00001FFFH; the second 1-KByte region covers read control of MSR addresses in the range C0000000H-C0001FFFH. The bitmaps for write control of these MSRs are located in the 2-KByte region immediately following the read control bitmaps. While the MSR bitmap address is part of VMCS, the MSR bitmaps themselves are not. This implies MSR bitmaps are not accessible through VMREAD and VMWRITE instructions but rather by using ordinary memory writes. Also, they are not specially cached by the processor and may be placed in normal cache-coherent memory by the VMM.

  When MSR bitmap addresses are properly programmed and the use-MSR-bitmap control (see Section 24.6.2) is set, the processor consults the associated bit in the appropriate bitmap on guest MSR accesses to the corresponding MSR and causes a VM exit if the bit in the bitmap is set. Otherwise, the access is permitted to proceed. This level of protection may be utilized by VMMs to selectively allow guest access to some MSRs while virtualizing others.
- **Default MSR protection**: If the use-MSR-bitmap control is not set, an attempt by a guest to access any MSR causes a VM exit. This also occurs for any attempt to access an MSR outside the ranges identified above (even if the use-MSR-bitmap control is set).

VM exits due to guest MSR accesses may be identified by the VMM through VM-exit reason codes. The MSR-read exit reason implies guest software attempted to read an MSR protected either by default or through MSR bitmaps. The MSR-write exit reason implies guest software attempting to write a MSR protected through the VM-execution controls. Upon VM exits caused by MSR accesses, the VMM may virtualize the guest MSR access through emulation of RDMSR/WRMSR.

30.10.2 Using VM-Exit Controls for MSRs

If a VMM allows its guest to access MSRs directly, the VMM may need to store guest MSR values and load host MSR values for these MSRs on VM exits. This is especially true if the VMM uses the same MSRs while in VMX root operation.

A VMM can use the VM-exit MSR-store-address and the VM-exit MSR-store-count exit control fields (see Section 24.7.2) to manage how MSRs are stored on VM exits. The VM-exit MSR-store-address field contains the physical address (16-byte aligned) of the VM-exit MSR-store area (a table of entries with 16 bytes per entry). Each table entry specifies an MSR whose value needs to be stored on VM exits. The VM-exit MSR-store-count contains the number of entries in the table.

Similarly the VM-exit MSR-load-address and VM-exit MSR-load-count fields point to the location and size of the VM-exit MSR load area. The entries in the VM-exit MSR-load area contain the host expected values of specific MSRs when a VM exit occurs.

Upon VM-exit, bits 127:64 of each entry in the VM-exit MSR-store area is updated with the contents of the MSR indexed by bits 31:0. Also, bits 127:64 of each entry in the VM-exit MSR-load area is updated by loading with values from bits 127:64 the contents of the MSR indexed by bits 31:0.

30.10.3 Using VM-Entry Controls for MSRs

A VMM may require specific MSRs to be loaded explicitly on VM entries while launching or resuming guest execution. The VM-entry MSR-load-address and VM-entry MSR-load-count entry control fields determine how MSRs are loaded on VM-entries. The VM-entry MSR-load-address and count fields are similar in structure and function to the VM-exit MSR-load address and count fields, except the MSR loading is done on VM-entries.

30.10.4 Handling Special-Case MSRs and Instructions

A number of instructions make use of designated MSRs in their operation. The VMM may need to consider saving the states of those MSRs. Instructions that merit such consideration include SYSENTER/SYSEXIT, SYSCALL/SYSRET, SWAPGS.

30.10.4.1 Handling IA32_EFER MSR

The IA32_EFER MSR includes bit fields that allow system software to enable processor features. For example: the SCE bit enables SYSCALL/SYSRET and the NXE bit enables the execute-disable bits in the paging-structure entries.

VMX provides hardware support to load the IA32_EFER MSR on VMX transitions and to save it on VM exits. Because of this, VMM software need not use the RDMSR and WRMSR instruction to give the register different values during host and guest execution.

30.10.4.2 Handling the SYSENTER and SYSEXIT Instructions

The SYSENTER and SYSEXIT instructions use three dedicated MSRs (IA32_SYSENTER_CS, IA32_SYSENTER_ESP and IA32_SYSENTER_EIP) to manage fast system calls. These MSRs may be utilized by both the VMM and the guest OS to manage system calls in VMX root operation and VMX non-root operation respectively.

VM entries load these MSRs from fields in the guest-state area of the VMCS. VM exits save the values of these MSRs into those fields and loads the MSRs from fields in the host-state area.

30.10.4.3 Handling the SYSCALL and SYSRET Instructions

The SYSCALL/SYSRET instructions are similar to SYSENTER/SYSEXIT but are designed to operate within the context of a 64-bit flat code segment. They are available only in 64-bit mode and only when the SCE bit of the IA32_EFER MSR is set. SYSCALL/SYSRET invocations can occur from either 32-bit compatibility mode application code or from 64-bit application code. Three related MSR registers (IA32_STAR, IA32_LSTAR, IA32_FMASK) are used in conjunction with fast system calls/returns that use these instructions.

64-Bit hosts which make use of these instructions in the VMM environment will need to save the guest state of the above registers on VM exit, load the host state, and restore the guest state on VM entry. One possible approach is to use the VM-exit MSR-save and MSR-load areas and the VM-entry MSR-load area defined by controls in the VMCS. A disadvantage to this approach, however, is that the approach results in the unconditional saving, loading, and restoring of MSR registers on each VM exit or VM entry.

Depending on the design of the VMM, it is likely that many VM-exits will require no fast system call support but the VMM will be burdened with the additional overhead of saving and restoring MSRs if the VMM chooses to support fast system call uniformly. Further, even if the host intends to support fast system calls during a VM-exit, some of the MSR values (such as the setting of the SCE bit in IA32_EFER) may not require modification as they may already be set to the appropriate value in the guest.

For performance reasons, a VMM may perform lazy save, load, and restore of these MSR values on certain VM exits when it is determined that this is acceptable. The lazy-save-load-restore operation can be carried out “manually” using RDMSR and WRMSR.

30.10.4.4 Handling the SWAPGS Instruction

The SWAPGS instruction is available only in 64-bit mode. It swaps the contents of two specific MSRs (IA32_GSBASE and IA32_KERNEL_GSBASE). The IA32_GSBASE MSR shadows the base address portion of the GS descriptor register; the IA32_KERNEL_GSBASE MSR holds the base address of the GS segment used by the kernel (typically it houses kernel structures). SWAPGS is intended for use with fast system calls when in 64-bit mode to allow immediate access to kernel structures on transition to kernel mode.

Similar to SYSCALL/SYSRET, IA-32e mode hosts which use fast system calls may need to save, load, and restore these MSR registers on VM exit and VM entry using the guidelines discussed in previous paragraphs.

30.10.4.5 Implementation Specific Behavior on Writing to Certain MSRs

As noted in Section 26.4 and Section 27.4, a processor may prevent writing to certain MSRs when loading guest states on VM entries or storing guest states on VM exits. This is done to ensure consistent operation. The subset and number of MSRs subject to restrictions are implementation specific. For initial VMX implementations, there are two MSRs: IA32_BIOS_UPDT_TRIG and IA32_BIOS_SIGN_ID (see Chapter 34).

30.10.5 Handling Accesses to Reserved MSR Addresses

Privileged software (either a VMM or a guest OS) can access a model specific register by specifying addresses in MSR address space. VMMs, however, must prevent a guest from accessing reserved MSR addresses in MSR address space.

Consult Chapter 34 for lists of supported MSRs and their usage. Use the MSR bitmap control to cause a VM exit when a guest attempts to access a reserved MSR address. The response to such a VM exit should be to reflect #GP(0) back to the guest.

30.11 HANDLING ACCESSES TO CONTROL REGISTERS

Bit fields in control registers (CR0, CR4) control various aspects of processor operation. The VMM must prevent guests from modifying bits in CR0 or CR4 that are reserved at the time the VMM is written.

Guest/host masks should be used by the VMM to cause VM exits when a guest attempts to modify reserved bits. Read shadows should be used to ensure that the guest always reads the reserved value (usually 0) for such bits. The VMM response to VM exits due to attempts from a guest to modify reserved bits should be to emulate the response which the processor would have normally produced (usually a #GP(0)).

30.12 PERFORMANCE CONSIDERATIONS

VMX provides hardware features that may be used for improving processor virtualization performance. VMMs must be designed to use this support properly. The basic idea behind most of these performance optimizations of the VMM is to reduce the number of VM exits while executing a guest VM.

This section lists ways that VMMs can take advantage of the performance enhancing features in VMX.

- **Read Access to Control Registers.** Analysis of common client workloads with common PC operating systems in a virtual machine shows a large number of VM-exits are caused by control register read accesses (particularly CR0). Reads of CR0 and CR4 does not cause VM exits. Instead, they return values from the CR0/CR4 read-shadows configured by the VMM in the guest controlling-VMCS with the guest-expected values.
- **Write Access to Control Registers.** Most VMM designs require only certain bits of the control registers to be protected from direct guest access. Write access to CR0/CR4 registers can be reduced by defining the host-owned and guest-owned bits in them through the CR0/CR4 host/guest masks in the VMCS. CR0/CR4 write values by the guest are qualified with the mask bits. If they change only guest-owned bits, they are allowed without causing VM exits. Any write that cause changes to host-owned bits cause VM exits and need to be handled by the VMM.
- **Access Rights based Page Table protection.** For VMM that implement access-rights-based page table protection, the VMCS provides a CR3 target value list that can be consulted by the processor to determine if a VM exit is required. Loading of CR3 with a value matching an entry in the CR3 target-list are allowed to proceed without VM exits. The VMM can utilize the CR3 target-list to save page-table hierarchies whose state is previously verified by the VMM.
- **Page-fault handling.** Another common cause for a VM exit is due to page-faults induced by guest address remapping done through virtual memory virtualization. VMX provides page-fault error-code mask and match fields in the VMCS to filter VM exits due to page-faults based on their cause (reflected in the error-code).

30.13 USE OF THE VMX-PREEMPTION TIMER

The VMX-preemption timer allows VMM software to preempt guest VM execution after a specified amount of time. Typical VMX-preemption timer usage is to program the initial VM quantum into the timer, save the timer value on each successive VM-exit (using the VM-exit control “save preemption timer value”) and run the VM until the timer expires.

In an alternative scenario, the VMM may use another timer (e.g. the TSC) to track the amount of time the VM has run while still using the VMX-preemption timer for VM preemption. In this scenario the VMM would not save the VMX-preemption timer on each VM-exit but instead would reload the VMX-preemption timer with initial VM quantum less the time the VM has already run. This scenario includes all the VM-entry and VM-exit latencies in the VM run time.

In both scenarios, on each successive VM-entry the VMX-preemption timer contains a smaller value until the VM quantum ends. If the VMX-preemption timer is loaded with a value smaller than the VM-entry latency then the VM will not execute any instructions before the timer expires. The VMM must ensure the initial VM quantum is greater than the VM-entry latency; otherwise the VM will make no forward progress.

CHAPTER 31
VIRTUALIZATION OF SYSTEM RESOURCES

31.1 OVERVIEW

When a VMM is hosting multiple guest environments (VMs), it must monitor potential interactions between software components using the same system resources. These interactions can require the virtualization of resources. This chapter describes the virtualization of system resources. These include: debugging facilities, address translation, physical memory, and microcode update facilities.

31.2 VIRTUALIZATION SUPPORT FOR DEBUGGING FACILITIES

The Intel 64 and IA-32 debugging facilities (see Chapter 17) provide breakpoint instructions, exception conditions, register flags, debug registers, control registers and storage buffers for functions related to debugging system and application software. In VMX operation, a VMM can support debugging system and application software from within virtual machines if the VMM properly virtualizes debugging facilities. The following list describes features relevant to virtualizing these facilities.

- The VMM can program the exception-bitmap (see Section 24.6.3) to ensure it gets control on debug functions (like breakpoint exceptions occurring while executing guest code such as INT3 instructions). Normally, debug exceptions modify debug registers (such as DR6, DR7, IA32_DEBUGCTL). However, if debug exceptions cause VM exits, exiting occurs before register modification.
- The VMM may utilize the VM-entry event injection facilities described in Section 26.5 to inject debug or breakpoint exceptions to the guest. See Section 31.2.1 for a more detailed discussion.
- The MOV-DR exiting control bit in the processor-based VM-execution control field (see Section 24.6.2) can be enabled by the VMM to cause VM exits on explicit guest access of various processor debug registers (for example, MOV to/from DR0-DR7). These exits would always occur on guest access of DR0-DR7 registers regardless of the values in CPL, DR4.DE or DR7.GD. Since all guest task switches cause VM exits, a VMM can control any indirect guest access or modification of debug registers during guest task switches.
- Guest software access to debug-related model-specific registers (such as IA32_DEBUGCTL MSR) can be trapped by the VMM through MSR access control features (such as the MSR-bitmaps that are part of processor-based VM-execution controls). See Section 30.10 for details on MSR virtualization.

- Debug registers such as DR7 and the IA32_DEBUGCTL MSR may be explicitly modified by the guest (through MOV-DR or WRMSR instructions) or modified implicitly by the processor as part of generating debug exceptions. The current values of DR7 and the IA32_DEBUGCTL MSR are saved to guest-state area of VMCS on every VM exit. Pending debug exceptions are debug exceptions that are recognized by the processor but not yet delivered. See Section 26.6.3 for details on pending debug exceptions.
- DR7 and the IA32-DEBUGCTL MSR are loaded from values in the guest-state area of the VMCS on every VM entry. This allows the VMM to properly virtualize debug registers when injecting debug exceptions to guest. Similarly, the RFLAGS[1] register is loaded on every VM entry (or pushed to stack if injecting a virtual event) from guest-state area of the VMCS. Pending debug exceptions are also loaded from guest-state area of VMCS so that they may be delivered after VM entry is completed.

31.2.1 Debug Exceptions

If a VMM emulates a guest instruction that would encounter a debug trap (single step or data or I/O breakpoint), it should cause that trap to be delivered. The VMM should not inject the debug exception using VM-entry event injection, but should set the appropriate bits in the pending debug exceptions field. This method will give the trap the right priority with respect to other events. (If the exception bitmap was programmed to cause VM exits on debug exceptions, the debug trap will cause a VM exit. At this point, the trap can be injected during VM entry with the proper priority.)

There is a valid pending debug exception if the BS bit (see Table 24-4) is set, regardless of the values of RFLAGS.TF or IA32_DEBUGCTL.BTF. The values of these bits do not impact the delivery of pending debug exceptions.

VMMs should exercise care when emulating a guest write (attempted using WRMSR) to IA32_DEBUGCTL to modify BTF if this is occurring with RFLAGS.TF = 1 and after a MOV SS or POP SS instruction (for example: while debug exceptions are blocked). Note the following:

- Normally, if WRMSR clears BTF while RFLAGS.TF = 1 and with debug exceptions blocked, a single-step trap will occur after WRMSR. A VMM emulating such an instruction should set the BS bit (see Table 24-4) in the pending debug exceptions field before VM entry.
- Normally, if WRMSR sets BTF while RFLAGS.TF = 1 and with debug exceptions blocked, neither a single-step trap nor a taken-branch trap can occur after WRMSR. A VMM emulating such an instruction should clear the BS bit (see Table 24-4) in the pending debug exceptions field before VM entry.

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.).

31.3 MEMORY VIRTUALIZATION

VMMs must control physical memory to ensure VM isolation and to remap guest physical addresses in host physical address space for virtualization. Memory virtualization allows the VMM to enforce control of physical memory and yet support guest OSs' expectation to manage memory address translation.

31.3.1 Processor Operating Modes & Memory Virtualization

Memory virtualization is required to support guest execution in various processor operating modes. This includes: protected mode with paging, protected mode with no paging, real-mode and any other transient execution modes. VMX allows guest operation in protected-mode with paging enabled and in virtual-8086 mode (with paging enabled) to support guest real-mode execution. Guest execution in transient operating modes (such as in real mode with one or more segment limits greater than 64-KByte) must be emulated by the VMM.

Since VMX operation requires processor execution in protected mode with paging (through CR0 and CR4 fixed bits), the VMM may utilize paging structures to support memory virtualization. To support guest real-mode execution, the VMM may establish a simple flat page table for guest linear to host physical address mapping. Memory virtualization algorithms may also need to capture other guest operating conditions (such as guest performing A20M# address masking) to map the resulting 20-bit effective guest physical addresses.

31.3.2 Guest & Host Physical Address Spaces

Memory virtualization provides guest software with contiguous guest physical address space starting zero and extending to the maximum address supported by the guest virtual processor's physical address width. The VMM utilizes guest physical to host physical address mapping to locate all or portions of the guest physical address space in host memory. The VMM is responsible for the policies and algorithms for this mapping which may take into account the host system physical memory map and the virtualized physical memory map exposed to a guest by the VMM. The memory virtualization algorithm needs to accommodate various guest memory uses (such as: accessing DRAM, accessing memory-mapped registers of virtual devices or core logic functions and so forth). For example:

- To support guest DRAM access, the VMM needs to map DRAM-backed guest physical addresses to host-DRAM regions. The VMM also requires the guest to host memory mapping to be at page granularity.
- Virtual devices (I/O devices or platform core logic) emulated by the VMM may claim specific regions in the guest physical address space to locate memory-mapped registers. Guest access to these virtual registers may be configured to cause page-fault induced VM-exits by marking these regions as always not

present. The VMM may handle these VM exits by invoking appropriate virtual device emulation code.

31.3.3 Virtualizing Virtual Memory by Brute Force

VMX provides the hardware features required to fully virtualize guest virtual memory accesses. VMX allows the VMM to trap guest accesses to the PAT (Page Attribute Table) MSR and the MTRR (Memory Type Range Registers). This control allows the VMM to virtualize the specific memory type of a guest memory. The VMM may control caching by controlling the guest CR0.CRD and CR0.NW bits, as well as by trapping guest execution of the INVD instruction. The VMM can trap guest CR3 loads and stores, and it may trap guest execution of INVLPG.

Because a VMM must retain control of physical memory, it must also retain control over the processor's address-translation mechanisms. Specifically, this means that only the VMM can access CR3 (which contains the base of the page directory) and can execute INVLPG (the only other instruction that directly manipulates the TLB).

At the same time that the VMM controls address translation, a guest operating system will also expect to perform normal memory management functions. It will access CR3, execute INVLPG, and modify (what it believes to be) page directories and page tables. Virtualization of address translation must tolerate and support guest attempts to control address translation.

A simple-minded way to do this would be to ensure that all guest attempts to access address-translation hardware trap to the VMM where such operations can be properly emulated. It must ensure that accesses to page directories and page tables also get trapped. This may be done by protecting these in-memory structures with conventional page-based protection. The VMM can do this because it can locate the page directory because its base address is in CR3 and the VMM receives control on any change to CR3; it can locate the page tables because their base addresses are in the page directory.

Such a straightforward approach is not necessarily desirable. Protection of the in-memory translation structures may be cumbersome. The VMM may maintain these structures with different values (e.g., different page base addresses) than guest software. This means that there must be traps on guest attempt to read these structures and that the VMM must maintain, in auxiliary data structures, the values to return to these reads. There must also be traps on modifications to these structures even if the translations they effect are never used. All this implies considerable overhead that should be avoided.

31.3.4 Alternate Approach to Memory Virtualization

Guest software is allowed to freely modify the guest page-table hierarchy without causing traps to the VMM. Because of this, the active page-table hierarchy might not always be consistent with the guest hierarchy. Any potential problems arising from inconsistencies can be solved using techniques analogous to those used by the processor and its TLB.

This section describes an alternative approach that allows guest software to freely access page directories and page tables. Traps occur on CR3 accesses and executions of INVLPG. They also occur when necessary to ensure that guest modifications to the translation structures actually take effect. The software mechanisms to support this approach are collectively called virtual TLB. This is because they emulate the functionality of the processor's physical translation look-aside buffer (TLB).

The basic idea behind the virtual TLB is similar to that behind the processor TLB. While the page-table hierarchy defines the relationship between physical to linear address, it does not directly control the address translation of each memory access. Instead, translation is controlled by the TLB, which is occasionally filled by the processor with translations derived from the page-table hierarchy. With a virtual TLB, the page-table hierarchy established by guest software (specifically, the guest operating system) does not control translation, either directly or indirectly. Instead, translation is controlled by the processor (through its TLB) and by the VMM (through a page-table hierarchy that it maintains).

Specifically, the VMM maintains an alternative page-table hierarchy that effectively caches translations derived from the hierarchy maintained by guest software. The remainder of this document refers to the former as the active page-table hierarchy (because it is referenced by CR3 and may be used by the processor to load its TLB) and the latter as the guest page-table hierarchy (because it is maintained by guest software). The entries in the active hierarchy may resemble the corresponding entries in the guest hierarchy in some ways and may differ in others.

Guest software is allowed to freely modify the guest page-table hierarchy without causing VM exits to the VMM. Because of this, the active page-table hierarchy might not always be consistent with the guest hierarchy. Any potential problems arising from any inconsistencies can be solved using techniques analogous to those used by the processor and its TLB. Note the following:

- Suppose the guest page-table hierarchy allows more access than active hierarchy (for example: there is a translation for a linear address in the guest hierarchy but not in the active hierarchy); this is analogous to a situation in which the TLB allows less access than the page-table hierarchy. If an access occurs that would be allowed by the guest hierarchy but not the active one, a page fault occurs; this is analogous to a TLB miss. The VMM gains control (as it handles all page faults) and can update the active page-table hierarchy appropriately; this corresponds to a TLB fill.
- Suppose the guest page-table hierarchy allows less access than the active hierarchy; this is analogous to a situation in which the TLB allows more access than the page-table hierarchy. This situation can occur only if the guest operating system has modified a page-table entry to reduce access (for example: by marking it not-present). Because the older, more permissive translation may have been cached in the TLB, the processor is architecturally permitted to use the older translation and allow more access. Thus, the VMM may (through the active page-table hierarchy) also allow greater access. For the new, less permissive

translation to take effect, guest software should flush any older translations from the TLB either by executing INVLPG or by loading CR3. Because both these operations will cause a trap to the VMM, the VMM will gain control and can remove from the active page-table hierarchy the translations indicated by guest software (the translation of a specific linear address for INVLPG or all translations for a load of CR3).

As noted previously, the processor reads the page-table hierarchy to cache translations in the TLB. It also writes to the hierarchy to main the accessed (A) and dirty (D) bits in the PDEs and PTEs. The virtual TLB emulates this behavior as follows:

- When a page is accessed by guest software, the A bit in the corresponding PTE (or PDE for a 4-MByte page) in the active page-table hierarchy will be set by the processor (the same is true for PDEs when active page tables are accessed by the processor). For guest software to operate properly, the VMM should update the A bit in the guest entry at this time. It can do this reliably if it keeps the active PTE (or PDE) marked not-present until it has set the A bit in the guest entry.
- When a page is written by guest software, the D bit in the corresponding PTE (or PDE for a 4-MByte page) in the active page-table hierarchy will be set by the processor. For guest software to operate properly, the VMM should update the D bit in the guest entry at this time. It can do this reliably if it keeps the active PTE (or PDE) marked read-only until it has set the D bit in the guest entry. This solution is valid for guest software running at privilege level 3; support for more privileged guest software is described in Section 31.3.5.

31.3.5 Details of Virtual TLB Operation

This section describes in more detail how a VMM could support a virtual TLB. It explains how an active page-table hierarchy is initialized and how it is maintained in response to page faults, uses of INVLPG, and accesses to CR3. The mechanisms described here are the minimum necessary. They may not result in the best performance.

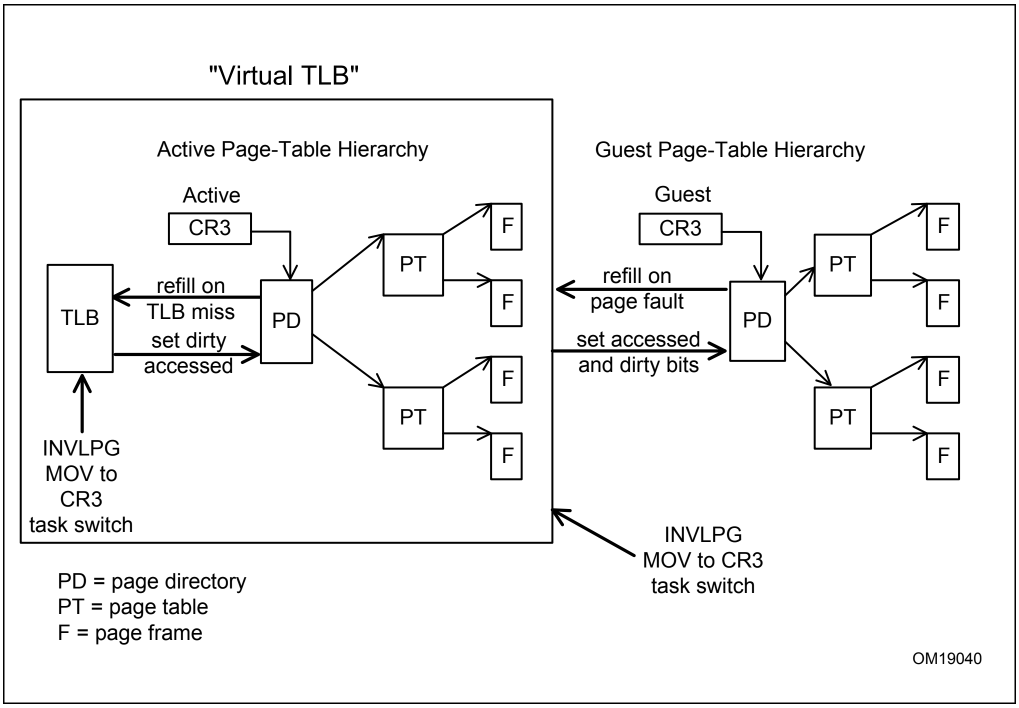

Figure 31-1. Virtual TLB Scheme

As noted above, the VMM maintains an active page-table hierarchy for each virtual machine that it supports. It also maintains, for each machine, values that the machine expects for control registers CR0, CR2, CR3, and CR4 (they control address translation). These values are called the guest control registers.

In general, the VMM selects the physical-address space that is allocated to guest software. The term guest address refers to an address installed by guest software in the guest CR3, in a guest PDE (as a page table base address or a page base address), or in a guest PTE (as a page base address). While guest software considers these to be specific physical addresses, the VMM may map them differently.

31.3.5.1 Initialization of Virtual TLB

To enable the Virtual TLB scheme, the VMCS must be set up to trigger VM exits on:

- All writes to CR3 (the CR3-target count should be 0) or the paging-mode bits in CR0 and CR4 (using the CR0 and CR4 guest/host masks)
- Page-fault (#PF) exceptions
- Execution of INVLPG

When guest software first enables paging, the VMM creates an aligned 4-KByte active page directory that is invalid (all entries marked not-present). This invalid directory is analogous to an empty TLB.

31.3.5.2 Response to Page Faults

Page faults can occur for a variety of reasons. In some cases, the page fault alerts the VMM to an inconsistency between the active and guest page-table hierarchy. In such cases, the VMM can update the former and re-execute the faulting instruction. In other cases, the hierarchies are already consistent and the fault should be handled by the guest operating system. The VMM can detect this and use an established mechanism for raising a page fault to guest software.

The VMM can handle a page fault by following these steps (The steps below assume the guest is operating in a paging mode without PAE. Analogous steps to handle address translation using PAE or four-level paging mechanisms can be derived by VMM developers according to the paging behavior defined in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*):

1. First consult the active PDE, which can be located using the upper 10 bits of the faulting address and the current value of CR3. The active PDE is the source of the fault if it is marked not present or if its R/W bit and U/S bits are inconsistent with the attempted guest access (the guest privilege level and the values of CR0.WP and CR4.SMEP should also be taken into account).
2. If the active PDE is the source of the fault, consult the corresponding guest PDE using the same 10 bits from the faulting address and the physical address that corresponds to the guest address in the guest CR3. If the guest PDE would cause a page fault (for example: it is marked not present), then raise a page fault to the guest operating system.

   The following steps assume that the guest PDE would not have caused a page fault.
3. If the active PDE is the source of the fault and the guest PDE contains, as page-table base address (if PS = 0) or page base address (PS = 1), a guest address that the VMM has chosen not to support; then raise a machine check (or some other abort) to the guest operating system.

   The following steps assume that the guest address in the guest PDE is supported for the virtual machine.
4. If the active PDE is marked not-present, then set the active PDE to correspond to guest PDE as follows:
   a. If the active PDE contains a page-table base address (if PS = 0), then allocate an aligned 4-KByte active page table marked completely invalid and set the page-table base address in the active PDE to be the physical address of the newly allocated page table.

b. If the active PDE contains a page base address (if PS = 1), then set the page base address in the active PDE to be the physical page base address that corresponds to the guest address in the guest PDE.

c. Set the P, U/S, and PS bits in the active PDE to be identical to those in the guest PDE.

d. Set the PWT, PCD, and G bits according to the policy of the VMM.

e. Set A = 1 in the guest PDE.

f. If D = 1 in the guest PDE or PS = 0 (meaning that this PDE refers to a page table), then set the R/W bit in the active PDE as in the guest PDE.

g. If D = 0 in the guest PDE, PS = 1 (this is a 4-MByte page), and the attempted access is a write; then set R/W in the active PDE as in the guest PDE and set D = 1 in the guest PDE.

h. If D = 0 in the guest PDE, PS = 1, and the attempted access is not a write; then set R/W = 0 in the active PDE.

i. After modifying the active PDE, re-execute the faulting instruction.

The remaining steps assume that the active PDE is already marked present.

5. If the active PDE is the source of the fault, the active PDE refers to a 4-MByte page (PS = 1), the attempted access is a write; D = 0 in the guest PDE, and the active PDE has caused a fault solely because it has R/W = 0; then set R/W in the active PDE as in the guest PDE; set D = 1 in the guest PDE, and re-execute the faulting instruction.

6. If the active PDE is the source of the fault and none of the above cases apply, then raise a page fault of the guest operating system.

   The remaining steps assume that the source of the original page fault is not the active PDE.

NOTE

It is possible that the active PDE might be causing a fault even though the guest PDE would not. However, this can happen only if the guest operating system increased access in the guest PDE and did not take action to ensure that older translations were flushed from the TLB. Such translations might have caused a page fault if the guest software were running on bare hardware.

7. If the active PDE refers to a 4-MByte page (PS = 1) but is not the source of the fault, then the fault resulted from an inconsistency between the active page-table hierarchy and the processor’s TLB. Since the transition to the VMM caused an address-space change and flushed the processor’s TLB, the VMM can simply re-execute the faulting instruction.

   The remaining steps assume that PS = 0 in the active and guest PDEs.

8. Consult the active PTE, which can be located using the next 10 bits of the faulting address (bits 21–12) and the physical page-table base address in the active PDE. The active PTE is the source of the fault if it is marked not-present or if its R/W bit and U/S bits are inconsistent with the attempted guest access (the guest privilege level and the values of CR0.WP and CR4.SMEP should also be taken into account).
9. If the active PTE is not the source of the fault, then the fault has resulted from an inconsistency between the active page-table hierarchy and the processor's TLB. Since the transition to the VMM caused an address-space change and flushed the processor's TLB, the VMM simply re-executes the faulting instruction.

   The remaining steps assume that the active PTE is the source of the fault.
10. Consult the corresponding guest PTE using the same 10 bits from the faulting address and the physical address that correspond to the guest page-table base address in the guest PDE. If the guest PTE would cause a page fault (it is marked not-present), the raise a page fault to the guest operating system.

    The following steps assume that the guest PTE would not have caused a page fault.
11. If the guest PTE contains, as page base address, a physical address that is not valid for the virtual machine being supported; then raise a machine check (or some other abort) to the guest operating system.

    The following steps assume that the address in the guest PTE is valid for the virtual machine.
12. If the active PTE is marked not-present, then set the active PTE to correspond to guest PTE:
    a. Set the page base address in the active PTE to be the physical address that corresponds to the guest page base address in the guest PTE.
    b. Set the P, U/S, and PS bits in the active PTE to be identical to those in the guest PTE.
    c. Set the PWT, PCD, and G bits according to the policy of the VMM.
    d. Set A = 1 in the guest PTE.
    e. If D = 1 in the guest PTE, then set the R/W bit in the active PTE as in the guest PTE.
    f. If D = 0 in the guest PTE and the attempted access is a write, then set R/W in the active PTE as in the guest PTE and set D = 1 in the guest PTE.
    g. If D = 0 in the guest PTE and the attempted access is not a write, then set R/W = 0 in the active PTE.
    h. After modifying the active PTE, re-execute the faulting instruction.

    The remaining steps assume that the active PTE is already marked present.
13. If the attempted access is a write, D = 0 (not dirty) in the guest PTE and the active PTE has caused a fault solely because it has R/W = 0 (read-only); then set

R/W in the active PTE as in the guest PTE, set D = 1 in the guest PTE and re-execute the faulting instruction.

14. If none of the above cases apply, then raise a page fault of the guest operating system.

31.3.5.3 Response to Uses of INVLPG

Operating-systems can use INVLPG to flush entries from the TLB. This instruction takes a linear address as an operand and software expects any cached translations for the address to be flushed. A VMM should set the processor-based VM-execution control "INVLPG exiting" to 1 so that any attempts by a privileged guest to execute INVLPG will trap to the VMM. The VMM can then modify the active page-table hierarchy to emulate the desired effect of the INVLPG.

The following steps are performed. Note that these steps are performed only if the guest invocation of INVLPG would not fault and only if the guest software is running at privilege level 0:

1. Locate the relevant active PDE using the upper 10 bits of the operand address and the current value of CR3. If the PDE refers to a 4-MByte page (PS = 1), then set P = 0 in the PDE.
2. If the PDE is marked present and refers to a page table (PS = 0), locate the relevant active PTE using the next 10 bits of the operand address (bits 21–12) and the page-table base address in the PDE. Set P = 0 in the PTE. Examine all PTEs in the page table; if they are now all marked not-present, de-allocate the page table and set P = 0 in the PDE (this step may be optional).

31.3.5.4 Response to CR3 Writes

A guest operating system may attempt to write to CR3. Any write to CR3 implies a TLB flush and a possible page table change. The following steps are performed:

1. The VMM notes the new CR3 value (used later to walk guest page tables) and emulates the write.
2. The VMM allocates a new PD page, with all invalid entries.
3. The VMM sets actual processor CR3 register to point to the new PD page.

The VMM may, at this point, speculatively fill in VTLB mappings for performance reasons.

31.4 MICROCODE UPDATE FACILITY

The microcode code update facility may be invoked at various points during the operation of a platform. Typically, the BIOS invokes the facility on all processors during the BIOS boot process. This is sufficient to boot the BIOS and operating system. As a microcode update more current than the system BIOS may be available, system software should provide another mechanism for invoking the microcode update facility. The implications of the microcode update mechanism on the design of the VMM are described in this section.

NOTE

Microcode updates must not be performed during VMX non-root operation. Updates performed in VMX non-root operation may result in unpredictable system behavior.

31.4.1 Early Load of Microcode Updates

The microcode update facility may be invoked early in the VMM or guest OS boot process. Loading the microcode update early provides the opportunity to correct errata affecting the boot process but the technique generally requires a reboot of the software.

A microcode update may be loaded from the OS or VMM image loader. Typically, such image loaders do not run on every logical processor, so this method effects only one logical processor. Later in the VMM or OS boot process, after bringing all application processors on-line, the VMM or OS needs to invoke the microcode update facility for all application processors.

Depending on the order of the VMM and the guest OS boot, the microcode update facility may be invoked by the VMM or the guest OS. For example, if the guest OS boots first and then loads the VMM, the guest OS may invoke the microcode update facility on all the logical processors. If a VMM boots before its guests, then the VMM may invoke the microcode update facility during its boot process. In both cases, the VMM or OS should invoke the microcode update facilities soon after performing the multiprocessor startup.

In the early load scenario, microcode updates may be contained in the VMM or OS image or, the VMM or OS may manage a separate database or file of microcode updates. Maintaining a separate microcode update image database has the advantage of reducing the number of required VMM or OS releases as a result of microcode update releases.

31.4.2 Late Load of Microcode Updates

A microcode update may be loaded during normal system operation. This allows system software to activate the microcode update at anytime without requiring a system reboot. This scenario does not allow the microcode update to correct errata which affect the processor’s boot process but does allow high-availability systems to activate microcode updates without interrupting the availability of the system. In this late load scenario, either the VMM or a designated guest may load the microcode update. If the guest is loading the microcode update, the VMM must make sure that the entire guest memory buffer (which contains the microcode update image) will not cause a page fault when accessed.

If the VMM loads the microcode update, then the VMM must have access to the current set of microcode updates. These updates could be part of the VMM image or could be contained in a separate microcode update image database (for example: a database file on disk or in memory). Again, maintaining a separate microcode update image database has the advantage of reducing the number of required VMM or OS releases as a result of microcode update releases.

The VMM may wish to prevent a guest from loading a microcode update or may wish to support the microcode update requested by a guest using emulation (without actually loading the microcode update). To prevent microcode update loading, the VMM may return a microcode update signature value greater than the value of IA32_BIOS_SIGN_ID MSR. A well behaved guest will not attempt to load an older microcode update. The VMM may also drop the guest attempts to write to IA32_BIOS_UPDT_TRIG MSR, preventing the guest from loading any microcode updates. Later, when the guest queries IA32_BIOS_SIGN_ID MSR, the VMM could emulate the microcode update signature that the guest expects.

In general, loading a microcode update later will limit guest software's visibility of features that may be enhanced by a microcode update.

CHAPTER 32 HANDLING BOUNDARY CONDITIONS IN A VIRTUAL MACHINE MONITOR

32.1 OVERVIEW

This chapter describes what a VMM must consider when handling exceptions, interrupts, error conditions, and transitions between activity states.

32.2 INTERRUPT HANDLING IN VMX OPERATION

The following bullets summarize VMX support for handling interrupts:

- **Control of processor exceptions**. The VMM can get control on specific guest exceptions through the exception-bitmap in the guest controlling VMCS. The exception bitmap is a 32-bit field that allows the VMM to specify processor behavior on specific exceptions (including traps, faults, and aborts). Setting a specific bit in the exception bitmap implies VM exits will be generated when the corresponding exception occurs. Any exceptions that are programmed not to cause VM exits are delivered directly to the guest through the guest IDT. The exception bitmap also controls execution of relevant instructions such as BOUND, INTO and INT3. VM exits on page-faults are treated in such a way the page-fault error code is qualified through the page-fault-error-code mask and match fields in the VMCS.
- **Control over triple faults**. If a fault occurs while attempting to call a double-fault handler in the guest and that fault is not configured to cause a VM exit in the exception bitmap, the resulting triple fault causes a VM exit.
- **Control of external interrupts**. VMX allows both host and guest control of external interrupts through the "external-interrupt exiting" VM execution control. If the control is 0, external-interrupts do not cause VM exits and the interrupt delivery is masked by the guest programmed RFLAGS.IF value.[1] If the control is 1, external-interrupts causes VM exits and are not masked by RFLAGS.IF. The VMM can identify VM exits due to external interrupts by checking the exit reason for an "external interrupt" (value = 1).

1. This chapter uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel 64 architecture. For processors that do not support Intel 64 architecture, this notation refers to the 32-bit forms of those registers (EAX, EIP, ESP, EFLAGS, etc.).

- **Control of other events**. There is a pin-based VM-execution control that controls system behavior (exit or no-exit) for NMI events. Most VMM usages will need handling of NMI external events in the VMM and hence will specify host control of these events.

  Some processors also support a pin-based VM-execution control called "virtual NMIs." When this control is set, NMIs cause VM exits, but the processor tracks guest readiness for virtual NMIs. This control interacts with the "NMI-window exiting" VM-execution control (see below).

  INIT and SIPI events always cause VM exits.
- **Acknowledge interrupt on exit**. The "acknowledge interrupt on exit" VM-exit control in the controlling VMCS controls processor behavior for external interrupt acknowledgement. If the control is 1, the processor acknowledges the interrupt controller to acquire the interrupt vector upon VM exit, and stores the vector in the VM-exit interruption-information field. If the control is 0, the external interrupt is not acknowledged during VM exit. Since RFLAGS.IF is automatically cleared on VM exits due to external interrupts, VMM re-enabling of interrupts (setting RFLAGS.IF = 1) initiates the external interrupt acknowledgement and vectoring of the external interrupt through the monitor/host IDT.
- **Event-masking Support**. VMX captures the masking conditions of specific events while in VMX non-root operation through the interruptibility-state field in the guest-state area of the VMCS.

  This feature allows proper virtualization of various interrupt blocking states, such as: (a) blocking of external interrupts for the instruction following STI; (b) blocking of interrupts for the instruction following a MOV-SS or POP-SS instruction; (c) SMI blocking of subsequent SMIs until the next execution of RSM; and (d) NMI/SMI blocking of NMIs until the next execution of IRET or RSM.

  INIT and SIPI events are treated specially. INIT assertions are always blocked in VMX root operation and while in SMM, and unblocked otherwise. SIPI events are always blocked in VMX root operation.

  The interruptibility state is loaded from the VMCS guest-state area on every VM entry and saved into the VMCS on every VM exit.
- **Event injection**. VMX operation allows injecting interruptions to a guest virtual machine through the use of VM-entry interrupt-information field in VMCS. Injectable interruptions include external interrupts, NMI, processor exceptions, software generated interrupts, and software traps. If the interrupt-information field indicates a valid interrupt, exception or trap event upon the next VM entry; the processor will use the information in the field to vector a virtual interruption through the guest IDT after all guest state and MSRs are loaded. Delivery through the guest IDT emulates vectoring in non-VMX operation by doing the normal privilege checks and pushing appropriate entries to the guest stack (entries may include RFLAGS, EIP and exception error code). A VMM with host control of NMI and external interrupts can use the event-injection facility to forward virtual interruptions to various guest virtual machines.

- **Interrupt-window exiting**. When set to 1, the “interrupt-window exiting” VM-execution control (Section 24.6.2) causes VM exits when guest RFLAGS.IF is 1 and no other conditions block external interrupts. A VM exit occurs at the beginning of any instruction at which RFLAGS.IF = 1 and on which the interruptibility state of the guest would allow delivery of an interrupt. For example: when the guest executes an STI instruction, RFLAGS = 1, and if at the completion of next instruction the interruptibility state masking due to STI is removed; a VM exit occurs if the “interrupt-window exiting” VM-execution control is 1. This feature allows a VMM to queue a virtual interrupt to the guest when the guest is not in an interruptible state. The VMM can set the “interrupt-window exiting” VM-execution control for the guest and depend on a VM exit to know when the guest becomes interruptible (and, therefore, when it can inject a virtual interrupt). The VMM can detect such VM exits by checking for the basic exit reason “interrupt-window” (value = 7). If this feature is not used, the VMM will need to poll and check the interruptibility state of the guest to deliver virtual interrupts.
- **NMI-window exiting**. If the “virtual NMIs” VM-execution is set, the processor tracks virtual-NMI blocking. The “NMI-window exiting” VM-execution control (Section 24.6.2) causes VM exits when there is no virtual-NMI blocking. For example, after execution of the IRET instruction, a VM exit occurs if the “NMI-window exiting” VM-execution control is 1. This feature allows a VMM to queue a virtual NMI to a guest when the guest is not ready to receive NMIs. The VMM can set the “NMI-window exiting” VM-execution control for the guest and depend on a VM exit to know when the guest becomes ready for NMIs (and, therefore, when it can inject a virtual NMI). The VMM can detect such VM exits by checking for the basic exit reason “NMI window” (value = 8). If this feature is not used, the VMM will need to poll and check the interruptibility state of the guest to deliver virtual NMIs.
- **VM-exit information**. The VM-exit information fields provide details on VM exits due to exceptions and interrupts. This information is provided through the exit-qualification, VM-exit-interruption-information, instruction-length and interruption-error-code fields. Also, for VM exits that occur in the course of vectoring through the guest IDT, information about the event that was being vectored through the guest IDT is provided in the IDT-vectoring-information and IDT-vectoring-error-code fields. These information fields allow the VMM to identify the exception cause and to handle it properly.

32.3 EXTERNAL INTERRUPT VIRTUALIZATION

VMX operation allows both host and guest control of external interrupts. While guest control of external interrupts might be suitable for partitioned usages (different CPU cores/threads and I/O devices partitioned to independent virtual machines), most VMMs built upon VMX are expected to utilize host control of external interrupts. The rest of this section describes a general host-controlled interrupt virtualization architecture for standard PC platforms through the use of VMX supported features.

With host control of external interrupts, the VMM (or the host OS in a hosted VMM model) manages the physical interrupt controllers in the platform and the interrupts generated through them. The VMM exposes software-emulated virtual interrupt controller devices (such as PIC and APIC) to each guest virtual machine instance.

32.3.1 Virtualization of Interrupt Vector Space

The Intel 64 and IA-32 architectures use 8-bit vectors of which 224 (20H – FFH) are available for external interrupts. Vectors are used to select the appropriate entry in the interrupt descriptor table (IDT). VMX operation allows each guest to control its own IDT. Host vectors refer to vectors delivered by the platform to the processor during the interrupt acknowledgement cycle. Guest vectors refer to vectors programmed by a guest to select an entry in its guest IDT. Depending on the I/O resource management models supported by the VMM design, the guest vector space may or may not overlap with the underlying host vector space.

- Interrupts from virtual devices: Guest vector numbers for virtual interrupts delivered to guests on behalf of emulated virtual devices have no direct relation to the host vector numbers of interrupts from physical devices on which they are emulated. A guest-vector assigned for a virtual device by the guest operating environment is saved by the VMM and utilized when injecting virtual interrupts on behalf of the virtual device.
- Interrupts from assigned physical devices: Hardware support for I/O device assignment allows physical I/O devices in the host platform to be assigned (direct-mapped) to VMs. Guest vectors for interrupts from direct-mapped physical devices take up equivalent space from the host vector space, and require the VMM to perform host-vector to guest-vector mapping for interrupts.

Figure 32-1 illustrates the functional relationship between host external interrupts and guest virtual external interrupts. Device A is owned by the host and generates external interrupts with host vector X. The host IDT is set up such that the interrupt service routine (ISR) for device driver A is hooked to host vector X as normal. VMM emulates (over device A) virtual device C in software which generates virtual interrupts to the VM with guest expected vector P. Device B is assigned to a VM and generates external interrupts with host vector Y. The host IDT is programmed to hook the VMM interrupt service routine (ISR) for assigned devices for vector Y, and the VMM handler injects virtual interrupt with guest vector Q to the VM. The guest operating system programs the guest to hook appropriate guest driver's ISR to vectors P and Q.

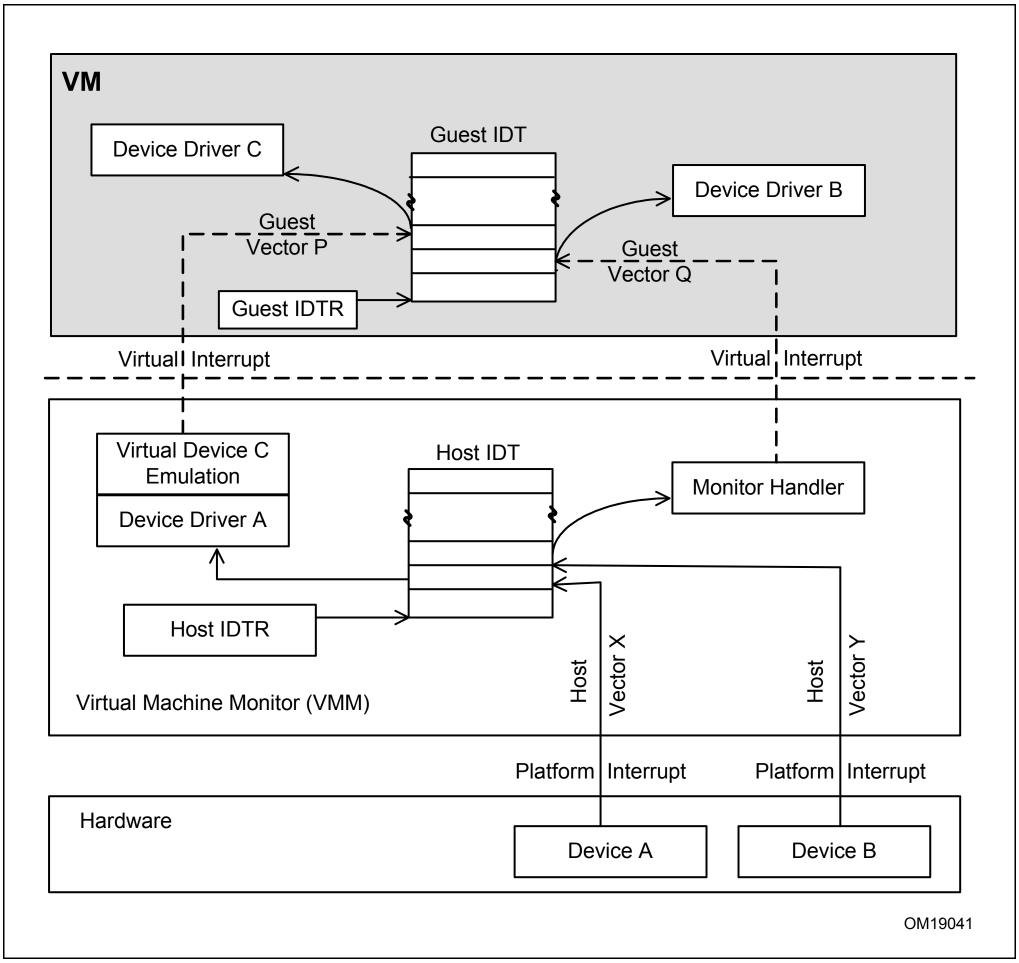

Figure 32-1. Host External Interrupts and Guest Virtual Interrupts

32.3.2 Control of Platform Interrupts

To meet the interrupt virtualization requirements, the VMM needs to take ownership of the physical interrupts and the various interrupt controllers in the platform. VMM control of physical interrupts may be enabled through the host-control settings of the "external-interrupt exiting" VM-execution control. To take ownership of the platform interrupt controllers, the VMM needs to expose the virtual interrupt controller devices to the virtual machines and restrict guest access to the platform interrupt controllers.

Intel 64 and IA-32 platforms can support three types of external interrupt control mechanisms: Programmable Interrupt Controllers (PIC), Advanced Programmable

Interrupt Controllers (APIC), and Message Signaled Interrupts (MSI). The following sections provide information on the virtualization of each of these mechanisms.

32.3.2.1 PIC Virtualization

Typical PIC-enabled platform implementations support dual 8259 interrupt controllers cascaded as master and slave controllers. They supporting up to 15 possible interrupt inputs. The 8259 controllers are programmed through initialization command words (ICWx) and operation command words (OCWx) accessed through specific I/O ports. The various interrupt line states are captured in the PIC through interrupt requests, interrupt service routines and interrupt mask registers.

Guest access to the PIC I/O ports can be restricted by activating I/O bitmaps in the guest controlling-VMCS (activate-I/O-bitmap bit in VM-execution control field set to 1) and pointing the I/O-bitmap physical addresses to valid bitmap regions. Bits corresponding to the PIC I/O ports can be cleared to cause a VM exit on guest access to these ports.

If the VMM is not supporting direct access to any I/O ports from a guest, it can set the unconditional-I/O-exiting in the VM-execution control field instead of activating I/O bitmaps. The exit-reason field in VM-exit information allows identification of VM exits due to I/O access and can provide an exit-qualification to identify details about the guest I/O operation that caused the VM exit.

The VMM PIC virtualization needs to emulate the platform PIC functionality including interrupt priority, mask, request and service states, and specific guest programmed modes of PIC operation.

32.3.2.2 xAPIC Virtualization

Most modern Intel 64 and IA-32 platforms include support for an APIC. While the standard PIC is intended for use on uniprocessor systems, APIC can be used in either uniprocessor or multi-processor systems.

APIC based interrupt control consists of two physical components: the interrupt acceptance unit (Local APIC) which is integrated with the processor, and the interrupt delivery unit (I/O APIC) which is part of the I/O subsystem. APIC virtualization involves protecting the platform’s local and I/O APICs and emulating them for the guest.

32.3.2.3 Local APIC Virtualization

The local APIC is responsible for the local interrupt sources, interrupt acceptance, dispensing interrupts to the logical processor, and generating inter-processor interrupts. Software interacts with the local APIC by reading and writing its memory-mapped registers residing within a 4-KByte uncached memory region with base address stored in the IA32_APIC_BASE MSR. Since the local APIC registers are memory-mapped, the VMM can utilize memory virtualization techniques (such as page-table virtualization) to trap guest accesses to the page frame hosting the virtual local APIC registers.

Local APIC virtualization in the VMM needs to emulate the various local APIC operations and registers, such as: APIC identification/format registers, the local vector table (LVT), the interrupt command register (ICR), interrupt capture registers (TMR, IRR and ISR), task and processor priority registers (TPR, PPR), the EOI register and the APIC-timer register. Since local APICs are designed to operate with non-specific EOI, local APIC emulation also needs to emulate broadcast of EOI to the guest's virtual I/O APICs for level triggered virtual interrupts.

A local APIC allows interrupt masking at two levels: (1) mask bit in the local vector table entry for local interrupts and (2) raising processor priority through the TPR registers for masking lower priority external interrupts. The VMM needs to comprehend these virtual local APIC mask settings as programmed by the guest in addition to the guest virtual processor interruptibility state (when injecting APIC routed external virtual interrupts to a guest VM).

VMX provides several features which help the VMM to virtualize the local APIC. These features allow many of guest TPR accesses (using CR8 only) to occur without VM exits to the VMM:

- The VMCS contains a "virtual-APIC address" field. This 64-bit field is the physical address of the 4-KByte virtual APIC page (4-KByte aligned). The virtual-APIC page contains a TPR shadow, which is accessed by the MOV CR8 instruction. The TPR shadow comprises bits 7:4 in byte 80H of the virtual-APIC page.
- The TPR threshold: bits 3:0 of this 32-bit field determine the threshold below which the TPR shadow cannot fall. A VM exit will occur after an execution of MOV CR8 that reduces the TPR shadow below this value.
- The processor-based VM-execution controls field contains a "use TPR shadow" bit and a "CR8-store exiting" bit. If the "use TPR shadow" VM-execution control is 1 and the "CR8-store exiting" VM-execution control is 0, then a MOV from CR8 reads from the TPR shadow. If the "CR8-store exiting" VM-execution control is 1, then MOV from CR8 causes a VM exit; the "use TPR shadow" VM-execution control is ignored in this case.
- The processor-based VM-execution controls field contains a "CR8-load exiting" bit. If the "use TPR shadow" VM-execution control is set and the "CR8-load exiting" VM-execution control is clear, then MOV to CR8 writes to the "TPR shadow". A VM exit will occur after this write if the value written is below the TPR threshold. If the "CR8-load exiting" VM-execution control is set, then MOV to CR8 causes a VM exit; the "use TPR shadow" VM-execution control is ignored in this case.

32.3.2.4 I/O APIC Virtualization

The I/O APIC registers are typically mapped to a 1 MByte region where each I/O APIC is allocated a 4K address window within this range. The VMM may utilize physical memory virtualization to trap guest accesses to the virtual I/O APIC memory-mapped registers. The I/O APIC virtualization needs to emulate the various I/O APIC operations and registers such as identification/version registers, indirect-I/O-access registers, EOI register, and the I/O redirection table. I/O APIC virtualization also need to emulate various redirection table entry settings such as delivery mode, destination mode, delivery status, polarity, masking, and trigger mode programmed by the guest and track remote-IRR state on guest EOI writes to various virtual local APICs.

32.3.2.5 Virtualization of Message Signaled Interrupts

The *PCI Local Bus Specification* (Rev. 2.2) introduces the concept of message signaled interrupts (MSI). MSI enable PCI devices to request service by writing a system-specified message to a system specified address. The transaction address specifies the message destination while the transaction data specifies the interrupt vector, trigger mode and delivery mode. System software is expected to configure the message data and address during MSI device configuration, allocating one or more no-shared messages to MSI capable devices. Chapter 10, "Advanced Programmable Interrupt Controller (APIC)," specifies the MSI message address and data register formats to be followed on Intel 64 and IA-32 platforms. While MSI is optional for conventional PCI devices, it is the preferred interrupt mechanism for PCI-Express devices.

Since the MSI address and data are configured through PCI configuration space, to control these physical interrupts the VMM needs to assume ownership of PCI configuration space. This allows the VMM to capture the guest configuration of message address and data for MSI-capable virtual and assigned guest devices. PCI configuration transactions on PC-compatible systems are generated by software through two different methods:

1. The standard CONFIG_ADDRESS/CONFIG_DATA register mechanism (CFCH/CF8H ports) as defined in the *PCI Local Bus Specification.*
2. The enhanced flat memory-mapped (MEMCFG) configuration mechanism as defined in the *PCI-Express Base Specification* (Rev. 1.0a.).

The CFCH/CF8H configuration access from guests can be trapped by the VMM through use of I/O-bitmap VM-execution controls. The memory-mapped PCI-Express MEMCFG guest configuration accesses can be trapped by VMM through physical memory virtualization.

32.3.3 Examples of Handling of External Interrupts

The following sections illustrate interrupt processing in a VMM (when used to support the external interrupt virtualization requirements).

32.3.3.1 Guest Setup

The VMM sets up the guest to cause a VM exit to the VMM on external interrupts. This is done by setting the “external-interrupt exiting” VM-execution control in the guest controlling-VMCS.

32.3.3.2 Processor Treatment of External Interrupt

Interrupts are automatically masked by hardware in the processor on VM exit by clearing RFLAGS.IF. The exit-reason field in VMCS is set to 1 to indicate an external interrupt as the exit reason.

If the VMM is utilizing the acknowledge-on-exit feature (by setting the “acknowledge interrupt on exit” VM-exit control), the processor acknowledges the interrupt, retrieves the host vector, and saves the interrupt in the VM-exit-interruption-information field (in the VM-exit information region of the VMCS) before transitioning control to the VMM.

32.3.3.3 Processing of External Interrupts by VMM

Upon VM exit, the VMM can determine the exit cause of an external interrupt by checking the exit-reason field (value = 1) in VMCS. If the acknowledge-interrupt-on-exit control (see Section 24.7.1) is enabled, the VMM can use the saved host vector (in the exit-interruption-information field) to switch to the appropriate interrupt handler. If the “acknowledge interrupt on exit” VM-exit control is 0, the VMM may re-enable interrupts (by setting RFLAGS.IF) to allow vectoring of external interrupts through the monitor/host IDT.

The following steps may need to be performed by the VMM to process an external interrupt:

- **Host Owned I/O Devices:** For host-owned I/O devices, the interrupting device is owned by the VMM (or hosting OS in a hosted VMM). In this model, the interrupt service routine in the VMM/host driver is invoked and, upon ISR completion, the appropriate write sequences (TPR updates, EOI etc.) to respective interrupt controllers are performed as normal. If the work completion indicated by the driver implies virtual device activity, the VMM runs the virtual device emulation. Depending on the device class, physical device activity could imply activity by multiple virtual devices mapped over the device. For each affected virtual device, the VMM injects a virtual external interrupt event to respective guest virtual machines. The guest driver interacts with the emulated virtual device to process the virtual interrupt. The interrupt controller emulation in the VMM supports various guest accesses to the VMM’s virtual interrupt controller.
- **Guest Assigned I/O Devices:** For assigned I/O devices, either the VMM uses a software proxy or it can directly map the physical device to the assigned VM. In both cases, servicing of the interrupt condition on the physical device is initiated by the driver running inside the guest VM. With host control of external

interrupts, interrupts from assigned physical devices cause VM exits to the VMM and vectoring through the host IDT to the registered VMM interrupt handler. To unblock delivery of other low priority platform interrupts, the VMM interrupt handler must mask the interrupt source (for level triggered interrupts) and issue the appropriate EOI write sequences.

Once the physical interrupt source is masked and the platform EOI generated, the VMM can map the host vector to its corresponding guest vector to inject the virtual interrupt into the assigned VM. The guest software does EOI write sequences to its virtual interrupt controller after completing interrupt processing. For level triggered interrupts, these EOI writes to the virtual interrupt controller may be trapped by the VMM which may in turn unmask the previously masked interrupt source.

32.3.3.4 Generation of Virtual Interrupt Events by VMM

The following provides some of the general steps that need to be taken by VMM designs when generating virtual interrupts:

1. Check virtual processor interruptibility state. The virtual processor interruptibility state is reflected in the guest RFLAGS.IF flag and the processor interruptibility-state saved in the guest state area of the controlling-VMCS. If RFLAGS.IF is set and the interruptibility state indicates readiness to take external interrupts (STI-masking and MOV-SS/POP-SS-masking bits are clear), the guest virtual processor is ready to take external interrupts. If the VMM design supports non-active guest sleep states, the VMM needs to make sure the current guest sleep state allows injection of external interrupt events.
2. If the guest virtual processor state is currently not interruptible, a VMM may utilize the “interrupt-window exiting” VM-execution to notify the VM (through a VM exit) when the virtual processor state changes to interruptible state.
3. Check the virtual interrupt controller state. If the guest VM exposes a virtual local APIC, the current value of its processor priority register specifies if guest software allows dispensing an external virtual interrupt with a specific priority to the virtual processor. If the virtual interrupt is routed through the local vector table (LVT) entry of the local APIC, the mask bits in the corresponding LVT entry specifies if the interrupt is currently masked. Similarly, the virtual interrupt controller’s current mask (IO-APIC or PIC) and priority settings reflect guest state to accept specific external interrupts. The VMM needs to check both the virtual processor and interrupt controller states to verify its guest interruptibility state. If the guest is currently interruptible, the VMM can inject the virtual interrupt. If the current guest state does not allow injecting a virtual interrupt, the interrupt needs to be queued by the VMM until it can be delivered.
4. Prioritize the use of VM-entry event injection. A VMM may use VM-entry event injection to deliver various virtual events (such as external interrupts, exceptions, traps, and so forth). VMM designs may prioritize use of virtual-interrupt injection between these event types. Since each VM entry allows injection of one event, depending on the VMM event priority policies, the VMM may need to queue the external virtual interrupt if a higher priority event is to be

delivered on the next VM entry. Since the VMM has masked this particular interrupt source (if it was level triggered) and done EOI to the platform interrupt controller, other platform interrupts can be serviced while this virtual interrupt event is queued for later delivery to the VM.

5. Update the virtual interrupt controller state. When the above checks have passed, before generating the virtual interrupt to the guest, the VMM updates the virtual interrupt controller state (Local-APIC, IO-APIC and/or PIC) to reflect assertion of the virtual interrupt. This involves updating the various interrupt capture registers, and priority registers as done by the respective hardware interrupt controllers. Updating the virtual interrupt controller state is required for proper interrupt event processing by guest software.
6. Inject the virtual interrupt on VM entry. To inject an external virtual interrupt to a guest VM, the VMM sets up the VM-entry interruption-information field in the guest controlling-VMCS before entry to guest using VMRESUME. Upon VM entry, the processor will use this vector to access the gate in guest's IDT and the value of RFLAGS and EIP in guest-state area of controlling-VMCS is pushed on the guest stack. If the guest RFLAGS.IF is clear, the STI-masking bit is set, or the MOV- SS/POP-SS-masking bit is set, the VM entry will fail and the processor will load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 26.7).

32.4 ERROR HANDLING BY VMM

Error conditions may occur during VM entries and VM exits and a few other situations. This section describes how VMM should handle these error conditions, including triple faults and machine-check exceptions.

32.4.1 VM-Exit Failures

All VM exits load processor state from the host-state area of the VMCS that was the controlling VMCS before the VM exit. This state is checked for consistency while being loaded. Because the host-state is checked on VM entry, these checks will generally succeed. Failure is possible only if host software is incorrect or if VMCS data in the VMCS region in memory has been written by guest software (or by I/O DMA) since the last VM entry. VM exits may fail for the following reasons:

- There was a failure on storing guest MSRs.
- There was failure in loading a PDPTR.
- The controlling VMCS has been corrupted (through writes to the corresponding VMCS region) in such a way that the implementation cannot complete the VM exit.
- There was a failure on loading host MSRs.
- A machine-check event occurred.

If one of these problems occurs on a VM exit, a VMX abort results.

32.4.2 Machine-Check Considerations

The following sequence determine how machine-check events are handled during VMXON, VMXOFF, VM entries, and VM exits:

- VMXOFF and VMXON:

  If a machine-check event occurs during VMXOFF or VMXON and CR4.MCE = 1, a machine-check exception (#MC) is generated. If CR4.MCE = 0, the processor goes to shutdown state.

- VM entry:

  If a machine-check event occurs during VM entry, one of the following three treatments must occur:

  a. Normal delivery before VM entry. If CR4.MCE = 1 before VM entry, delivery of a machine-check exception (#MC) through the host IDT occurs. If CR4.MCE = 0, the processor goes to shutdown state.

  b. Normal delivery after VM entry. If CR4.MCE = 1 after VM entry, delivery of a machine-check exception (#MC) through the guest IDT occurs (alternatively, this exception may cause a VM exit). If CR4.MCE = 0, the processor goes to shutdown state.

  c. Load state from the host-state area of the working VMCS as if a VM exit had occurred (see Section 26.7). The basic exit reason will be “VM-entry failure due to machine-check event.”

  If the machine-check event occurs after any guest state has been loaded, option a above will not be used; it may be used if the machine-check event occurs while checking host state and VMX controls (or while reporting a failure due to such checks). An implementation may use option b only if all guest state has been loaded properly.

- VM exit:

  If a machine-check event occurs during VM exit, one of the following three treatments must occur:

  a. Normal delivery before VM exit. If CR4.MCE = 1 before the VM exit, delivery of a machine-check exception (#MC) through the guest IDT (alternatively, this may cause a VM exit). If CR4.MCE = 0, the processor goes to shutdown state.

  b. Normal delivery after VM exit. If CR4.MCE = 1 after the VM exit, delivery of a machine-check exception (#MC) through the host IDT. If CR4.MCE = 0, the processor goes to shutdown state.

  c. Fail the VM exit. If the VM exit is to VMX root operation, a VMX abort will result; it will block events as done normally in VMX abort. The VMX abort indicator will show that a machine-check event induced the abort operation.

If a machine-check event is induced by an action in VMX non-root operation before any determination is made that the inducing action may cause a VM exit, that machine-check event should be considered as happening during guest execution in VMX non-root operation. This is the case even if the part of the action that caused the machine-check event was VMX-specific (for example, the processor's consulting an I/O bitmap). If a machine-check exception occurs and if bit 12H of the exception bitmap is cleared to 0, the exception is delivered to the guest through gate 12H of its IDT; if the bit is set to 1, the machine-check exception causes a VM exit.

NOTE

The state saved in the guest-state area on VM exits due to machine-check exceptions should be considered suspect. A VMM should consult the RIPV and EIPV bits in the IA32_MCG_STATUS MSR before resuming a guest that caused a VM exit due to a machine-check exception.

32.4.3 MCA Error Handling Guidelines for VMM

Section 32.4.2 covers general requirements for VMMs to handle machine-check exceptions, when normal operation of the guest machine and/or the VMM is no longer possible. enhancements of machine-check architecture in newer processors may support software recovery of uncorrected MC errors (UCR) signaled through either machine-check exceptions or corrected machine-check interrupt (CMCI). Section 15.5 and Section 15.6 describes details of these more recent enhancements of machine-check architecture.

In general, Virtual Machine Monitor (VMM) error handling should follow the recommendations for OS error handling described in Section 15.3, Section 15.6, Section 15.9, and Section 15.10. This section describes additional guidelines for hosted and native hypervisor-based VMM implementations to support corrected MC errors and recoverable uncorrected MC errors.

Because a hosted VMM provides virtualization services in the context of an existing standard host OS, the host OS controls platform hardware through the host OS services such as the standard OS device drivers. In hosted VMMs. MCA errors will be handled by the host OS error handling software.

In native VMMs, the hypervisor runs on the hardware directly, and may provide only a limited set of platform services for guest VMs. Most platform services may instead be provided by a "control OS". In hypervisor-based VMMs, MCA errors will either be delivered directly to the VMM MCA handler (when the error is signaled while in the VMM context) or cause by a VM exit from a guest VM or be delivered to the MCA intercept handler. There are two general approaches the hypervisor can use to handle the MCA error: either within the hypervisor itself or by forwarding the error to the control OS.

32.4.3.1 VMM Error Handling Strategies

Broadly speaking, there are two strategies that VMMs may take for error handling:

- Basic error handling: in this approach the guest VM is treated as any other thread of execution. If the error recovery action does not support restarting the thread after handling the error, the guest VM should be terminated.
- MCA virtualization: in this approach, the VMM virtualizes the MCA events and hardware. This enables the VMM to intercept MCA events and inject an MCA into the guest VM. The guest VM then has the opportunity to attempt error recovery actions, rather than being terminated by the VMM.

Details of these approaches and implementation considerations for hosted and native VMMs are discussed below.

32.4.3.2 Basic VMM MCA error recovery handling

The simplest approach is for the VMM to treat the guest VM as any other thread of execution:

- MCE's that occur outside the stream of execution of a virtual machine guest will cause an MCE abort and may be handled by the MCA error handler following the recovery actions and guidelines described in Section 15.9, and Section 15.10. This includes logging the error and taking appropriate recovery actions when necessary. The VMM must not resume the interrupted thread of execution or another VM until it has taken the appropriate recovery action or, in the case of fatal MCAs, reset the system.
- MCE's that occur while executing in the context of a virtual machine will be intercepted by the VMM. The MCA intercept handler may follow the error handling guidelines listed in Section 15.9 and Section 15.10 for SRAO and SRAR errors. For SRAR errors, terminating the thread of execution will involve terminating the affected guest VM. For fatal errors the MCA handler should log the error and reset the system -- the VMM should not resume execution of the interrupted VM.

32.4.3.3 Implementation Considerations for the Basic Model

For hosted VMMs, the host OS MCA error handling code will perform error analysis and initiate the appropriate recovery actions. For the basic model this flow does not change when terminating a guest VM although the specific actions needed to terminate a guest VM may be different than terminating an application or user process.

For native, hypervisor-based VMMs, MCA errors will either be delivered directly to the VMM MCA handler (when the error is signaled while in the VMM context) or cause a VM exit from a guest VM or be delivered to the MCA intercept handler. There are two general approaches the hypervisor can use to handle the MCA error: either by forwarding the error to the control OS or within the hypervisor itself. These approaches are described in the following paragraphs.

The hypervisor may forward the error to the control OS for handling errors. This approach simplifies the hypervisor error handling since it relies on the control OS to implement the basic error handling model. The control OS error handling code will be similar to the error handling code in the hosted VMM. Errors can be forwarded to the control OS via an OS callback or by injecting an MCE event into the control OS. Injecting an MCE will cause the control OS MCA error handler to be invoked. The control OS is responsible for terminating the affected guest VM, if necessary, which may require cooperation from the hypervisor.

Alternatively, the error may be handled completely in the hypervisor. The hypervisor error handler is enhanced to implement the basic error handling model and the hypervisor error handler has the capability to fully analyze the error information and take recovery actions based on the guidelines. In this case error handling steps in the hypervisor are similar to those for the hosted VMM described above (where the hypervisor replaces the host OS actions). The hypervisor is responsible for terminating the affected guest VM, if necessary.

In all cases, if a fatal error is detected the VMM error handler should log the error and reset the system. The VMM error handler must ensure that guest VMs are not resumed after a fatal error is detected to ensure error containment is maintained.

32.4.3.4 MCA Virtualization

A more sophisticated approach for handling errors is to virtualize the MCA. This involves virtualizing the MCA hardware and intercepting the MCA event in the VMM when a guest VM is interrupted by an MCA. After analyzing the error, the VMM error handler may then decide to inject an MCE abort into the guest VM for attempted guest VM error recovery. This would enable the guest OS the opportunity to take recovery actions specific to that guest.

For MCA virtualization, the VMM must provide the guest physical address for memory errors instead of the system physical address when reporting the errors to the guest VM. To compute the guest physical address, the VMM needs to maintain a reverse mapping of system physical page addresses to guest physical page addresses.

When the MCE is injected into the guest VM, the guest OS MCA handler would be invoked. The guest OS implements the MCA handling guidelines and it could potentially terminate the interrupted thread of execution within the guest instead of terminating the VM. The guest OS may also disable use of the affected page by the guest. When disabling the page the VMM error handler may handle the case where a page is shared by the VMM and a guest or by two guests. In these cases the page use must be disabled in both contexts to ensure no subsequent consumption errors are generated.

32.4.3.5 Implementation Considerations for the MCA Virtualization Model

MCA virtualization may be done in either hosted VMMs or hypervisor-based VMMs. The error handling flow is similar to the flow described in the basic handling case. The major difference is that the recovery action includes injecting the MCE abort into the guest VM to enable recovery by the guest OS when the MCA interrupts the execution of a guest VM.

32.5 HANDLING ACTIVITY STATES BY VMM

A VMM might place a logic processor in the wait-for-SIPI activity state if supporting certain guest operating system using the multi-processor (MP) start-up algorithm. A guest with direct access to the physical local APIC and using the MP start-up algorithm sends an INIT-SIPI-SIPI IPI sequence to start the application processor. In order to trap the SIPIs, the VMM must start the logic processor which is the target of the SIPIs in wait-for-SIPI mode.

CHAPTER 33
VMX INSTRUCTION REFERENCE

NOTE

This chapter was previously located in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B* as chapter 5.

For the current document revision, this chapter is located here as Chapter 33, "VMX Instruction Reference," and duplicated in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2C* as chapter 5.

For all future document revisions, this chapter will be located here as Chapter 33, "VMX Instruction Reference," and will not be duplicated elsewhere.

33.1 OVERVIEW

This chapter describes the virtual-machine extensions (VMX) for the Intel 64 and IA-32 architectures. VMX is intended to support virtualization of processor hardware and a system software layer acting as a host to multiple guest software environments. The virtual-machine extensions (VMX) includes five instructions that manage the virtual-machine control structure (VMCS), four instructions that manage VMX operation, two TLB-management instructions, and two instructions for use by guest software. Additional details of VMX are described in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

The behavior of the VMCS-maintenance instructions is summarized below:

- **VMPTRLD —** This instruction takes a single 64-bit source operand that is in memory. It makes the referenced VMCS active and current, loading the current-VMCS pointer with this operand and establishes the current VMCS based on the contents of VMCS-data area in the referenced VMCS region. Because this makes the referenced VMCS active, a logical processor may start maintaining on the processor some of the VMCS data for the VMCS.
- **VMPTRST —** This instruction takes a single 64-bit destination operand that is in memory. The current-VMCS pointer is stored into the destination operand.
- **VMCLEAR —** This instruction takes a single 64-bit operand that is in memory. The instruction sets the launch state of the VMCS referenced by the operand to "clear", renders that VMCS inactive, and ensures that data for the VMCS have been written to the VMCS-data area in the referenced VMCS region. If the operand is the same as the current-VMCS pointer, that pointer is made invalid.

- **VMREAD —** This instruction reads a component from the VMCS (the encoding of that field is given in a register operand) and stores it into a destination operand that may be a register or in memory.
- **VMWRITE —** This instruction writes a component to the VMCS (the encoding of that field is given in a register operand) from a source operand that may be a register or in memory.

The behavior of the VMX management instructions is summarized below:

- **VMLAUNCH —** This instruction launches a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
- **VMRESUME —** This instruction resumes a virtual machine managed by the VMCS. A VM entry occurs, transferring control to the VM.
- **VMXOFF —** This instruction causes the processor to leave VMX operation.
- **VMXON —** This instruction takes a single 64-bit source operand that is in memory. It causes a logical processor to enter VMX root operation and to use the memory referenced by the operand to support VMX operation.

The behavior of the VMX-specific TLB-management instructions is summarized below:

- **INVEPT —** This instruction invalidates entries in the TLBs and paging-structure caches that were derived from extended page tables (EPT).
- **INVVPID —** This instruction invalidates entries in the TLBs and paging-structure caches based on a Virtual-Processor Identifier (VPID).

None of the instructions above can be executed in compatibility mode; they generate invalid-opcode exceptions if executed in compatibility mode.

The behavior of the guest-available instructions is summarized below:

- **VMCALL —** This instruction allows software in VMX non-root operation to call the VMM for service. A VM exit occurs, transferring control to the VMM.
- **VMFUNC —** This instruction allows software in VMX non-root operation to invoke a VM function, which is processor functionality enabled and configured by software in VMX root operation. No VM exit occurs.

33.2 CONVENTIONS

The operation sections for the VMX instructions in Section 33.3 use the pseudo-function VMexit, which indicates that the logical processor performs a VM exit.

The operation sections also use the pseudo-functions VMsucceed, VMfail, VMfailInvalid, and VMfailValid. These pseudo-functions signal instruction success or failure by setting or clearing bits in RFLAGS and, in some cases, by writing the VM-instruction error field. The following pseudocode fragments detail these functions:

```
VMsucceed:
    CF ← 0;
    PF ← 0;
    AF ← 0;
    ZF ← 0;
    SF ← 0;
    OF ← 0;

VMfail(ErrorNumber):
    IF VMCS pointer is valid
        THEN VMfailValid(ErrorNumber);
        ELSE VMfailInvalid;
    FI;

VMfailInvalid:
    CF ← 1;
    PF ← 0;
    AF ← 0;
    ZF ← 0;
    SF ← 0;
    OF ← 0;

VMfailValid(ErrorNumber):// executed only if there is a current VMCS
    CF ← 0;
    PF ← 0;
    AF ← 0;
    ZF ← 1;
    SF ← 0;
    OF ← 0;
    Set the VM-instruction error field to ErrorNumber;
```

The different VM-instruction error numbers are enumerated in Section 33.4, “VM Instruction Error Numbers”.

33.3 VMX INSTRUCTIONS

This section provides detailed descriptions of the VMX instructions.

INVEPT— Invalidate Translations Derived from EPT

Opcode	Instruction	Description
66 0F 38 80	INVEPT r64, m128	Invalidates EPT-derived entries in the TLBs and paging-structure caches (in 64-bit mode)
66 0F 38 80	INVEPT r32, m128	Invalidates EPT-derived entries in the TLBs and paging-structure caches (outside 64-bit mode)

Description

Invalidates mappings in the translation lookaside buffers (TLBs) and paging-structure caches that were derived from extended page tables (EPT). (See Chapter 28, "VMX Support for Address Translation" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.) Invalidation is based on the **INVEPT type** specified in the register operand and the **INVEPT descriptor** specified in the memory operand.

Outside IA-32e mode, the register operand is always 32 bits, regardless of the value of CS.D; in 64-bit mode, the register operand has 64 bits (the instruction cannot be executed in compatibility mode).

The INVEPT types supported by a logical processors are reported in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A, "VMX Capability Reporting Facility" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*). There are two INVEPT types currently defined:

- Single-context invalidation. If the INVEPT type is 1, the logical processor invalidates all mappings associated with bits 51:12 of the EPT pointer (EPTP) specified in the INVEPT descriptor. It may invalidate other mappings as well.
- Global invalidation: If the INVEPT type is 2, the logical processor invalidates mappings associated with all EPTPs.

If an unsupported INVEPT type is specified, the instruction fails.

INVEPT invalidates all the specified mappings for the indicated EPTP(s) regardless of the VPID and PCID values with which those mappings may be associated.

The INVEPT descriptor comprises 128 bits and contains a 64-bit EPTP value in bits 63:0 (see Figure 33-1).

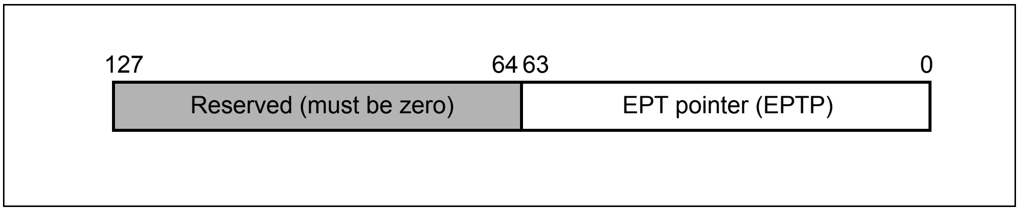

Figure 33-1. INVEPT Descriptor

Operation

```
IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
         INVEPT_TYPE ← value of register operand;
         IF IA32_VMX_EPT_VPID_CAP MSR indicates that processor does not support INVEPT_TYPE
              THEN VMfail(Invalid operand to INVEPT/INVVPID);
              ELSE     // INVEPT_TYPE must be 1 or 2
                   INVEPT_DESC ← value of memory operand;
                   EPTP ← INVEPT_DESC[63:0];
                   CASE INVEPT_TYPE OF
                        1:                  // single-context invalidation
                             IF VM entry with the “enable EPT“ VM execution control set to 1
                             would fail due to the EPTP value
                                  THEN VMfail(Invalid operand to INVEPT/INVVPID);
                                  ELSE
                                       Invalidate mappings associated with EPTP[51:12];
                                       VMsucceed;
                             FI;
                             BREAK;
                        2:                  // global invalidation
                             Invalidate mappings associated with all EPTPs;
                             VMsucceed;
                             BREAK;
                   ESAC;
         FI;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.

#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If not in VMX operation.
	If the logical processor does not support EPT (IA32_VMX_PROCBASED_CTLS2[33]=0).
	If the logical processor supports EPT (IA32_VMX_PROCBASED_CTLS2[33]=1) but does not support the INVEPT instruction (IA32_VMX_EPT_VPID_CAP[20]=0).

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the INVEPT instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The INVEPT instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The INVEPT instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.
	If the logical processor does not support EPT (IA32_VMX_PROCBASED_CTLS2[33]=0).
	If the logical processor supports EPT (IA32_VMX_PROCBASED_CTLS2[33]=1) but does not support the INVEPT instruction (IA32_VMX_EPT_VPID_CAP[20]=0).

INVVPID— Invalidate Translations Based on VPID

Opcode	Instruction	Description
66 0F 38 81	INVVPID r64, m128	Invalidates entries in the TLBs and paging-structure caches based on VPID (in 64-bit mode)
66 0F 38 81	INVVPID r32, m128	Invalidates entries in the TLBs and paging-structure caches based on VPID (outside 64-bit mode)

Description

Invalidates mappings in the translation lookaside buffers (TLBs) and paging-structure caches based on **virtual-processor identifier** (VPID). (See Chapter 28, "VMX Support for Address Translation" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.) Invalidation is based on the **INVVPID type** specified in the register operand and the **INVVPID descriptor** specified in the memory operand.

Outside IA-32e mode, the register operand is always 32 bits, regardless of the value of CS.D; in 64-bit mode, the register operand has 64 bits (the instruction cannot be executed in compatibility mode).

The INVVPID types supported by a logical processors are reported in the IA32_VMX_EPT_VPID_CAP MSR (see Appendix A, "VMX Capability Reporting Facility" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*). There are four INVVPID types currently defined:

- Individual-address invalidation: If the INVVPID type is 0, the logical processor invalidates mappings for the linear address and VPID specified in the INVVPID descriptor. In some cases, it may invalidate mappings for other linear addresses (or other VPIDs) as well.
- Single-context invalidation: If the INVVPID type is 1, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor. In some cases, it may invalidate mappings for other VPIDs as well.
- All-contexts invalidation: If the INVVPID type is 2, the logical processor invalidates all mappings tagged with all VPIDs except VPID 0000H. In some cases, it may invalidate translations with VPID 0000H as well.
- Single-context invalidation, retaining global translations: If the INVVPID type is 3, the logical processor invalidates all mappings tagged with the VPID specified in the INVVPID descriptor except global translations. In some cases, it may invalidate global translations (and mappings with other VPIDs) as well. See the "Caching Translation Information" section in Chapter 4 of the *IA-32 Intel Architecture Software Developer's Manual, Volumes 3A* for information about global translations.

If an unsupported INVVPID type is specified, the instruction fails.

INVVPID invalidates all the specified mappings for the indicated VPID(s) regardless of the EPTP and PCID values with which those mappings may be associated.

The INVVPID descriptor comprises 128 bits and consists of a VPID and a linear address as shown in Figure 33-2.

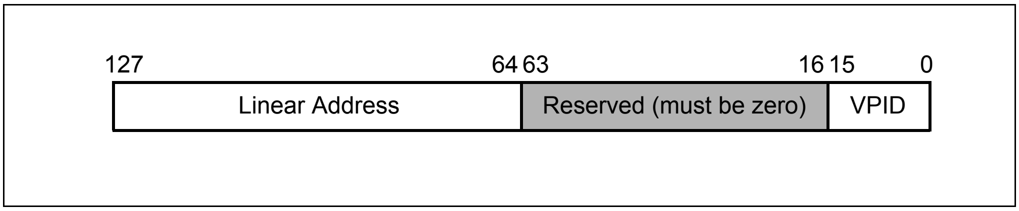

Figure 33-2. INVVPID Descriptor

Operation

```
IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        INVVPID_TYPE ← value of register operand;
        IF IA32_VMX_EPT_VPID_CAP MSR indicates that processor does not support
        INVVPID_TYPE
            THEN VMfail(Invalid operand to INVEPT/INVVPID);
            ELSE            // INVVPID_TYPE must be in the range 0–3
                INVVPID_DESC ← value of memory operand;
                IF INVVPID_DESC[63:16] ≠ 0
                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                    ELSE
                        CASE INVVPID_TYPE OF
                            0:                  // individual-address invalidation
                                VPID ← INVVPID_DESC[15:0];
                                IF VPID = 0
                                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                                    ELSE
                                        GL_ADDR ← INVVPID_DESC[127:64];
                                        IF (GL_ADDR is not in a canonical form)
                                            THEN
                                                VMfail(Invalid operand to INVEPT/INVVPID);
                                            ELSE
                                        Invalidate mappings for GL_ADDR tagged with VPID;
                                        VMsucceed;
                                    FI;
                                FI;
                                BREAK;
                            1:              // single-context invalidation
                                VPID ← INVVPID_DESC[15:0];
                                IF VPID = 0
                                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                                    ELSE
                                        Invalidate all mappings tagged with VPID;
                                        VMsucceed;
                                FI;
                                BREAK;
                            2:              // all-context invalidation
                                Invalidate all mappings tagged with all non-zero VPIDs;
                                VMsucceed;
                                BREAK;
                            3:              // single-context invalidation retaining globals
                                VPID ← INVVPID_DESC[15:0];
                                IF VPID = 0
                                    THEN VMfail(Invalid operand to INVEPT/INVVPID);
                                    ELSE
                                        Invalidate all mappings tagged with VPID except global translations;
                                        VMsucceed;
                                FI;
                                BREAK;
                        ESAC;
                FI;
        FI;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.

	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If not in VMX operation.
	If the logical processor does not support VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=0).
	If the logical processor supports VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=1) but does not support the INVVPID instruction (IA32_VMX_EPT_VPID_CAP[32]=0).

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the INVVPID instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The INVVPID instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The INVVPID instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.
	If the logical processor does not support VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=0).
	If the logical processor supports VPIDs (IA32_VMX_PROCBASED_CTLS2[37]=1) but does not support the INVVPID instruction (IA32_VMX_EPT_VPID_CAP[32]=0).

VMCALL—Call to VM Monitor

Opcode	Instruction	Description
0F 01 C1	VMCALL	Call to VM monitor by causing VM exit.

Description

This instruction allows guest software can make a call for service into an underlying VM monitor. The details of the programming interface for such calls are VMM-specific; this instruction does nothing more than cause a VM exit, registering the appropriate exit reason.

Use of this instruction in VMX root operation invokes an SMM monitor (see Section 29.15.2 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*). This invocation will activate the dual-monitor treatment of system-management interrupts (SMIs) and system-management mode (SMM) if it is not already active (see Section 29.15.6 in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*).

Operation

```
IF not in VMX operation
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF in SMM or the logical processor does not support the dual-monitor treatment of SMIs and
SMM or the valid bit in the IA32_SMM_MONITOR_CTL MSR is clear
    THEN VMfail (VMCALL executed in VMX root operation);
ELSIF dual-monitor treatment of SMIs and SMM is active
    THEN perform an SMM VM exit (see Section 29.15.2
     of the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
ELSIF launch state of current VMCS is not clear
    THEN VMfailValid(VMCALL with non-clear VMCS);
ELSIF VM-exit control fields are not valid (see Section 29.15.6.1 of the Intel® 64 and IA-32 Archi-
tectures Software Developer's Manual, Volume 3C)
    THEN VMfailValid (VMCALL with invalid VM-exit control fields);
ELSE
    enter SMM;
    read revision identifier in MSEG;
    IF revision identifier does not match that supported by processor
        THEN
            leave SMM;
            VMfailValid(VMCALL with incorrect MSEG revision identifier);
        ELSE
            read SMM-monitor features field in MSEG (see Section 29.15.6.2,
            in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C);
            IF features field is invalid
                THEN
                    leave SMM;
                    VMfailValid(VMCALL with invalid SMM-monitor features);
                ELSE activate dual-monitor treatment of SMIs and SMM (see Section 29.15.6
                in the Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume
                3C);
            FI;
        FI;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0 and the logical processor is in VMX root operation.
#UD	If executed outside VMX operation.

Real-Address Mode Exceptions

#UD	If executed outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	If executed outside VMX non-root operation.

Compatibility Mode Exceptions

#UD	If executed outside VMX non-root operation.

64-Bit Mode Exceptions

#UD	If executed outside VMX non-root operation.

VMCLEAR—Clear Virtual-Machine Control Structure

Opcode	Instruction	Description
66 0F C7 /6	VMCLEAR m64	Copy VMCS data to VMCS region in memory.

Description

This instruction applies to the VMCS whose VMCS region resides at the physical address contained in the instruction operand. The instruction ensures that VMCS data for that VMCS (some of these data may be currently maintained on the processor) are copied to the VMCS region in memory. It also initializes parts of the VMCS region (for example, it sets the launch state of that VMCS to clear). See Chapter 24, “Virtual-Machine Control Structures,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*.

The operand of this instruction is always 64 bits and is always in memory. If the operand is the current-VMCS pointer, then that pointer is made invalid (set to FFFFFFFF_FFFFFFFFH).

Note that the VMCLEAR instruction might not explicitly write any VMCS data to memory; the data may be already resident in memory before the VMCLEAR is executed.

Operation

```
IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VM exit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        addr ← contents of 64-bit in-memory operand;
        IF addr is not 4KB-aligned OR
        addr sets any bits beyond the physical-address width[1]
            THEN VMfail(VMCLEAR with invalid physical address);
        ELSIF addr = VMXON pointer
            THEN VMfail(VMCLEAR with VMXON pointer);
            ELSE
                ensure that data for VMCS referenced by the operand is in memory;
                initialize implementation-specific data in VMCS region;
```

1. If IA32_VMX_BASIC[48] is read as 1, VMfail occurs if addr sets any bits in the range 63:32; see Appendix A.1.

```
                launch state of VMCS referenced by the operand ← “clear”
                IF operand addr = current-VMCS pointer
                    THEN current-VMCS pointer ← FFFFFFFF_FFFFFFFFH;
                FI;
                VMsucceed;
        FI;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the memory operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If operand is a register.
	If not in VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMCLEAR instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMCLEAR instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.

#PF(fault-code)	If a page fault occurs in accessing the memory operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If not in VMX operation.

VMFUNC—Invoke VM Function

Opcode	Instruction	Description
0F 01 D4	VMFUNC	Invoke VM function specified in EAX.

Description

This instruction allows software in VMX non-root operation to invoke a VM function, which is processor functionality enabled and configured by software in VMX root operation. The value of EAX selects the specific VM function being invoked.

The behavior of each VM function (including any additional fault checking) is specified in Section 25.7.4, "VM Functions," in *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

Operation

```
Perform functionality of the VM function specified in EAX;
```

Flags Affected

Depends on the VM function specified in EAX. See Section 25.7.4, "VM Functions," in *Intel 64 and IA-32 Architecture Software Developer's Manual, Volume 3C*.

Protected Mode Exceptions (not including those defined by specific VM functions)

#UD	If executed outside VMX non-root operation.
	If "enable VM functions" VM-execution control is 0
	If EAX ≥ 64
	If the bit at position EAX is 0 in the VM-function controls

Real-Address Mode Exceptions

Same exceptions as in protected mode.

Virtual-8086 Exceptions

Same exceptions as in protected mode.

Compatibility Mode Exceptions

Same exceptions as in protected mode.

64-Bit Mode Exceptions

Same exceptions as in protected mode.

VMLAUNCH/VMRESUME—Launch/Resume Virtual Machine

Opcode	Instruction	Description
0F 01 C2	VMLAUNCH	Launch virtual machine managed by current VMCS.
0F 01 C3	VMRESUME	Resume virtual machine managed by current VMCS.

Description

Effects a VM entry managed by the current VMCS.

- VMLAUNCH fails if the launch state of current VMCS is not "clear". If the instruction is successful, it sets the launch state to "launched."
- VMRESUME fails if the launch state of the current VMCS is not "launched."

If VM entry is attempted, the logical processor performs a series of consistency checks as detailed in Chapter 26, "VM Entries," in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*. Failure to pass checks on the VMX controls or on the host-state area passes control to the instruction following the VMLAUNCH or VMRESUME instruction. If these pass but checks on the guest-state area fail, the logical processor loads state from the host-state area of the VMCS, passing control to the instruction referenced by the RIP field in the host-state area.

VM entry is not allowed when events are blocked by MOV SS or POP SS. Neither VMLAUNCH nor VMRESUME should be used immediately after either MOV to SS or POP to SS.

Operation

```
IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
ELSIF events are being blocked by MOV SS
    THEN VMfailValid(VM entry with events blocked by MOV SS);
ELSIF (VMLAUNCH and launch state of current VMCS is not "clear")
    THEN VMfailValid(VMLAUNCH with non-clear VMCS);
ELSIF (VMRESUME and launch state of current VMCS is not "launched")
    THEN VMfailValid(VMRESUME with non-launched VMCS);
    ELSE
         Check settings of VMX controls and host-state area;
         IF invalid settings
```

```
            THEN VMfailValid(VM entry with invalid VMX-control field(s)) or
                    VMfailValid(VM entry with invalid host-state field(s)) or
                    VMfailValid(VM entry with invalid executive-VMCS pointer)) or
                    VMfailValid(VM entry with non-launched executive VMCS) or
                    VMfailValid(VM entry with executive-VMCS pointer not VMXON pointer) or
                    VMfailValid(VM entry with invalid VM-execution control fields in executive
                    VMCS)
                    as appropriate;
            ELSE
                Attempt to load guest state and PDPTRs as appropriate;
                clear address-range monitoring;
                IF failure in checking guest state or PDPTRs
                    THEN VM entry fails (see Section 26.7, in the
                    Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C);
                    ELSE
                        Attempt to load MSRs from VM-entry MSR-load area;
                        IF failure
                            THEN VM entry fails (see Section 26.7, in the Intel® 64 and IA-32
Architectures Software Developer’s Manual, Volume 3C);
                            ELSE
                                IF VMLAUNCH
                                    THEN launch state of VMCS ← “launched”;
                                FI;
                                IF in SMM and “entry to SMM” VM-entry control is 0
                                    THEN
                                        IF “deactivate dual-monitor treatment” VM-entry
                                        control is 0
                                            THEN SMM-transfer VMCS pointer ←
                                            current-VMCS pointer;
                                        FI;
                                        IF executive-VMCS pointer is VMX pointer
                                            THEN current-VMCS pointer ←
                                            VMCS-link pointer;
                                            ELSE current-VMCS pointer ←
                                            executive-VMCS pointer;
                                        FI;
                                        leave SMM;
                                FI;
                                VM entry succeeds;
                        FI;
                FI;
        FI;
FI;
```

Further details of the operation of the VM-entry appear in Chapter 26 of *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C*.

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
#UD	If executed outside VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMLAUNCH and VMRESUME instructions are not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMLAUNCH and VMRESUME instructions are not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
#UD	If executed outside VMX operation.

VMPTRLD—Load Pointer to Virtual-Machine Control Structure

Opcode	Instruction	Description
0F C7 /6	VMPTRLD m64	Loads the current VMCS pointer from memory.

Description

Marks the current-VMCS pointer valid and loads it with the physical address in the instruction operand. The instruction fails if its operand is not properly aligned, sets unsupported physical-address bits, or is equal to the VMXON pointer. In addition, the instruction fails if the 32 bits in memory referenced by the operand do not match the VMCS revision identifier supported by this processor.[2]

The operand of this instruction is always 64 bits and is always in memory.

Operation

```
IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
         addr ← contents of 64-bit in-memory source operand;
         IF addr is not 4KB-aligned OR
         addr sets any bits beyond the physical-address width[3]
              THEN VMfail(VMPTRLD with invalid physical address);
         ELSIF addr = VMXON pointer
              THEN VMfail(VMPTRLD with VMXON pointer);
              ELSE
                   rev ← 32 bits located at physical address addr;
                   IF rev ≠ VMCS revision identifier supported by processor
                        THEN VMfail(VMPTRLD with incorrect VMCS revision identifier);
                        ELSE
                             current-VMCS pointer ← addr;
                             VMsucceed;
```

2. Software should consult the VMX capability MSR VMX_BASIC to discover the VMCS revision identifier supported by this processor (see Appendix A, “VMX Capability Reporting Facility,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*).

3. If IA32_VMX_BASIC[48] is read as 1, VMfail occurs if addr sets any bits in the range 63:32; see Appendix A.1.

```
                FI;
        FI;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If operand is a register.
	If not in VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMPTRLD instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMPTRLD instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.

#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If not in VMX operation.

VMPTRST—Store Pointer to Virtual-Machine Control Structure

Opcode	Instruction	Description
0F C7 /7	VMPTRST m64	Stores the current VMCS pointer into memory.

Description

Stores the current-VMCS pointer into a specified memory address. The operand of this instruction is always 64 bits and is always in memory.

Operation

```
IF (register operand) or (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE
        64-bit in-memory destination operand ← current-VMCS pointer;
        VMsucceed;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory destination operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the destination operand is located in a read-only data segment or any code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory destination operand.
#SS(0)	If the memory destination operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If operand is a register.
	If not in VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMPTRST instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMPTRST instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMPTRST instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the destination operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory destination operand.
#SS(0)	If the destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If not in VMX operation.

VMREAD—Read Field from Virtual-Machine Control Structure

Opcode	Instruction	Description
0F 78	VMREAD r/m64, r64	Reads a specified VMCS field (in 64-bit mode).
0F 78	VMREAD r/m32, r32	Reads a specified VMCS field (outside 64-bit mode).

Description

Reads a specified field from the VMCS and stores it into a specified destination operand (register or memory).

The specific VMCS field is identified by the VMCS-field encoding contained in the register source operand. Outside IA-32e mode, the source operand has 32 bits, regardless of the value of CS.D. In 64-bit mode, the source operand has 64 bits; however, if bits 63:32 of the source operand are not zero, VMREAD will fail due to an attempt to access an unsupported VMCS component (see operation section).

The effective size of the destination operand, which may be a register or in memory, is always 32 bits outside IA-32e mode (the setting of CS.D is ignored with respect to operand size) and 64 bits in 64-bit mode. If the VMCS field specified by the source operand is shorter than this effective operand size, the high bits of the destination operand are cleared to 0. If the VMCS field is longer, then the high bits of the field are not read.

Note that any faults resulting from accessing a memory destination operand can occur only after determining, in the operation section below, that the VMCS pointer is valid and that the specified VMCS field is supported.

Operation

```
IF (not in VMX operation) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
ELSIF register source operand does not correspond to any VMCS field
    THEN VMfailValid(VMREAD/VMWRITE from/to unsupported VMCS component);
    ELSE
        DEST ← contents of VMCS field indexed by register source operand;
        VMsucceed;
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If a memory destination operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the destination operand is located in a read-only data segment or any code segment.
#PF(fault-code)	If a page fault occurs in accessing a memory destination operand.
#SS(0)	If a memory destination operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If not in VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMREAD instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMREAD instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMREAD instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory destination operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing a memory destination operand.
#SS(0)	If the memory destination operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

VMRESUME—Resume Virtual Machine

See VMLAUNCH/VMRESUME—Launch/Resume Virtual Machine.

VMWRITE—Write Field to Virtual-Machine Control Structure

Opcode	Instruction	Description
0F 79	VMWRITE r64, r/m64	Writes a specified VMCS field (in 64-bit mode)
0F 79	VMWRITE r32, r/m32	Writes a specified VMCS field (outside 64-bit mode)

Description

Writes to a specified field in the VMCS specified by a secondary source operand (register only) using the contents of a primary source operand (register or memory).

The VMCS field is identified by the VMCS-field encoding contained in the register secondary source operand. Outside IA-32e mode, the secondary source operand is always 32 bits, regardless of the value of CS.D. In 64-bit mode, the secondary source operand has 64 bits; however, if bits 63:32 of the secondary source operand are not zero, VMWRITE will fail due to an attempt to access an unsupported VMCS component (see operation section).

The effective size of the primary source operand, which may be a register or in memory, is always 32 bits outside IA-32e mode (the setting of CS.D is ignored with respect to operand size) and 64 bits in 64-bit mode. If the VMCS field specified by the secondary source operand is shorter than this effective operand size, the high bits of the primary source operand are ignored. If the VMCS field is longer, then the high bits of the field are cleared to 0.

Note that any faults resulting from accessing a memory source operand occur after determining, in the operation section below, that the VMCS pointer is valid but before determining if the destination VMCS field is supported.

Operation

```
IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF current-VMCS pointer is not valid
    THEN VMfailInvalid;
ELSIF register destination operand does not correspond to any VMCS field
    THEN VMfailValid(VMREAD/VMWRITE from/to unsupported VMCS component);
ELSIF VMCS field indexed by register destination operand is read-only)
    THEN VMfailValid(VMWRITE to read-only VMCS component);
    ELSE
        VMCS field indexed by register destination operand ← SRC;
        VMsucceed;
```

```
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If a memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing a memory source operand.
#SS(0)	If a memory source operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If not in VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMWRITE instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMWRITE instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMWRITE instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If the current privilege level is not 0.
	If the memory source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing a memory source operand.
#SS(0)	If the memory source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If not in VMX operation.

VMXOFF—Leave VMX Operation

Opcode	Instruction	Description
0F 01 C4	VMXOFF	Leaves VMX operation.

Description

Takes the logical processor out of VMX operation, unblocks INIT signals, conditionally re-enables A20M, and clears any address-range monitoring.[4]

Operation

```
IF (not in VMX operation) or (CR0.PE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
ELSIF dual-monitor treatment of SMIs and SMM is active
    THEN VMfail(VMXOFF under dual-monitor treatment of SMIs and SMM);
    ELSE
        leave VMX operation;
        unblock INIT;
        IF IA32_SMM_MONITOR_CTL[2] = 0[5]
            THEN unblock SMIs;
        IF outside SMX operation[6]
            THEN unblock and enable A20M;
        FI;
        clear address-range monitoring;
        VMsucceed;
FI;
```

4. See the information on MONITOR/MWAIT in Chapter 8, "Multiple-Processor Management," of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A*.
5. Setting IA32_SMM_MONITOR_CTL[bit 2] to 1 prevents VMXOFF from unblocking SMIs regardless of the value of the register's value bit (bit 0). Not all processors allow this bit to be set to 1. Software should consult the VMX capability MSR IA32_VMX_MISC (see Appendix A.6) to determine whether this is allowed.
6. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GETSEC[SENTER]. See Chapter 6, "Safer Mode Extensions Reference."

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If executed in VMX root operation with CPL > 0.
#UD	If executed outside VMX operation.

Real-Address Mode Exceptions

#UD	A logical processor cannot be in real-address mode while in VMX operation and the VMXOFF instruction is not recognized outside VMX operation.

Virtual-8086 Mode Exceptions

#UD	The VMXOFF instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMXOFF instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If executed in VMX root operation with CPL > 0.
#UD	If executed outside VMX operation.

VMXON—Enter VMX Operation

Opcode	Instruction	Description
F3 0F C7 /6	VMXON m64	Enter VMX root operation.

Description

Puts the logical processor in VMX operation with no current VMCS, blocks INIT signals, disables A20M, and clears any address-range monitoring established by the MONITOR instruction.[7]

The operand of this instruction is a 4KB-aligned physical address (the VMXON pointer) that references the VMXON region, which the logical processor may use to support VMX operation. This operand is always 64 bits and is always in memory.

Operation

```
IF (register operand) or (CR0.PE = 0) or (CR4.VMXE = 0) or (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1
and CS.L = 0)
    THEN #UD;
ELSIF not in VMX operation
    THEN
        IF (CPL > 0) or (in A20M mode) or
        (the values of CR0 and CR4 are not supported in VMX operation[8]) or
        (bit 0 (lock bit) of IA32_FEATURE_CONTROL MSR is clear) or
        (in SMX operation[9] and bit 1 of IA32_FEATURE_CONTROL MSR is clear) or
        (outside SMX operation and bit 2 of IA32_FEATURE_CONTROL MSR is clear)
            THEN #GP(0);
            ELSE
                addr ← contents of 64-bit in-memory source operand;
                IF addr is not 4KB-aligned or
                addr sets any bits beyond the physical-address width[10]
                    THEN VMfailInvalid;
```

7. See the information on MONITOR/MWAIT in Chapter 8, “Multiple-Processor Management,” of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3A*.
8. See Section 19.8 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B*.
9. A logical processor is in SMX operation if GETSEC[SEXIT] has not been executed since the last execution of GETSEC[SENTER]. A logical processor is outside SMX operation if GETSEC[SENTER] has not been executed or if GETSEC[SEXIT] was executed after the last execution of GET-SEC[SENTER]. See Chapter 6, “Safer Mode Extensions Reference.”
10. If IA32_VMX_BASIC[48] is read as 1, VMfailInvalid occurs if addr sets any bits in the range 63:32; see Appendix A.1.

```
                        ELSE
                            rev ← 32 bits located at physical address addr;
                            IF rev ≠ VMCS revision identifier supported by processor
                                THEN VMfailInvalid;
                                ELSE
                                    current-VMCS pointer ← FFFFFFFF_FFFFFFFFH;
                                    enter VMX operation;
                                    block INIT signals;
                                    block and disable A20M;
                                    clear address-range monitoring;
                                    VMsucceed;
                            FI;
                    FI;
            FI;
ELSIF in VMX non-root operation
    THEN VMexit;
ELSIF CPL > 0
    THEN #GP(0);
    ELSE VMfail("VMXON executed in VMX root operation");
FI;
```

Flags Affected

See the operation section and Section 33.2.

Protected Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL>0 or with invalid CR0 or CR4 fixed bits.
	If executed in A20M mode.
	If the memory source operand effective address is outside the CS, DS, ES, FS, or GS segment limit.
	If the DS, ES, FS, or GS register contains an unusable segment.
	If the source operand is located in an execute-only code segment.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the memory source operand effective address is outside the SS segment limit.
	If the SS register contains an unusable segment.
#UD	If operand is a register.
	If executed with CR4.VMXE = 0.

Real-Address Mode Exceptions

#UD	The VMXON instruction is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	The VMXON instruction is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

#UD	The VMXON instruction is not recognized in compatibility mode.

64-Bit Mode Exceptions

#GP(0)	If executed outside VMX operation with CPL > 0 or with invalid CR0 or CR4 fixed bits.
	If executed in A20M mode.
	If the source operand is in the CS, DS, ES, FS, or GS segments and the memory address is in a non-canonical form.
#PF(fault-code)	If a page fault occurs in accessing the memory source operand.
#SS(0)	If the source operand is in the SS segment and the memory address is in a non-canonical form.
#UD	If operand is a register.
	If executed with CR4.VMXE = 0.

33.4 VM INSTRUCTION ERROR NUMBERS

For certain error conditions, the VM-instruction error field is loaded with an error number to indicate the source of the error. Table 33-1 lists VM-instruction error numbers.

Table 33-1. VM-Instruction Error Numbers

Error Number	Description
1	VMCALL executed in VMX root operation
2	VMCLEAR with invalid physical address
3	VMCLEAR with VMXON pointer
4	VMLAUNCH with non-clear VMCS
5	VMRESUME with non-launched VMCS
6	VMRESUME after VMXOFF (VMXOFF and VMXON between VMLAUNCH and VMRESUME)[a]
7	VM entry with invalid control field(s)[b,c]
8	VM entry with invalid host-state field(s)[b]
9	VMPTRLD with invalid physical address
10	VMPTRLD with VMXON pointer
11	VMPTRLD with incorrect VMCS revision identifier
12	VMREAD/VMWRITE from/to unsupported VMCS component
13	VMWRITE to read-only VMCS component
15	VMXON executed in VMX root operation
16	VM entry with invalid executive-VMCS pointer[b]
17	VM entry with non-launched executive VMCS[b]
18	VM entry with executive-VMCS pointer not VMXON pointer (when attempting to deactivate the dual-monitor treatment of SMIs and SMM)[b]
19	VMCALL with non-clear VMCS (when attempting to activate the dual-monitor treatment of SMIs and SMM)
20	VMCALL with invalid VM-exit control fields
22	VMCALL with incorrect MSEG revision identifier (when attempting to activate the dual-monitor treatment of SMIs and SMM)
23	VMXOFF under dual-monitor treatment of SMIs and SMM
24	VMCALL with invalid SMM-monitor features (when attempting to activate the dual-monitor treatment of SMIs and SMM)

Table 33-1. VM-Instruction Error Numbers (Contd.)

Error Number	Description
25	VM entry with invalid VM-execution control fields in executive VMCS (when attempting to return from SMM)[b,c]
26	VM entry with events blocked by MOV SS.
28	Invalid operand to INVEPT/INVVPID.

NOTES:

a. Earlier versions of this manual described this error as “VMRESUME with a corrupted VMCS”.

b. VM-entry checks on control fields and host-state fields may be performed in any order. Thus, an indication by error number of one cause does not imply that there are not also other errors. Different processors may give different error numbers for the same VMCS.

c. Error number 7 is not used for VM entries that return from SMM that fail due to invalid VM-execution control fields in the executive VMCS. Error number 25 is used for these cases.

CHAPTER 34
MODEL-SPECIFIC REGISTERS (MSRS)

This chapter lists MSRs provided in Intel® Core™ 2 processor family, Intel® Atom™, Intel® Core™ Duo, Intel® Core™ Solo, Pentium® 4 and Intel® Xeon® processors, P6 family processors, and Pentium® processors in Tables 34-13, 34-18, and 34-19, respectively. All MSRs listed can be read with the RDMSR and written with the WRMSR instructions.

Register addresses are given in both hexadecimal and decimal. The register name is the mnemonic register name and the bit description describes individual bits in registers.

Model specific registers and its bit-fields may be supported for a finite range of processor families/models. To distinguish between different processor family and/or models, software must use CPUID.01H leaf function to query the combination of DisplayFamily and DisplayModel to determine model-specific availability of MSRs (see CPUID instruction in Chapter 3, "Instruction Set Reference, A-L" in the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*). Table 34-1 lists the signature values of DisplayFamily and DisplayModel for various processor families or processor number series.

Table 34-1. CPUID Signature Values of DisplayFamily_DisplayModel

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_3AH	Next Generation Intel Core processor
06_2DH	Next Generation Intel Xeon processor
06_2FH	Intel Xeon processor E7 family
06_2AH	Intel Xeon processor E3 family; Second Generation Intel Core i7, i5, i3 Processors 2xxx Series
06_2EH	Intel Xeon processor 7500, 6500 series
06_25H, 06_2CH	Intel Xeon processors 3600, 5600 series, Intel Core i7, i5 and i3 Processors
06_1EH, 06_1FH	Intel Core i7 and i5 Processors
06_1AH	Intel Core i7 Processor, Intel Xeon Processor 3400, 3500, 5500 series
06_1DH	Intel Xeon Processor MP 7400 series
06_17H	Intel Xeon Processor 3100, 3300, 5200, 5400 series, Intel Core 2 Quad processors 8000, 9000 series

Table 34-1. CPUID Signature (Contd.)Values of DisplayFamily_DisplayModel (Contd.)

DisplayFamily_DisplayModel	Processor Families/Processor Number Series
06_0FH	Intel Xeon Processor 3000, 3200, 5100, 5300, 7300 series, Intel Core 2 Quad processor 6000 series, Intel Core 2 Extreme 6000 series, Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors, Intel Pentium dual-core processors
06_0EH	Intel Core Duo, Intel Core Solo processors
06_0DH	Intel Pentium M processor
06_1CH	Intel Atom processor
0F_06H	Intel Xeon processor 7100, 5000 Series, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
0F_03H, 0F_04H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4, Pentium D processors
06_09H	Intel Pentium M processor
0F_02H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4 processors
0F_0H, 0F_01H	Intel Xeon Processor, Intel Xeon Processor MP, Intel Pentium 4 processors
06_7H, 06_08H, 06_0AH, 06_0BH	Intel Pentium III Xeon Processor, Intel Pentium III Processor
06_03H, 06_05H	Intel Pentium II Xeon Processor, Intel Pentium II Processor
06_01H	Intel Pentium Pro Processor
05_01H, 05_02H, 05_04H	Intel Pentium Processor, Intel Pentium Processor with MMX Technology

34.1 ARCHITECTURAL MSRS

Many MSRs have carried over from one generation of IA-32 processors to the next and to Intel 64 processors. A subset of MSRs and associated bit fields, which do not change on future processor generations, are now considered architectural MSRs. For historical reasons (beginning with the Pentium 4 processor), these “architectural MSRs” were given the prefix “IA32_”. Table 34-2 lists the architectural MSRs, their addresses, their current names, their names in previous IA-32 processors, and bit fields that are considered architectural. MSR addresses outside Table 34-2 and certain bitfields in an MSR address that may overlap with architectural MSR addresses are model-specific. Code that accesses a machine specified MSR and that is executed on a processor that does not support that MSR will generate an exception.

Architectural MSR or individual bit fields in an architectural MSR may be introduced or transitioned at the granularity of certain processor family/model or the presence of certain CPUID feature flags. The right-most column of Table 34-2 provides information on the introduction of each architectural MSR or its individual fields. This information is expressed either as signature values of "DF_DM" (see Table 34-1) or via CPUID flags.

Certain bit field position may be related to the maximum physical address width, the value of which is expressed as "MAXPHYWID" in Table 34-2. "MAXPHYWID" is reported by CPUID.8000_0008H leaf.

MSR address range between 40000000H - 400000FFH is marked as a specially reserved range. All existing and future processors will not implement any features using any MSR in this range.

Table 34-2. IA-32 Architectural MSRs

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
0H	0	IA32_P5_MC_ADDR (P5_MC_ADDR)	See Section 34.12, "MSRs in Pentium Processors."	**Pentium Processor (05_01H)**
1H	1	IA32_P5_MC_TYPE (P5_MC_TYPE)	See Section 34.12, "MSRs in Pentium Processors."	DF_DM = 05_01H
6H	6	IA32_MONITOR_FILTER_SIZE	See Section 8.10.5, "Monitor/Mwait Address Range Determination."	0F_03H
10H	16	IA32_TIME_STAMP_COUNTER (TSC)	See Section 17.12, "Time-Stamp Counter."	05_01H
17H	23	IA32_PLATFORM_ID (MSR_PLATFORM_ID)	**Platform ID. (RO)** The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.	06_01H
		49:0	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		52:50	**Platform Id. (RO)** Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7	
		63:53	Reserved.	
1BH	27	IA32_APIC_BASE (APIC_BASE)		06_01H
		7:0	Reserved	
		8	BSP flag (R/W)	
		9	Reserved	
		10	Enable x2APIC mode	06_1AH
		11	APIC Global Enable (R/W)	
		(MAXPHYWID - 1):12	APIC Base (R/W)	
		63: MAXPHYWID	Reserved	
3AH	58	IA32_FEATURE_CONTROL	**Control Features in Intel 64 Processor. (R/W)**	If CPUID.01H: ECX[bit 5 or bit 6] = 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	Lock bit (R/WO): (1 = locked). When set, locks this MSR from being written, writes to this bit will result in GP(0). Note: Once the Lock bit is set, the contents of this register cannot be modified. Therefore the lock bit must be set after configuring support for Intel Virtualization Technology and prior to transferring control to an option ROM or the OS. Hence, once the Lock bit is set, the entire IA32_FEATURE_CONTROL_MSR contents are preserved across RESET when PWRGOOD is not deasserted.	If CPUID.01H:ECX[bit 5 or bit 6] = 1
		1	Enable VMX inside SMX operation (R/WL): This bit enables a system executive to use VMX in conjunction with SMX to support Intel® Trusted Execution Technology. BIOS must set this bit only when the CPUID function 1 returns VMX feature flag and SMX feature flag set (ECX bits 5 and 6 respectively).	If CPUID.01H:ECX[bit 5 and bit 6] are set to 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		2	Enable VMX outside SMX operation (R/WL): This bit enables VMX for system executive that do not require SMX. BIOS must set this bit only when the CPUID function 1 returns VMX feature flag set (ECX bit 5).	If CPUID.01H:ECX[bit 5 or bit 6] = 1
		7:3	Reserved	
		14:8	SENTER Local Function Enables (R/WL): When set, each bit in the field represents an enable control for a corresponding SENTER function. This bit is supported only if CPUID.1:ECX.[bit 6] is set	If CPUID.01H:ECX[bit 6] = 1
		15	SENTER Global Enable (R/WL): This bit must be set to enable SENTER leaf functions. This bit is supported only if CPUID.1:ECX.[bit 6] is set	If CPUID.01H:ECX[bit 6] = 1
		63:16	Reserved	
79H	121	IA32_BIOS_UPDT_TRIG (BIOS_UPDT_TRIG)	BIOS Update Trigger (W) Executing a WRMSR instruction to this MSR causes a microcode update to be loaded into the processor. See Section 9.11.6, "Microcode Update Loader." A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
8BH	139	IA32_BIOS_SIGN_ID (BIOS_SIGN/BBL_CR_D3)	BIOS Update Signature (RO) Returns the microcode update signature following the execution of CPUID.01H. A processor may prevent writing to this MSR when loading guest states on VM entries or saving guest states on VM exits.	06_01H
		31:0	Reserved	
		63:32	It is recommended that this field be pre-loaded with 0 prior to executing CPUID. If the field remains 0 following the execution of CPUID; this indicates that no microcode update is loaded. Any non-zero value is the microcode update signature.	
9BH	155	IA32_SMM_MONITOR_CTL	SMM Monitor Configuration (R/W)	If CPUID.01H: ECX[bit 5 or bit 6] = 1
		0	Valid (R/W)	
		1	Reserved	
		2	Controls SMI unblocking by VMXOFF (see Section 29.14.4)	If IA32_VMX_MISC[bit 28])
		11:3	Reserved	
		31:12	MSEG Base (R/W)	
		63:32	Reserved	
C1H	193	IA32_PMC0 (PERFCTR0)	General Performance Counter 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
C2H	194	IA32_PMC1 (PERFCTR1)	General Performance Counter 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
C3H	195	IA32_PMC2	General Performance Counter 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
C4H	196	IA32_PMC3	General Performance Counter 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
C5H	197	IA32_PMC4	General Performance Counter 4 (R/W)	If CPUID.0AH: EAX[15:8] > 4
C6H	198	IA32_PMC5	General Performance Counter 5 (R/W)	If CPUID.0AH: EAX[15:8] > 5
C7H	199	IA32_PMC6	General Performance Counter 6 (R/W)	If CPUID.0AH: EAX[15:8] > 6
C8H	200	IA32_PMC7	General Performance Counter 7 (R/W)	If CPUID.0AH: EAX[15:8] > 7
E7H	231	IA32_MPERF	Maximum Qualified Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	**C0_MCNT: C0 Maximum Frequency Clock Count.** Increments at fixed interval (relative to TSC freq.) when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_APERF.	
E8H	232	IA32_APERF	Actual Performance Clock Counter (R/Write to clear)	If CPUID.06H: ECX[0] = 1
		63:0	**C0_ACNT: C0 Actual Frequency Clock Count.** Accumulates core clock counts at the coordinated clock frequency, when the logical processor is in C0. Cleared upon overflow / wrap-around of IA32_MPERF.	
FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved.	
		10	WC Supported when set.	
		11	SMRR Supported when set.	
		63:12	Reserved.	
174H	372	IA32_SYSENTER_CS	SYSENTER_CS_MSR (R/W)	06_01H
		15:0	CS Selector	
		63:16	Reserved.	
175H	373	IA32_SYSENTER_ESP	SYSENTER_ESP_MSR (R/W)	06_01H
176H	374	IA32_SYSENTER_EIP	SYSENTER_EIP_MSR (R/W)	06_01H
179H	377	IA32_MCG_CAP (MCG_CAP)	Global Machine Check Capability (RO)	06_01H
		7:0	Count: Number of reporting banks.	
		8	MCG_CTL_P: IA32_MCG_CTL is present if this bit is set	
		9	MCG_EXT_P: Extended machine check state registers are present if this bit is set	
		10	MCP_CMCI_P: Support for corrected MC error event is present.	06_1AH
		11	MCG_TES_P: Threshold-based error status register are present if this bit is set.	
		15:12	Reserved	
		23:16	MCG_EXT_CNT: Number of extended machine check state registers present.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		24	MCG_SER_P: The processor supports software error recovery if this bit is set.	
		63:25	Reserved.	
17AH	378	IA32_MCG_STATUS (MCG_STATUS)	Global Machine Check Status (RO)	06_01H
17BH	379	IA32_MCG_CTL (MCG_CTL)	Global Machine Check Control (R/W)	06_01H
180H-185H	384-389	Reserved		06_0EH[1]
186H	390	IA32_PERFEVTSEL0 (PERFEVTSEL0)	Performance Event Select Register 0 (R/W)	If CPUID.0AH: EAX[15:8] > 0
		7:0	Event Select: Selects a performance event logic unit.	
		15:8	UMask: Qualifies the microarchitectural condition to detect on the selected event logic.	
		16	USR: Counts while in privilege level is not ring 0.	
		17	OS: Counts while in privilege level is ring 0.	
		18	Edge: Enables edge detection if set.	
		19	PC: enables pin control.	
		20	INT: enables interrupt on counter overflow.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		21	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	
		22	EN: enables the corresponding performance counter to commence counting when this bit is set.	
		23	INV: invert the CMASK.	
		31:24	CMASK: When CMASK is not zero, the corresponding performance counter increments each cycle if the event count is greater than or equal to the CMASK.	
		63:32	Reserved.	
187H	391	IA32_PERFEVTSEL1 (PERFEVTSEL1)	Performance Event Select Register 1 (R/W)	If CPUID.0AH: EAX[15:8] > 1
188H	392	IA32_PERFEVTSEL2	Performance Event Select Register 2 (R/W)	If CPUID.0AH: EAX[15:8] > 2
189H	393	IA32_PERFEVTSEL3	Performance Event Select Register 3 (R/W)	If CPUID.0AH: EAX[15:8] > 3
18AH-197H	394-407	Reserved		06_0EH[2]
198H	408	IA32_PERF_STATUS	(RO)	0F_03H
		15:0	Current performance State Value	
		63:16	Reserved.	
199H	409	IA32_PERF_CTL	(R/W)	0F_03H

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		15:0	Target performance State Value	
		31:16	Reserved.	
		32	IDA Engage. (R/W) When set to 1: disengages IDA	06_0FH (Mobile)
		63:33	Reserved.	
19AH	410	IA32_CLOCK_MODULATION	Clock Modulation Control (R/W) See Section 14.5.3, "Software Controlled Clock Modulation."	0F_0H
		0	Extended On-Demand Clock Modulation Duty Cycle:	If CPUID.06H:EAX[5] = 1
		3:1	On-Demand Clock Modulation Duty Cycle: Specific encoded values for target duty cycle modulation.	
		4	On-Demand Clock Modulation Enable: Set 1 to enable modulation.	
		63:5	Reserved.	
19BH	411	IA32_THERM_INTERRUPT	Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the processor's thermal sensors and thermal monitor. See Section 14.5.2, "Thermal Monitor."	0F_0H
		0	High-Temperature Interrupt Enable	
		1	Low-Temperature Interrupt Enable	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		2	PROCHOT# Interrupt Enable	
		3	FORCEPR# Interrupt Enable	
		4	Critical Temperature Interrupt Enable	
		7:5	Reserved.	
		14:8	Threshold #1 Value	
		15	Threshold #1 Interrupt Enable	
		22:16	Threshold #2 Value	
		23	Threshold #2 Interrupt Enable	
		24	Power Limit Notification Enable	If CPUID.06H:EAX[4] = 1
		63:25	Reserved.	
19CH	412	IA32_THERM_STATUS	Thermal Status Information (RO) Contains status information about the processor’s thermal sensor and automatic thermal monitoring facilities. See Section 14.5.2, “Thermal Monitor”	0F_0H
		0	Thermal Status (RO):	
		1	Thermal Status Log (R/W):	
		2	PROCHOT # or FORCEPR# event (RO)	
		3	PROCHOT # or FORCEPR# log (R/WC0)	
		4	Critical Temperature Status (RO)	
		5	Critical Temperature Status log (R/WC0)	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		6	Thermal Threshold #1 Status (RO)	If CPUID.01H:ECX[8] = 1
		7	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		8	Thermal Threshold #2 Status (RO)	If CPUID.01H:ECX[8] = 1
		9	Thermal Threshold #1 log (R/WC0)	If CPUID.01H:ECX[8] = 1
		10	Power Limitation Status (RO)	If CPUID.06H:EAX[4] = 1
		11	Power Limitation log (R/WC0)	If CPUID.06H:EAX[4] = 1
		15:12	Reserved.	
		22:16	Digital Readout (RO)	If CPUID.06H:EAX[0] = 1
		26:23	Reserved.	
		30:27	Resolution in Degrees Celsius (RO)	If CPUID.06H:EAX[0] = 1
		31	Reading Valid (RO)	If CPUID.06H:EAX[0] = 1
		63:32	Reserved.	
1A0H	416	IA32_MISC_ENABLE	**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	**Fast-Strings Enable.** When set, the fast-strings feature (for REP MOVS and REP STORS) is enabled (default); when clear, fast-strings are disabled.	0F_0H
		2:1	Reserved.	
		3	**Automatic Thermal Control Circuit Enable. (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows the processor to automatically reduce power consumption in response to TCC activation. 0 = Disabled (default). Note: In some products clearing this bit might be ignored in critical thermal conditions, and TM1, TM2 and adaptive thermal throttling will still be activated.	0F_0H
		6:4	Reserved	
		7	**Performance Monitoring Available. (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled	0F_0H
		10:8	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		11	**Branch Trace Storage Unavailable. (RO)** 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported	0F_0H
		12	**Precise Event Based Sampling (PEBS) Unavailable. (RO)** 1 = PEBS is not supported; 0 = PEBS is supported.	06_0FH
		15:13	Reserved.	
		16	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 0= Enhanced Intel SpeedStep Technology disabled 1 = Enhanced Intel SpeedStep Technology enabled	06_0DH
		17	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	**Decimal**			
		18	ENABLE MONITOR FSM. (R/W) When this bit is set to 0, the MONITOR feature flag is not set (CPUID.01H:ECX[bit 3] = 0). This indicates that MONITOR/MWAIT are not supported. Software attempts to execute MONITOR/MWAIT will cause #UD when this bit is 0. When this bit is set to 1 (default), MONITOR/MWAIT are supported (CPUID.01H:ECX[bit 3] = 1). If the SSE3 feature flag ECX[0] is not set (CPUID.01H:ECX[bit 0] = 0), the OS must not attempt to alter this bit. BIOS must leave it in the default state. Writing this bit when the SSE3 feature flag is set to 0 may generate a #GP exception.	0F_03H
		21:19	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		22	**Limit CPUID Maxval. (R/W)** When this bit is set to 1, CPUID.00H returns a maximum value in EAX[7:0] of 3. BIOS should contain a setup question that allows users to specify when the installed OS does not support CPUID functions greater than 3. Before setting this bit, BIOS must execute the CPUID.0H and examine the maximum value returned in EAX[7:0]. If the maximum value is greater than 3, the bit is supported. Otherwise, the bit is not supported. Writing to this bit when the maximum value is greater than 3 may generate a #GP exception. Setting this bit may cause unexpected behavior in software that depends on the availability of CPUID leaves greater than 3.	0F_03H
		23	**xTPR Message Disable. (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority.	if CPUID.01H:ECX[14] = 1
		33:24	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		34	**XD Bit Disable. (R/W)** When set to 1, the Execute Disable Bit feature (XD Bit) is disabled and the XD Bit extended feature flag will be clear (CPUID.80000001H: EDX[20]=0). When set to a 0 (default), the Execute Disable Bit feature (if available) allows the OS to enable PAE paging and take advantage of data only pages. BIOS must not alter the contents of this bit location, if XD bit is not supported.. Writing this bit to 1 when the XD Bit extended feature flag is set to 0 may generate a #GP exception.	if CPUID.80000001 H:EDX[20] = 1
		63:35	Reserved.	
1B0H	432	IA32_ENERGY_PERF_BIAS	Performance Energy Bias Hint (R/W)	if CPUID.6H:ECX[3] = 1
		3:0	Power Policy Preference: 0 indicates preference to highest performance. 15 indicates preference to maximize energy saving.	
		63:4	Reserved.	
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package Thermal Status Information (RO) Contains status information about the package's thermal sensor. See Section 14.6, "Package Level Thermal Management."	06_2AH

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	Pkg Thermal Status (RO):	
		1	Pkg Thermal Status Log (R/W):	
		2	Pkg PROCHOT # event (RO)	
		3	Pkg PROCHOT # log (R/WC0)	
		4	Pkg Critical Temperature Status (RO)	
		5	Pkg Critical Temperature Status log (R/WC0)	
		6	Pkg Thermal Threshold #1 Status (RO)	
		7	Pkg Thermal Threshold #1 log (R/WC0)	
		8	Pkg Thermal Threshold #2 Status (RO)	
		9	Pkg Thermal Threshold #1 log (R/WC0)	
		10	Pkg Power Limitation Status (RO)	
		11	Pkg Power Limitation log (R/WC0)	
		15:12	Reserved.	
		22:16	Pkg Digital Readout (RO)	
		63:23	Reserved.	
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Pkg Thermal Interrupt Control (R/W) Enables and disables the generation of an interrupt on temperature transitions detected with the package's thermal sensor. See Section 14.6, "Package Level Thermal Management."	06_2AH

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	Pkg High-Temperature Interrupt Enable	
		1	Pkg Low-Temperature Interrupt Enable	
		2	Pkg PROCHOT# Interrupt Enable	
		3	Reserved.	
		4	Pkr Overheat Interrupt Enable	
		7:5	Reserved.	
		14:8	Pkg Threshold #1 Value	
		15	Pkg Threshold #1 Interrupt Enable	
		22:16	Pkg Threshold #2 Value	
		23	Pkg Threshold #2 Interrupt Enable	
		24	Pkg Power Limit Notification Enable	
		63:25	Reserved.	
1D9H	473	IA32_DEBUGCTL (MSR_DEBUGCTLA, MSR_DEBUGCTLB)	Trace/Profile Resource Control (R/W)	06_0EH
		0	LBR: Setting this bit to 1 enables the processor to record a running trace of the most recent branches taken by the processor in the LBR stack.	06_01H
		1	BTF: Setting this bit to 1 enables the processor to treat EFLAGS.TF as single-step on branches instead of single-step on instructions.	06_01H
		5:2	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		6	TR: Setting this bit to 1 enables branch trace messages to be sent.	06_0EH
		7	BTS: Setting this bit enables branch trace messages (BTMs) to be logged in a BTS buffer.	06_0EH
		8	BTINT: When clear, BTMs are logged in a BTS buffer in circular fashion. When this bit is set, an interrupt is generated by the BTS facility when the BTS buffer is full.	06_0EH
		9	1: BTS_OFF_OS: When set, BTS or BTM is skipped if CPL = 0.	06_0FH
		10	BTS_OFF_USR: When set, BTS or BTM is skipped if CPL > 0.	06_0FH
		11	FREEZE_LBRS_ON_PMI: When set, the LBR stack is frozen on a PMI request.	If CPUID.01H: ECX[15] = 1 and CPUID.0AH: EAX[7:0] > 1
		12	FREEZE_PERFMON_ON_PMI: When set, each ENABLE bit of the global counter control MSR are frozen (address 3BFH) on a PMI request	If CPUID.01H: ECX[15] = 1 and CPUID.0AH: EAX[7:0] > 1
		13	ENABLE_UNCORE_PMI: When set, enables the logical processor to receive and generate PMI on behalf of the uncore.	06_1AH
		14	FREEZE_WHILE_SMM: When set, freezes perfmon and trace messages while in SMM.	if IA32_PERF_CAPABILITIES[12] = '1
		63:15	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
1F2H	498	IA32_SMRR_PHYSBASE	**SMRR Base Address. (Writeable only in SMM)** Base address of SMM memory range.	06_1AH
		7:0	Type. Specifies memory type of the range.	
		11:8	Reserved.	
		31:12	**PhysBase.** SMRR physical Base Address.	
		63:32	Reserved.	
1F3H	499	IA32_SMRR_PHYSMASK	**SMRR Range Mask. (Writeable only in SMM)** Range Mask of SMM memory range.	06_1AH
		10:0	Reserved.	
		11	**Valid.** Enable range mask	
		31:12	**PhysMask.** SMRR address range mask.	
		63:32	Reserved.	
1F8H	504	IA32_PLATFORM_DCA_CAP	DCA Capability (R)	06_0FH
1F9H	505	IA32_CPU_DCA_CAP	If set, CPU supports Prefetch-Hint type.	
1FAH	506	IA32_DCA_0_CAP	DCA type 0 Status and Control register.	06_2EH
		0	DCA_ACTIVE: Set by HW when DCA is fuse-enabled and no defeatures are set.	06_2EH
		2:1	TRANSACTION	06_2EH
		6:3	DCA_TYPE	06_2EH
		10:7	DCA_QUEUE_SIZE	06_2EH
		12:11	Reserved.	06_2EH

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		16:13	DCA_DELAY: Writes will update the register but have no HW side-effect.	06_2EH
		23:17	Reserved.	06_2EH
		24	SW_BLOCK: SW can request DCA block by setting this bit.	06_2EH
		25	Reserved.	06_2EH
		26	HW_BLOCK: Set when DCA is blocked by HW (e.g. CR0.CD = 1).	06_2EH
		31:27	Reserved.	06_2EH
200H	512	IA32_MTRR_PHYSBASE0 (MTRRphysBase0)	See Section 11.11.2.3, “Variable Range MTRRs.”	06_01H
201H	513	IA32_MTRR_PHYSMASK0	MTRRphysMask0	06_01H
202H	514	IA32_MTRR_PHYSBASE1	MTRRphysBase1	06_01H
203H	515	IA32_MTRR_PHYSMASK1	MTRRphysMask1	06_01H
204H	516	IA32_MTRR_PHYSBASE2	MTRRphysBase2	06_01H
205H	517	IA32_MTRR_PHYSMASK2	MTRRphysMask2	06_01H
206H	518	IA32_MTRR_PHYSBASE3	MTRRphysBase3	06_01H
207H	519	IA32_MTRR_PHYSMASK3	MTRRphysMask3	06_01H
208H	520	IA32_MTRR_PHYSBASE4	MTRRphysBase4	06_01H
209H	521	IA32_MTRR_PHYSMASK4	MTRRphysMask4	06_01H
20AH	522	IA32_MTRR_PHYSBASE5	MTRRphysBase5	06_01H
20BH	523	IA32_MTRR_PHYSMASK5	MTRRphysMask5	06_01H
20CH	524	IA32_MTRR_PHYSBASE6	MTRRphysBase6	06_01H
20DH	525	IA32_MTRR_PHYSMASK6	MTRRphysMask6	06_01H
20EH	526	IA32_MTRR_PHYSBASE7	MTRRphysBase7	06_01H
20FH	527	IA32_MTRR_PHYSMASK7	MTRRphysMask7	06_01H
210H	528	IA32_MTRR_PHYSBASE8	MTRRphysBase8	if IA32_MTRR_CAP[7:0] > 8

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
211H	529	IA32_MTRR_PHYSMASK8	MTRRphysMask8	if IA32_MTRR_CAP[7:0] > 8
212H	530	IA32_MTRR_PHYSBASE9	MTRRphysBase9	if IA32_MTRR_CAP[7:0] > 9
213H	531	IA32_MTRR_PHYSMASK9	MTRRphysMask9	if IA32_MTRR_CAP[7:0] > 9
250H	592	IA32_MTRR_FIX64K_00000	MTRRfix64K_00000	06_01H
258H	600	IA32_MTRR_FIX16K_80000	MTRRfix16K_80000	06_01H
259H	601	IA32_MTRR_FIX16K_A0000	MTRRfix16K_A0000	06_01H
268H	616	IA32_MTRR_FIX4K_C0000 (MTRRfix4K_C0000)	See Section 11.11.2.2, “Fixed Range MTRRs.”	06_01H
269H	617	IA32_MTRR_FIX4K_C8000	MTRRfix4K_C8000	06_01H
26AH	618	IA32_MTRR_FIX4K_D0000	MTRRfix4K_D0000	06_01H
26BH	619	IA32_MTRR_FIX4K_D8000	MTRRfix4K_D8000	06_01H
26CH	620	IA32_MTRR_FIX4K_E0000	MTRRfix4K_E0000	06_01H
26DH	621	IA32_MTRR_FIX4K_E8000	MTRRfix4K_E8000	06_01H
26EH	622	IA32_MTRR_FIX4K_F0000	MTRRfix4K_F0000	06_01H
26FH	623	IA32_MTRR_FIX4K_F8000	MTRRfix4K_F8000	06_01H
277H	631	IA32_PAT	IA32_PAT (R/W)	06_05H
		2:0	PA0	
		7:3	Reserved	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		10:8	PA1	
		15:11	Reserved	
		18:16	PA2	
		23:19	Reserved	
		26:24	PA3	
		31:27	Reserved	
		34:32	PA4	
		39:35	Reserved	
		42:40	PA5	
		47:43	Reserved	
		50:48	PA6	
		55:51	Reserved	
		58:56	PA7	
		63:59	Reserved	
280H	640	IA32_MC0_CTL2	(R/W)	06_1AH
		14:0	Corrected error count threshold	
		29:15	Reserved	
		30	CMCI_EN	
		63:31	Reserved	
281H	641	IA32_MC1_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
282H	642	IA32_MC2_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
283H	643	IA32_MC3_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
284H	644	IA32_MC4_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
285H	645	IA32_MC5_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
286H	646	IA32_MC6_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
287H	647	IA32_MC7_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
288H	648	IA32_MC8_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_1AH
289H	649	IA32_MC9_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
28AH	650	IA32_MC10_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
28BH	651	IA32_MC11_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
28CH	652	IA32_MC12_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
28DH	653	IA32_MC13_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
28EH	654	IA32_MC14_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
28FH	655	IA32_MC15_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
290H	656	IA32_MC16_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
291H	657	IA32_MC17_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
292H	658	IA32_MC18_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
293H	659	IA32_MC19_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
294H	660	IA32_MC20_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
295H	661	IA32_MC21_CTL2	(R/W) same fields as IA32_MC0_CTL2	06_2EH
2FFH	767	IA32_MTRR_DEF_TYPE	MTRRdefType (R/W)	06_01H
		2:0	Default Memory Type	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		9:3	Reserved	
		10	Fixed Range MTRR Enable	
		11	MTRR Enable	
		63:12	Reserved	
309H	777	IA32_FIXED_CTR0 (MSR_PERF_FIXED_CTR0)	Fixed-Function Performance Counter 0 (R/W): Counts Instr_Retired.Any	If CPUID.0AH: EDX[4:0] > 0
30AH	778	IA32_FIXED_CTR1 (MSR_PERF_FIXED_CTR1)	Fixed-Function Performance Counter 1 0 (R/W): Counts CPU_CLK_Unhalted.Core	If CPUID.0AH: EDX[4:0] > 1
30BH	779	IA32_FIXED_CTR2 (MSR_PERF_FIXED_CTR2)	Fixed-Function Performance Counter 0 0 (R/W): Counts CPU_CLK_Unhalted.Ref	If CPUID.0AH: EDX[4:0] > 2
345H	837	IA32_PERF_CAPABILITIES	RO	If CPUID.01H: ECX[15] = 1
		5:0	LBR format	
		6	PEBS Trap	
		7	PEBSSaveArchRegs	
		11:8	PEBS Record Format	
		12	1: Freeze while SMM is supported	
		13	1: Full width of counter writable via IA32_A_PMCx	
		63:14	Reserved	
38DH	909	IA32_FIXED_CTR_CTRL (MSR_PERF_FIXED_CTR_CTRL)	Fixed-Function Performance Counter Control (R/W) Counter increments while the results of ANDing respective enable bit in IA32_PERF_GLOBAL_CTRL with the corresponding OS or USR bits in this MSR is true.	If CPUID.0AH: EAX[7:0] > 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		0	EN0_OS: Enable Fixed Counter 0 to count while CPL = 0	
		1	EN0_Usr: Enable Fixed Counter 0 to count while CPL > 0	
		2	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		3	EN0_PMI: Enable PMI when fixed counter 0 overflows	
		4	EN1_OS: Enable Fixed Counter 1to count while CPL = 0	
		5	EN1_Usr: Enable Fixed Counter 1to count while CPL > 0	
		6	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		7	EN1_PMI: Enable PMI when fixed counter 1 overflows	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		8	EN2_OS: Enable Fixed Counter 2 to count while CPL = 0	
		9	EN2_Usr: Enable Fixed Counter 2 to count while CPL > 0	
		10	AnyThread: When set to 1, it enables counting the associated event conditions occurring across all logical processors sharing a processor core. When set to 0, the counter only increments the associated event conditions occurring in the logical processor which programmed the MSR.	If CPUID.0AH: EAX[7:0] > 2
		11	EN2_PMI: Enable PMI when fixed counter 2 overflows	
		63:12	Reserved	
38EH	910	IA32_PERF_GLOBAL_STATUS (MSR_PERF_GLOBAL_STATUS)	Global Performance Counter Status (RO)	If CPUID.0AH: EAX[7:0] > 0
		0	Ovf_PMC0: Overflow status of IA32_PMC0	If CPUID.0AH: EAX[7:0] > 0
		1	Ovf_PMC1: Overflow status of IA32_PMC1	If CPUID.0AH: EAX[7:0] > 0
		2	Ovf_PMC2: Overflow status of IA32_PMC2	06_2EH
		3	Ovf_PMC3: Overflow status of IA32_PMC3	06_2EH
		31:4	Reserved	
		32	Ovf_FixedCtr0: Overflow status of IA32_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		33	Ovf_FixedCtr1: Overflow status of IA32_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	Ovf_FixedCtr2: Overflow status of IA32_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		60:35	Reserved	
		61	Ovf_Uncore: Uncore counter overflow status	If CPUID.0AH: EAX[7:0] > 2
		62	OvfBuf: DS SAVE area Buffer overflow status	If CPUID.0AH: EAX[7:0] > 0
		63	CondChg: status bits of this register has changed	If CPUID.0AH: EAX[7:0] > 0
38FH	911	IA32_PERF_GLOBAL_CTRL (MSR_PERF_GLOBAL_CTRL)	Global Performance Counter Control (R/W) Counter increments while the result of ANDing respective enable bit in this MSR with the corresponding OS or USR bits in the general-purpose or fixed counter control MSR is true.	If CPUID.0AH: EAX[7:0] > 0
		0	EN_PMC0	If CPUID.0AH: EAX[7:0] > 0
		1	EN_PMC1	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved	
		32	EN_FIXED_CTR0	If CPUID.0AH: EAX[7:0] > 1
		33	EN_FIXED_CTR1	If CPUID.0AH: EAX[7:0] > 1
		34	EN_FIXED_CTR2	If CPUID.0AH: EAX[7:0] > 1
		63:35	Reserved	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL (MSR_PERF_GLOBAL_OVF_CTRL)	Global Performance Counter Overflow Control (R/W)	If CPUID.0AH: EAX[7:0] > 0
		0	Set 1 to Clear Ovf_PMC0 bit	If CPUID.0AH: EAX[7:0] > 0
		1	Set 1 to Clear Ovf_PMC1 bit	If CPUID.0AH: EAX[7:0] > 0
		31:2	Reserved	
		32	Set 1 to Clear Ovf_FIXED_CTR0 bit	If CPUID.0AH: EAX[7:0] > 1
		33	Set 1 to Clear Ovf_FIXED_CTR1 bit	If CPUID.0AH: EAX[7:0] > 1
		34	Set 1 to Clear Ovf_FIXED_CTR2 bit	If CPUID.0AH: EAX[7:0] > 1
		60:35	Reserved	
		61	Set 1 to Clear Ovf_Uncore: bit	06_2EH
		62	Set 1 to Clear OvfBuf: bit	If CPUID.0AH: EAX[7:0] > 0
		63	Set to 1to clear CondChg: bit	If CPUID.0AH: EAX[7:0] > 0
3F1H	1009	IA32_PEBS_ENABLE	PEBS Control (R/W)	
		0	Enable PEBS on IA32_PMC0	06_0FH
		1-3	Reserved or Model specific	
		31:4	Reserved	
		35-32	Reserved or Model specific	
		63:36	Reserved	
400H	1024	IA32_MC0_CTL	MC0_CTL	P6 Family Processors
401H	1025	IA32_MC0_STATUS	MC0_STATUS	P6 Family Processors
402H	1026	IA32_MC0_ADDR[1]	MC0_ADDR	P6 Family Processors

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
403H	1027	IA32_MC0_MISC	MC0_MISC	P6 Family Processors
404H	1028	IA32_MC1_CTL	MC1_CTL	P6 Family Processors
405H	1029	IA32_MC1_STATUS	MC1_STATUS	P6 Family Processors
406H	1030	IA32_MC1_ADDR[2]	MC1_ADDR	P6 Family Processors
407H	1031	IA32_MC1_MISC	MC1_MISC	P6 Family Processors
408H	1032	IA32_MC2_CTL	MC2_CTL	P6 Family Processors
409H	1033	IA32_MC2_STATUS	MC2_STATUS	P6 Family Processors
40AH	1034	IA32_MC2_ADDR[1]	MC2_ADDR	P6 Family Processors
40BH	1035	IA32_MC2_MISC	MC2_MISC	P6 Family Processors
40CH	1036	IA32_MC3_CTL	MC3_CTL	P6 Family Processors
40DH	1037	IA32_MC3_STATUS	MC3_STATUS	P6 Family Processors
40EH	1038	IA32_MC3_ADDR[1]	MC3_ADDR	P6 Family Processors
40FH	1039	IA32_MC3_MISC	MC3_MISC	P6 Family Processors
410H	1040	IA32_MC4_CTL	MC4_CTL	P6 Family Processors
411H	1041	IA32_MC4_STATUS	MC4_STATUS	P6 Family Processors
412H	1042	IA32_MC4_ADDR[1]	MC4_ADDR	P6 Family Processors
413H	1043	IA32_MC4_MISC	MC4_MISC	P6 Family Processors

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
414H	1044	IA32_MC5_CTL	MC5_CTL	06_0FH
415H	1045	IA32_MC5_STATUS	MC5_STATUS	06_0FH
416H	1046	IA32_MC5_ADDR[1]	MC5_ADDR	06_0FH
417H	1047	IA32_MC5_MISC	MC5_MISC	06_0FH
418H	1048	IA32_MC6_CTL	MC6_CTL	06_1DH
419H	1049	IA32_MC6_STATUS	MC6_STATUS	06_1DH
41AH	1050	IA32_MC6_ADDR[1]	MC6_ADDR	06_1DH
41BH	1051	IA32_MC6_MISC	MC6_MISC	06_1DH
41CH	1052	IA32_MC7_CTL	MC7_CTL	06_1AH
41DH	1053	IA32_MC7_STATUS	MC7_STATUS	06_1AH
41EH	1054	IA32_MC7_ADDR[1]	MC7_ADDR	06_1AH
41FH	1055	IA32_MC7_MISC	MC7_MISC	06_1AH
420H	1056	IA32_MC8_CTL	MC8_CTL	06_1AH
421H	1057	IA32_MC8_STATUS	MC8_STATUS	06_1AH
422H	1058	IA32_MC8_ADDR[1]	MC8_ADDR	06_1AH
423H	1059	IA32_MC8_MISC	MC8_MISC	06_1AH
424H	1060	IA32_MC9_CTL	MC9_CTL	06_2EH
425H	1061	IA32_MC9_STATUS	MC9_STATUS	06_2EH
426H	1062	IA32_MC9_ADDR[1]	MC9_ADDR	06_2EH
427H	1063	IA32_MC9_MISC	MC9_MISC	06_2EH
428H	1064	IA32_MC10_CTL	MC10_CTL	06_2EH
429H	1065	IA32_MC10_STATUS	MC10_STATUS	06_2EH
42AH	1066	IA32_MC10_ADDR[1]	MC10_ADDR	06_2EH
42BH	1067	IA32_MC10_MISC	MC10_MISC	06_2EH
42CH	1068	IA32_MC11_CTL	MC11_CTL	06_2EH
42DH	1069	IA32_MC11_STATUS	MC11_STATUS	06_2EH
42EH	1070	IA32_MC11_ADDR[1]	MC11_ADDR	06_2EH
42FH	1071	IA32_MC11_MISC	MC11_MISC	06_2EH
430H	1072	IA32_MC12_CTL	MC12_CTL	06_2EH

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
431H	1073	IA32_MC12_STATUS	MC12_STATUS	06_2EH
432H	1074	IA32_MC12_ADDR[1]	MC12_ADDR	06_2EH
433H	1075	IA32_MC12_MISC	MC12_MISC	06_2EH
434H	1076	IA32_MC13_CTL	MC13_CTL	06_2EH
435H	1077	IA32_MC13_STATUS	MC13_STATUS	06_2EH
436H	1078	IA32_MC13_ADDR[1]	MC13_ADDR	06_2EH
437H	1079	IA32_MC13_MISC	MC13_MISC	06_2EH
438H	1080	IA32_MC14_CTL	MC14_CTL	06_2EH
439H	1081	IA32_MC14_STATUS	MC14_STATUS	06_2EH
43AH	1082	IA32_MC14_ADDR[1]	MC14_ADDR	06_2EH
43BH	1083	IA32_MC14_MISC	MC14_MISC	06_2EH
43CH	1084	IA32_MC15_CTL	MC15_CTL	06_2EH
43DH	1085	IA32_MC15_STATUS	MC15_STATUS	06_2EH
43EH	1086	IA32_MC15_ADDR[1]	MC15_ADDR	06_2EH
43FH	1087	IA32_MC15_MISC	MC15_MISC	06_2EH
440H	1088	IA32_MC16_CTL	MC16_CTL	06_2EH
441H	1089	IA32_MC16_STATUS	MC16_STATUS	06_2EH
442H	1090	IA32_MC16_ADDR[1]	MC16_ADDR	06_2EH
443H	1091	IA32_MC16_MISC	MC16_MISC	06_2EH
444H	1092	IA32_MC17_CTL	MC17_CTL	06_2EH
445H	1093	IA32_MC17_STATUS	MC17_STATUS	06_2EH
446H	1094	IA32_MC17_ADDR[1]	MC17_ADDR	06_2EH
447H	1095	IA32_MC17_MISC	MC17_MISC	06_2EH
448H	1096	IA32_MC18_CTL	MC18_CTL	06_2EH
449H	1097	IA32_MC18_STATUS	MC18_STATUS	06_2EH
44AH	1098	IA32_MC18_ADDR[1]	MC18_ADDR	06_2EH
44BH	1099	IA32_MC18_MISC	MC18_MISC	06_2EH
44CH	1100	IA32_MC19_CTL	MC19_CTL	06_2EH
44DH	1101	IA32_MC19_STATUS	MC19_STATUS	06_2EH

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
44EH	1102	IA32_MC19_ADDR[1]	MC19_ADDR	06_2EH
44FH	1103	IA32_MC19_MISC	MC19_MISC	06_2EH
450H	1104	IA32_MC20_CTL	MC20_CTL	06_2EH
451H	1105	IA32_MC20_STATUS	MC20_STATUS	06_2EH
452H	1106	IA32_MC20_ADDR[1]	MC20_ADDR	06_2EH
453H	1107	IA32_MC20_MISC	MC20_MISC	06_2EH
454H	1108	IA32_MC21_CTL	MC21_CTL	06_2EH
455H	1109	IA32_MC21_STATUS	MC21_STATUS	06_2EH
456H	1110	IA32_MC21_ADDR[1]	MC21_ADDR	06_2EH
457H	1111	IA32_MC21_MISC	MC21_MISC	06_2EH
480H	1152	IA32_VMX_BASIC	**Reporting Register of Basic VMX Capabilities. (R/O)** See Appendix A.1, "Basic VMX Information."	If CPUID.01H:ECX.[bit 5] = 1
481H	1153	IA32_VMX_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If CPUID.01H:ECX.[bit 5] = 1
482H	1154	IA32_VMX_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If CPUID.01H:ECX.[bit 5] = 1
483H	1155	IA32_VMX_EXIT_CTLS	**Capability Reporting Register of VM-exit Controls. (R/O)** See Appendix A.4, "VM-Exit Controls."	If CPUID.01H:ECX.[bit 5] = 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
484H	1156	IA32_VMX_ENTRY_CTLS	**Capability Reporting Register of VM-entry Controls. (R/O)** See Appendix A.5, “VM-Entry Controls.”	If CPUID.01H:ECX.[bit 5] = 1
485H	1157	IA32_VMX_MISC	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Appendix A.6, “Miscellaneous Data.”	If CPUID.01H:ECX.[bit 5] = 1
486H	1158	IA32_VMX_CR0_FIXED0	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Appendix A.7, “VMX-Fixed Bits in CR0.”	If CPUID.01H:ECX.[bit 5] = 1
487H	1159	IA32_VMX_CR0_FIXED1	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Appendix A.7, “VMX-Fixed Bits in CR0.”	If CPUID.01H:ECX.[bit 5] = 1
488H	1160	IA32_VMX_CR4_FIXED0	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Appendix A.8, “VMX-Fixed Bits in CR4.”	If CPUID.01H:ECX.[bit 5] = 1
489H	1161	IA32_VMX_CR4_FIXED1	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Appendix A.8, “VMX-Fixed Bits in CR4.”	If CPUID.01H:ECX.[bit 5] = 1
48AH	1162	IA32_VMX_VMCS_ENUM	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Appendix A.9, “VMCS Enumeration.”	If CPUID.01H:ECX.[bit 5] = 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
48BH	1163	IA32_VMX_PROCBASED_CTLS2	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3.3, "Secondary Processor-Based VM-Execution Controls."	If (CPUID.01H:ECX.[bit 5] and IA32_VMX_PROCBASED_CTLS[bit 63])
48CH	1164	IA32_VMX_EPT_VPID_CAP	**Capability Reporting Register of EPT and VPID. (R/O)** See Appendix A.10, "VPID and EPT Capabilities."	If (CPUID.01H:ECX.[bit 5], IA32_VMX_PROCBASED_CTLS[bit 63], and either IA32_VMX_PROCBASED_CTLS2[bit 33] or IA32_VMX_PROCBASED_CTLS2[bit 37])
48DH	1165	IA32_VMX_TRUE_PINBASED_CTLS	**Capability Reporting Register of Pin-based VM-execution Flex Controls. (R/O)** See Appendix A.3.1, "Pin-Based VM-Execution Controls."	If (CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC [bit 55])
48EH	1166	IA32_VMX_TRUE_PROCBASED_CTLS	**Capability Reporting Register of Primary Processor-based VM-execution Flex Controls. (R/O)** See Appendix A.3.2, "Primary Processor-Based VM-Execution Controls."	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC [bit 55])

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
48FH	1167	IA32_VMX_TRUE_EXIT_CTLS	**Capability Reporting Register of VM-exit Flex Controls. (R/O)** See Appendix A.4, "VM-Exit Controls."	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC [bit 55])
490H	1168	IA32_VMX_TRUE_ENTRY_CTLS	**Capability Reporting Register of VM-entry Flex Controls. (R/O)** See Appendix A.5, "VM-Entry Controls."	If(CPUID.01H:ECX.[bit 5] = 1 and IA32_VMX_BASIC [bit 55])
4C1H	1217	IA32_A_PMC0	Full Width Writable IA32_PMC0 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 0) & IA32_PERF_CAPABILITIES[13] = 1
4C2H	1218	IA32_A_PMC1	Full Width Writable IA32_PMC1 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 1) & IA32_PERF_CAPABILITIES[13] = 1
4C3H	1219	IA32_A_PMC2	Full Width Writable IA32_PMC2 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 2) & IA32_PERF_CAPABILITIES[13] = 1
4C4H	1220	IA32_A_PMC3	Full Width Writable IA32_PMC3 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 3) & IA32_PERF_CAPABILITIES[13] = 1
4C5H	1221	IA32_A_PMC4	Full Width Writable IA32_PMC4 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 4) & IA32_PERF_CAPABILITIES[13] = 1
4C6H	1222	IA32_A_PMC5	Full Width Writable IA32_PMC5 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 5) & IA32_PERF_CAPABILITIES[13] = 1

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
4C7H	1223	IA32_A_PMC6	Full Width Writable IA32_PMC6 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 6) & IA32_PERF_CAPABILITIES[13] = 1
4C8H	1224	IA32_A_PMC7	Full Width Writable IA32_PMC7 Alias (R/W)	(If CPUID.0AH: EAX[15:8] > 7) & IA32_PERF_CAPABILITIES[13] = 1
600H	1536	IA32_DS_AREA	**DS Save Area. (R/W)** Points to the linear address of the first byte of the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.9.4, "Debug Store (DS) Mechanism."	0F_0H
		63:0	The linear address of the first byte of the DS buffer management area, if IA-32e mode is active.	
		31:0	The linear address of the first byte of the DS buffer management area, if not in IA-32e mode.	
		63:32	Reserved iff not in IA-32e mode.	
6E0H	1760	IA32_TSC_DEADLINE	**TSC Target of Local APIC's TSC Deadline Mode. (R/W)**	If(CPUID.01H:ECX.[bit 25] = 1
802H	2050	IA32_X2APIC_APICID	**x2APIC ID Register. (R/O)** See x2APIC Specification	If (CPUID.01H:ECX.[bit 21] = 1)
803H	2051	IA32_X2APIC_VERSION	**x2APIC Version Register. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
808H	2056	IA32_X2APIC_TPR	**x2APIC Task Priority Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
80AH	2058	IA32_X2APIC_PPR	**x2APIC Processor Priority Register. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80BH	2059	IA32_X2APIC_EOI	**x2APIC EOI Register. (W/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80DH	2061	IA32_X2APIC_LDR	**x2APIC Logical Destination Register. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
80FH	2063	IA32_X2APIC_SIVR	**x2APIC Spurious Interrupt Vector Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
810H	2064	IA32_X2APIC_ISR0	**x2APIC In-Service Register Bits 31:0. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
811H	2065	IA32_X2APIC_ISR1	**x2APIC In-Service Register Bits 63:32. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
812H	2066	IA32_X2APIC_ISR2	**x2APIC In-Service Register Bits 95:64. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
813H	2067	IA32_X2APIC_ISR3	**x2APIC In-Service Register Bits 127:96. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
814H	2068	IA32_X2APIC_ISR4	**x2APIC In-Service Register Bits 159:128. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
815H	2069	IA32_X2APIC_ISR5	**x2APIC In-Service Register Bits 191:160. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
816H	2070	IA32_X2APIC_ISR6	**x2APIC In-Service Register Bits 223:192. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
817H	2071	IA32_X2APIC_ISR7	**x2APIC In-Service Register Bits 255:224. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
818H	2072	IA32_X2APIC_TMR0	**x2APIC Trigger Mode Register Bits 31:0. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
819H	2073	IA32_X2APIC_TMR1	**x2APIC Trigger Mode Register Bits 63:32. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81AH	2074	IA32_X2APIC_TMR2	**x2APIC Trigger Mode Register Bits 95:64. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81BH	2075	IA32_X2APIC_TMR3	**x2APIC Trigger Mode Register Bits 127:96. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81CH	2076	IA32_X2APIC_TMR4	**x2APIC Trigger Mode Register Bits 159:128 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81DH	2077	IA32_X2APIC_TMR5	**x2APIC Trigger Mode Register Bits 191:160 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81EH	2078	IA32_X2APIC_TMR6	**x2APIC Trigger Mode Register Bits 223:192 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
81FH	2079	IA32_X2APIC_TMR7	**x2APIC Trigger Mode Register Bits 255:224 (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
820H	2080	IA32_X2APIC_IRR0	**x2APIC Interrupt Request Register Bits 31:0. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
821H	2081	IA32_X2APIC_IRR1	**x2APIC Interrupt Request Register Bits 63:32. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
822H	2082	IA32_X2APIC_IRR2	**x2APIC Interrupt Request Register Bits 95:64. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
823H	2083	IA32_X2APIC_IRR3	**x2APIC Interrupt Request Register Bits 127:96. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
824H	2084	IA32_X2APIC_IRR4	**x2APIC Interrupt Request Register Bits 159:128. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
825H	2085	IA32_X2APIC_IRR5	**x2APIC Interrupt Request Register Bits 191:160. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
826H	2086	IA32_X2APIC_IRR6	**x2APIC Interrupt Request Register Bits 223:192. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
827H	2087	IA32_X2APIC_IRR7	**x2APIC Interrupt Request Register Bits 255:224. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
828H	2088	IA32_X2APIC_ESR	**x2APIC Error Status Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
82FH	2095	IA32_X2APIC_LVT_CMCI	**x2APIC LVT Corrected Machine Check Interrupt Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
830H	2096	IA32_X2APIC_ICR	**x2APIC Interrupt Command Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
832H	2098	IA32_X2APIC_LVT_TIMER	**x2APIC LVT Timer Interrupt Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
833H	2099	IA32_X2APIC_LVT_THERMAL	**x2APIC LVT Thermal Sensor Interrupt Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
834H	2100	IA32_X2APIC_LVT_PMI	**x2APIC LVT Performance Monitor Interrupt Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
835H	2101	IA32_X2APIC_LVT_LINT0	**x2APIC LVT LINT0 Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
836H	2102	IA32_X2APIC_LVT_LINT1	**x2APIC LVT LINT1 Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
837H	2103	IA32_X2APIC_LVT_ERROR	**x2APIC LVT Error Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
838H	2104	IA32_X2APIC_INIT_COUNT	**x2APIC Initial Count Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
839H	2105	IA32_X2APIC_CUR_COUNT	**x2APIC Current Count Register. (R/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
83EH	2110	IA32_X2APIC_DIV_CONF	**x2APIC Divide Configuration Register. (R/W)**	If (CPUID.01H:ECX.[bit 21] = 1)
83FH	2111	IA32_X2APIC_SELF_IPI	**x2APIC Self IPI Register. (W/O)**	If (CPUID.01H:ECX.[bit 21] = 1)
4000_0000H - 4000_00FFH		Reserved MSR Address Space	**All existing and future processors will not implement MSR in this range.**	
C000_0080H		IA32_EFER	**Extended Feature Enables.**	If (CPUID.80000001.EDX.[bit 20] or CPUID.80000001.EDX.[bit29])
		0	**SYSCALL Enable. (R/W)** Enables SYSCALL/SYSRET instructions in 64-bit mode.	
		7:1	Reserved.	
		8	**IA-32e Mode Enable. (R/W)** Enables IA-32e mode operation.	
		9	Reserved.	

Table 34-2. IA-32 Architectural MSRs (Contd.)

Register Address		Architectural MSR Name and bit fields (Former MSR Name)	MSR/Bit Description	Introduced as Architectural MSR
Hex	Decimal			
		10	**IA-32e Mode Active. (R)** Indicates IA-32e mode is active when set.	
		11	Execute Disable Bit Enable. (R)	
		63:12	Reserved.	
C000_ 0081H		IA32_STAR	**System Call Target Address. (R/W)**	If CPUID.80000001. EDX.[bit 29] = 1
C000_ 0082H		IA32_LSTAR	**IA-32e Mode System Call Target Address. (R/W)**	If CPUID.80000001. EDX.[bit 29] = 1
C000_ 0084H		IA32_FMASK	**System Call Flag Mask. (R/W)**	If CPUID.80000001. EDX.[bit 29] = 1
C000_ 0100H		IA32_FS_BASE	**Map of BASE Address of FS. (R/W)**	If CPUID.80000001. EDX.[bit 29] = 1
C000_ 0101H		IA32_GS_BASE	**Map of BASE Address of GS. (R/W)**	If CPUID.80000001. EDX.[bit 29] = 1
C000_ 0102H		IA32_KERNEL_GS_BASE	**Swap Target of BASE Address of GS. (R/W)**	If CPUID.80000001. EDX.[bit 29] = 1
C000_ 0103H		IA32_TSC_AUX	Auxiliary TSC (RW)	If CPUID.80000001 H: EDX[27] = 1
		31:0	AUX: Auxiliary signature of TSC	
		63:32	Reserved.	

NOTES:

1. In processors based on Intel NetBurst® microarchitecture, MSR addresses 180H-197H are supported, software must treat them as model-specific. Starting with Intel Core Duo processors, MSR addresses 180H-185H, 188H-197H are reserved.

2. The *_ADDR MSRs may or may not be present; this depends on flag settings in IA32_MC*i*_STATUS. See Section 15.3.2.3 and Section 15.3.2.4 for more information.

34.2 MSRS IN THE INTEL® CORE™ 2 PROCESSOR FAMILY

Table 34-3 lists model-specific registers (MSRs) for Intel Core 2 processor family and for Intel Xeon processors based on Intel Core microarchitecture, architectural MSR addresses are also included in Table 34-3. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_0FH, see Table 34-1.

MSRs listed in Table 34-2 and Table 34-3 are also supported by processors based on the Enhanced Intel Core microarchitecture. Processors based on the Enhanced Intel Core microarchitecture have the CPUID signature DisplayFamily_DisplayModel of 06_17H.

The column "Shared/Unique" applies to multi-core processors based on Intel Core microarchitecture. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ ADDR	Unique	See Section 34.12, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_ TYPE	Unique	See Section 34.12, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_ FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination." andTable 34-2
10H	16	IA32_TIME_ STAMP_COUNTER	Unique	See Section 17.12, "Time-Stamp Counter," and see Table 34-2
17H	23	IA32_PLATFORM_I D	Shared	**Platform ID. (R)** See Table 34-2.
17H	23	MSR_PLATFORM_I D	Shared	**Model Specific Platform ID. (R)**
		7:0		Reserved.
		12:8		**Maximum Qualified Ratio. (R)** The maximum allowed bus ratio.
		49:13		Reserved.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		52:50		See Table 34-2.
		63:53		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location." and Table 34-2
2AH	42	MSR_EBL_CR_ POWERON	Shared	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved
		1		**Data Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		**Response Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		3		**MCERR# Drive Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		**Address Parity Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		5		Reserved
		6		Reserved
		7		**BINIT# Driver Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		**Output Tri-state Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		9		**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		10		**MCERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		11		Intel TXT Capable Chipset. (R/O) 1 = Present; 0 = Not Present
		12		**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		13		**Reserved**
		14		**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved.
		17:16		**APIC Cluster ID. (R/O)**
		18		**N/2 Non-Integer Bus Ratio. (R/O)** 0 = Integer ratio; 1 = Non-integer ratio
		19		Reserved.
		21: 20		**Symmetric Arbitration ID. (R/O)**
		26:22		**Integer Bus Frequency Ratio. (R/O)**
3AH	58	IA32_FEATURE_ CONTROL	Unique	**Control Features in Intel 64Processor. (R/W).** See Table 34-2.
		3	Unique	**SMRR Enable. (R/WL).** When this bit is set and the lock bit is set makes the SMRR_PHYS_BASE and SMRR_PHYS_MASK registers read visible and writeable while in SMM.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
40H	64	MSR_ LASTBRANCH_0_F ROM_IP	Unique	**Last Branch Record 0 From IP. (R/W)** One of four pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last four branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.10, “Last Branch, Interrupt, and Exception Recording (Pentium M Processors).”
41H	65	MSR_ LASTBRANCH_1_F ROM_IP	Unique	**Last Branch Record 1 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_F ROM_IP	Unique	**Last Branch Record 2 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_F ROM_IP	Unique	**Last Branch Record 3 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
60H	96	MSR_ LASTBRANCH_0_ TO_LIP	Unique	**Last Branch Record 0 To IP. (R/W)** One of four pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last four branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_ LASTBRANCH_1_ TO_LIP	Unique	**Last Branch Record 1 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
62H	98	MSR_ LASTBRANCH_2_ TO_LIP	Unique	**Last Branch Record 2 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
63H	99	MSR_ LASTBRANCH_3_ TO_LIP	Unique	**Last Branch Record 3 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
79H	121	IA32_BIOS_ UPDT_TRIG	Unique	**BIOS Update Trigger Register. (W)** See Table 34-2.
8BH	139	IA32_BIOS_ SIGN_ID	Unique	**BIOS Update Signature ID. (RO)** See Table 34-2.
A0H	160	MSR_SMRR_PHYS BASE	Unique	**System Management Mode Base Address register. (WO in SMM)** Model-specific implementation of SMRR-like interface, read visible and write only in SMM.
		11:0		Reserved.
		31:12		PhysBase. SMRR physical Base Address.
		63:32		Reserved.
A1H	161	MSR_SMRR_PHYS MASK	Unique	**System Management Mode Physical Address Mask register. (WO in SMM)** Model-specific implementation of SMRR-like interface, read visible and write only in SMM..
		10:0		Reserved.
		11		Valid. Physical address base and range mask are valid.
		31:12		PhysMask. SMRR physical address range mask.
		63:32		Reserved.
C1H	193	IA32_PMC0	Unique	**Performance counter register.** See Table 34-2.
C2H	194	IA32_PMC1	Unique	**Performance counter register.** See Table 34-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO).** This field indicates the intended scaleable bus clock speed for processors based on Intel Core microarchitecture:

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
				266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B.
		63:3		Reserved.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO).** This field indicates the intended scaleable bus clock speed for processors based on Enhanced Intel Core microarchitecture:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) ▪ 010B: 200 MHz (FSB 800) ▪ 000B: 267 MHz (FSB 1067) ▪ 100B: 333 MHz (FSB 1333) ▪ 110B: 400 MHz (FSB 1600)
				133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
				266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 110B. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 111B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW)** see Table 34-2
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW)** See Table 34-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 34-2.
		11	Unique	**SMRR Capability Using MSR 0A0H and 0A1H. (R)**
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.
		23		**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
174H	372	IA32_SYSENTER_CS	Unique	See Table 34-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 34-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 34-2.
179H	377	IA32_MCG_CAP	Unique	See Table 34-2.
17AH	378	IA32_MCG_ STATUS	Unique	
		0		**RIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP.** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Unique	See Table 34-2.
187H	391	IA32_ PERFEVTSEL1	Unique	See Table 34-2.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
198H	408	IA32_PERF_STATUS	Shared	See Table 34-2.
198H	408	MSR_PERF_STATUS	Shared	
		15:0		Current Performance State Value.
		30:16		Reserved.
		31		XE Operation (R/O). If set, XE operation is enabled. Default is cleared.
		39:32		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		45		Reserved.
		46		Non-Integer Bus Ratio (R/O) Indicates non-integer bus ratio is enabled. Applies processors based on Enhanced Intel Core microarchitecture.
		63:47		Reserved.
199H	409	IA32_PERF_CTL	Unique	See Table 34-2.
19AH	410	IA32_CLOCK_MODULATION	Unique	**Clock Modulation. (R/W)** See Table 34-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_INTERRUPT	Unique	**Thermal Interrupt Control. (R/W)** See Table 34-2.
19CH	412	IA32_THERM_STATUS	Unique	**Thermal Monitor Status. (R/W)** See Table 34-2.
19DH	413	MSR_THERM2_CTL	Unique	

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		15:0		Reserved.
		16		**TM_SELECT. (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:16		Reserved.
1A0	416	IA32_MISC_ ENABLE		**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0		**Fast-Strings Enable.** See Table 34-2.
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable. (R/W)** See Table 34-2.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available. (R)** See Table 34-2.
		8		Reserved.
		9		**Hardware Prefetcher Disable. (R/W)** When set, disables the hardware prefetcher operation on streams of data. When clear (default), enables the prefetch queue. Disabling of the hardware prefetcher may impact processor performance.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		10	Shared	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable. (RO)** See Table 34-2.
		12	Shared	**Precise Event Based Sampling Unavailable. (RO)** See Table 34-2.
		13	Shared	**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable. (R/W)** See Table 34-2.
		18	Shared	ENABLE MONITOR FSM. (R/W) See Table 34-2.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		19	Shared	**Adjacent Cache Line Prefetch Disable. (R/W)** When set to 1, the processor fetches the cache line that contains data currently required by the processor. When set to 0, the processor fetches cache lines that comprise a cache line pair (128 bytes). Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.
		20	Shared	**Enhanced Intel SpeedStep Technology Select Lock. (R/WO)** When set, this bit causes the following bits to become read-only: ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.
		22	Shared	**Limit CPUID Maxval. (R/W)** See Table 34-2.
		23	Shared	**xTPR Message Disable. (R/W)** See Table 34-2.
		33:24		Reserved.
		34	Unique	**XD Bit Disable. (R/W)** See Table 34-2.
		36:35		Reserved.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		37	Unique	**DCU Prefetcher Disable. (R/W)** When set to 1, The DCU L1 data cache prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The DCU prefetcher is an L1 data cache prefetcher. When the DCU prefetcher detects multiple loads from the same line done within a time limit, the DCU prefetcher assumes the next line will be required. The next line is prefetched in to the L1 data cache from memory or L2.
		38	Shared	**IDA Disable. (R/W)** When set to 1 on processors that support IDA, the Intel Dynamic Acceleration feature (IDA) is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of IDA is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of IDA. If power-on default value is 1, IDA is available in the processor. If power-on default value is 0, IDA is not available.
		39	Unique	**IP Prefetcher Disable. (R/W)** When set to 1, The IP prefetcher is disabled. The default value after reset is 0. BIOS may write '1' to disable this feature. The IP prefetcher is an L1 data cache prefetcher. The IP prefetcher looks for sequential load history to determine whether to prefetch the next expected data into the L1 cache from memory or L2.
		63:40		Reserved.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1C9H	457	MSR_ LASTBRANCH_ TOS	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control. (R/W)** see Table 34-2
1DDH	477	MSR_LER_FROM_ LIP	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_ LIP	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYS BASE0	Unique	See Table 34-2.
201H	513	IA32_MTRR_PHYS MASK0	Unique	See Table 34-2.
202H	514	IA32_MTRR_PHYS BASE1	Unique	See Table 34-2.
203H	515	IA32_MTRR_PHYS MASK1	Unique	See Table 34-2.
204H	516	IA32_MTRR_PHYS BASE2	Unique	See Table 34-2.
205H	517	IA32_MTRR_PHYS MASK2	Unique	See Table 34-2.
206H	518	IA32_MTRR_PHYS BASE3	Unique	See Table 34-2.
207H	519	IA32_MTRR_PHYS MASK3	Unique	See Table 34-2.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
208H	520	IA32_MTRR_PHYS BASE4	Unique	See Table 34-2.
209H	521	IA32_MTRR_PHYS MASK4	Unique	See Table 34-2.
20AH	522	IA32_MTRR_PHYS BASE5	Unique	See Table 34-2.
20BH	523	IA32_MTRR_PHYS MASK5	Unique	See Table 34-2.
20CH	524	IA32_MTRR_PHYS BASE6	Unique	See Table 34-2.
20DH	525	IA32_MTRR_PHYS MASK6	Unique	See Table 34-2.
20EH	526	IA32_MTRR_PHYS BASE7	Unique	See Table 34-2.
20FH	527	IA32_MTRR_PHYS MASK7	Unique	See Table 34-2.
250H	592	IA32_MTRR_FIX6 4K_00000	Unique	See Table 34-2.
258H	600	IA32_MTRR_FIX1 6K_80000	Unique	See Table 34-2.
259H	601	IA32_MTRR_FIX1 6K_A0000	Unique	See Table 34-2.
268H	616	IA32_MTRR_FIX4 K_C0000	Unique	See Table 34-2.
269H	617	IA32_MTRR_FIX4 K_C8000	Unique	See Table 34-2.
26AH	618	IA32_MTRR_FIX4 K_D0000	Unique	See Table 34-2.
26BH	619	IA32_MTRR_FIX4 K_D8000	Unique	See Table 34-2.
26CH	620	IA32_MTRR_FIX4 K_E0000	Unique	See Table 34-2.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
26DH	621	IA32_MTRR_FIX4K_E8000	Unique	See Table 34-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Unique	See Table 34-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Unique	See Table 34-2.
277H	631	IA32_PAT	Unique	See Table 34-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Unique	**Default Memory Types. (R/W)** See Table 34-2.
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0. (R/W)** See Table 34-2.
309H	777	MSR_PERF_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0. (R/W)**
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1. (R/W)** See Table 34-2.
30AH	778	MSR_PERF_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1. (R/W)**
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2. (R/W)** See Table 34-2.
30BH	779	MSR_PERF_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2. (R/W)**
345H	837	IA32_PERF_CAPABILITIES	Unique	See Table 34-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
345H	837	MSR_PERF_CAPABILITIES	Unique	RO. This applies to processors that do not support architectural perfmon version 2.
		5:0		LBR Format. See Table 34-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 34-2.
		63:8		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register. (R/W)** See Table 34-2.

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
38DH	909	MSR_PERF_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register. (R/W)**
38EH	910	IA32_PERF_GLOBAL_STAUS	Unique	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
38EH	910	MSR_PERF_GLOBAL_STAUS	Unique	See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	MSR_PERF_GLOBAL_CTRL	Unique	See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Unique	See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 34-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_ STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_ STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
412H	1042	MSR_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC3_MISC	Unique	
414H	1044	MSR_MC5_CTL	Unique	
415H	1045	MSR_MC5_ STATUS	Unique	
416H	1046	MSR_MC5_ADDR	Unique	
417H	1047	MSR_MC5_MISC	Unique	
419H	1045	MSR_MC6_ STATUS	Unique	Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 15.3.2.2, "IA32_MCi_STATUS MSRS." and Chapter 23.
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBA SED_CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Table 34-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCB ASED_CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_ CTLS	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** See Table 34-2. See Appendix A.4, "VM-Exit Controls."

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
484H	1156	IA32_VMX_ ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** See Table 34-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_ FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_ FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FI XED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FI XED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_ VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Table 34-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCB ASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	**DS Save Area. (R/W).** See Table 34-2. See Section 18.9.4, "Debug Store (DS) Mechanism."
107CC H		MSR_EMON_L3_C TR_CTL0	Unique	**GBUSQ Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
107CDH		MSR_EMON_L3_CTR_CTL1	Unique	**GBUSQ Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CEH		MSR_EMON_L3_CTR_CTL2	Unique	**GSNPQ Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107CFH		MSR_EMON_L3_CTR_CTL3	Unique	**GSNPQ Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D0H		MSR_EMON_L3_CTR_CTL4	Unique	**FSB Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D1H		MSR_EMON_L3_CTR_CTL5	Unique	**FSB Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D2H		MSR_EMON_L3_CTR_CTL6	Unique	**FSB Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
107D3H		MSR_EMON_L3_CTR_CTL7	Unique	**FSB Event Control/Counter Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2

Table 34-3. MSRs in Processors Based on Intel Core Microarchitecture (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
107D8 H		MSR_EMON_L3 _GL_CTL	Unique	**L3/FSB Common Control Register. (R/W).** Apply to Intel Xeon processor 7400 series (processor signature 06_1D) only. See Section 17.2.2
C000_ 0080H		IA32_EFER	Unique	**Extended Feature Enables.** See Table 34-2.
C000_ 0081H		IA32_STAR	Unique	**System Call Target Address. (R/W).** See Table 34-2.
C000_ 0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address. (R/W).** See Table 34-2.
C000_ 0084H		IA32_FMASK	Unique	**System Call Flag Mask. (R/W).** See Table 34-2.
C000_ 0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS. (R/W).** See Table 34-2.
C000_ 0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS. (R/W).** See Table 34-2.
C000_ 0102H		IA32_KERNEL_GS BASE	Unique	**Swap Target of BASE Address of GS. (R/W).** See Table 34-2.

34.3 MSRS IN THE INTEL® ATOM™ PROCESSOR FAMILY

Table 34-4 lists model-specific registers (MSRs) for Intel Atom processor family, architectural MSR addresses are also included in Table 34-4. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1CH, see Table 34-1.

The column "Shared/Unique" applies to logical processors sharing the same core in processors based on the Intel Atom microarchitecture. "Unique" means each logical processor has a separate MSR, or a bit field in an MSR governs only a logical processor. "Shared" means the MSR or the bit field in an MSR address governs the operation of both logical processors in the same core.

Table 34-4. MSRs in Intel Atom Processor Family

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ ADDR	Shared	See Section 34.12, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_ TYPE	Shared	See Section 34.12, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_ FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination." andTable 34-2
10H	16	IA32_TIME_ STAMP_COUNTER	Shared	See Section 17.12, "Time-Stamp Counter," and see Table 34-2.
17H	23	IA32_PLATFORM_I D	Shared	**Platform ID. (R)** See Table 34-2.
17H	23	MSR_PLATFORM_I D	Shared	**Model Specific Platform ID. (R)**
		7:0		Reserved.
		12:8		**Maximum Qualified Ratio. (R)** The maximum allowed bus ratio.
		63:13		Reserved.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location," and Table 34-2.
2AH	42	MSR_EBL_CR_ POWERON	Shared	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		**Data Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		2		**Response Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		3		**AERR# Drive Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		4		**BERR# Enable for initiator bus requests. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		5		Reserved.
		6		Reserved.
		7		**BINIT# Driver Enable. (R/W)** 1 = Enabled; 0 = Disabled Always 0.
		8		Reserved.
		9		**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled
		10		**AERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		11		Reserved.
		12		**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0.
		13		**Reserved.**
		14		**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		**APIC Cluster ID. (R/O)** Always 00B.
		19: 18		Reserved.
		21: 20		**Symmetric Arbitration ID. (R/O)** Always 00B.
		26:22		**Integer Bus Frequency Ratio. (R/O)**

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_ CONTROL	Unique	**Control Features in Intel 64Processor. (R/W).** See Table 34-2.
40H	64	MSR_ LASTBRANCH_0_F ROM_IP	Unique	**Last Branch Record 0 From IP. (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last eight branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1_F ROM_IP	Unique	**Last Branch Record 1 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
42H	66	MSR_ LASTBRANCH_2_F ROM_IP	Unique	**Last Branch Record 2 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
43H	67	MSR_ LASTBRANCH_3_F ROM_IP	Unique	**Last Branch Record 3 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
44H	68	MSR_ LASTBRANCH_4_F ROM_IP	Unique	**Last Branch Record 4 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
45H	69	MSR_ LASTBRANCH_5_F ROM_IP	Unique	**Last Branch Record 5 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
46H	70	MSR_ LASTBRANCH_6_F ROM_IP	Unique	**Last Branch Record 6 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
47H	71	MSR_ LASTBRANCH_7_F ROM_IP	Unique	**Last Branch Record 7 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
60H	96	MSR_ LASTBRANCH_0_ TO_LIP	Unique	**Last Branch Record 0 To IP. (R/W)** One of eight pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last eight branches, exceptions, or interrupts taken by the processor.
61H	97	MSR_ LASTBRANCH_1_ TO_LIP	Unique	**Last Branch Record 1 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
62H	98	MSR_ LASTBRANCH_2_ TO_LIP	Unique	**Last Branch Record 2 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
63H	99	MSR_ LASTBRANCH_3_ TO_LIP	Unique	**Last Branch Record 3 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
64H	100	MSR_ LASTBRANCH_4_ TO_LIP	Unique	**Last Branch Record 4 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
65H	101	MSR_ LASTBRANCH_5_ TO_LIP	Unique	**Last Branch Record 5 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
66H	102	MSR_ LASTBRANCH_6_ TO_LIP	Unique	**Last Branch Record 6 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
67H	103	MSR_ LASTBRANCH_7_ TO_LIP	Unique	**Last Branch Record 7 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
79H	121	IA32_BIOS_ UPDT_TRIG	Unique	**BIOS Update Trigger Register. (W)** See Table 34-2.
8BH	139	IA32_BIOS_ SIGN_ID	Unique	**BIOS Update Signature ID. (RO)** See Table 34-2.
C1H	193	IA32_PMC0	Unique	**Performance counter register.** See Table 34-2.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	**Dec**			
C2H	194	IA32_PMC1	Unique	**Performance counter register.** See Table 34-2.
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed(RO).** This field indicates the intended scaleable bus clock speed for processors based on Intel Atom microarchitecture:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW)** See Table 34-2.
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW)** See Table 34-2.
FEH	254	IA32_MTRRCAP	Shared	**Memory Type Range Register. (R)** See Table 34-2.
11EH	281	MSR_BBL_CR_ CTL3	Shared	
		0		**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		23		**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 34-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 34-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 34-2.
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted
		1		**EIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP.** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Unique	See Table 34-2.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
187H	391	IA32_ PERFEVTSEL1	Unique	See Table 34-2.
198H	408	IA32_PERF_STAT US	Shared	See Table 34-2.
198H	408	MSR_PERF_STATU S	Shared	
		15:0		Current Performance State Value.
		39:16		Reserved.
		44:40		Maximum Bus Ratio (R/O) Indicates maximum bus ratio configured for the processor.
		63:45		Reserved.
199H	409	IA32_PERF_CTL	Unique	See Table 34-2.
19AH	410	IA32_CLOCK_ MODULATION	Unique	**Clock Modulation. (R/W)** See Table 34-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
19BH	411	IA32_THERM_ INTERRUPT	Unique	**Thermal Interrupt Control. (R/W)** See Table 34-2.
19CH	412	IA32_THERM_ STATUS	Unique	**Thermal Monitor Status. (R/W)** See Table 34-2.
19DH	413	MSR_THERM2_ CTL	Shared	
		15:0		Reserved.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		16		**TM_SELECT. (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 are enabled.
		63:17		Reserved.
1A0	416	IA32_MISC_ ENABLE	Unique	**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0		**Fast-Strings Enable.** See Table 34-2.
		2:1		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable. (R/W)** See Table 34-2.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available. (R)** See Table 34-2.
		8		Reserved.
		9		Reserved.
		10	Shared	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable. (RO)** See Table 34-2.
		12	Shared	**Precise Event Based Sampling Unavailable. (RO)** See Table 34-2.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		13	Shared	**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermally managed state. The BIOS must enable this feature if the TM2 feature flag (CPUID.1:ECX[8]) is set; if the TM2 feature flag is not set, this feature is not supported and BIOS must not alter the contents of the TM2 bit location. The processor is operating out of specification if both this bit and the TM1 bit are set to 0.
		15:14		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable. (R/W)** See Table 34-2.
		18	Shared	ENABLE MONITOR FSM. (R/W) See Table 34-2.
		19		Reserved.
		20	Shared	**Enhanced Intel SpeedStep Technology Select Lock. (R/WO)** When set, this bit causes the following bits to become read-only: ▪ Enhanced Intel SpeedStep Technology Select Lock (this bit), ▪ Enhanced Intel SpeedStep Technology Enable bit. The bit must be set before an Enhanced Intel SpeedStep Technology transition is requested. This bit is cleared on reset.
		21		Reserved.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		22	Unique	**Limit CPUID Maxval. (R/W)** See Table 34-2.
		23	Shared	**xTPR Message Disable. (R/W)** See Table 34-2.
		33:24		Reserved.
		34	Unique	**XD Bit Disable. (R/W)** See Table 34-2.
		63:35		Reserved.
1C9H	457	MSR_ LASTBRANCH_ TOS	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-2) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control. (R/W)** See Table 34-2.
1DDH	477	MSR_LER_FROM_ LIP	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_ LIP	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
200H	512	IA32_MTRR_PHYS BASE0	Shared	See Table 34-2.
201H	513	IA32_MTRR_PHYS MASK0	Shared	See Table 34-2.
202H	514	IA32_MTRR_PHYS BASE1	Shared	See Table 34-2.
203H	515	IA32_MTRR_PHYS MASK1	Shared	See Table 34-2.
204H	516	IA32_MTRR_PHYS BASE2	Shared	See Table 34-2.
205H	517	IA32_MTRR_PHYS MASK2	Shared	See Table 34-2.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
206H	518	IA32_MTRR_PHYS BASE3	Shared	See Table 34-2.
207H	519	IA32_MTRR_PHYS MASK3	Shared	See Table 34-2.
208H	520	IA32_MTRR_PHYS BASE4	Shared	See Table 34-2.
209H	521	IA32_MTRR_PHYS MASK4	Shared	See Table 34-2.
20AH	522	IA32_MTRR_PHYS BASE5	Shared	See Table 34-2.
20BH	523	IA32_MTRR_PHYS MASK5	Shared	See Table 34-2.
20CH	524	IA32_MTRR_PHYS BASE6	Shared	See Table 34-2.
20DH	525	IA32_MTRR_PHYS MASK6	Shared	See Table 34-2.
20EH	526	IA32_MTRR_PHYS BASE7	Shared	See Table 34-2.
20FH	527	IA32_MTRR_PHYS MASK7	Shared	See Table 34-2.
250H	592	IA32_MTRR_FIX6 4K_00000	Shared	See Table 34-2.
258H	600	IA32_MTRR_FIX1 6K_80000	Shared	See Table 34-2.
259H	601	IA32_MTRR_FIX1 6K_A0000	Shared	See Table 34-2.
268H	616	IA32_MTRR_FIX4 K_C0000	Shared	See Table 34-2.
269H	617	IA32_MTRR_FIX4 K_C8000	Shared	See Table 34-2.
26AH	618	IA32_MTRR_FIX4 K_D0000	Shared	See Table 34-2.
26BH	619	IA32_MTRR_FIX4 K_D8000	Shared	See Table 34-2.

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
26CH	620	IA32_MTRR_FIX4K_E0000	Shared	See Table 34-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Shared	See Table 34-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Shared	See Table 34-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Shared	See Table 34-2.
277H	631	IA32_PAT	Unique	See Table 34-2.
309H	777	IA32_FIXED_CTR0	Unique	**Fixed-Function Performance Counter Register 0. (R/W)** See Table 34-2.
30AH	778	IA32_FIXED_CTR1	Unique	**Fixed-Function Performance Counter Register 1. (R/W)** See Table 34-2.
30BH	779	IA32_FIXED_CTR2	Unique	**Fixed-Function Performance Counter Register 2. (R/W)** See Table 34-2.
345H	837	IA32_PERF_CAPABILITIES	Shared	See Table 34-2. See Section 17.4.1, "IA32_DEBUGCTL MSR."
38DH	909	IA32_FIXED_CTR_CTRL	Unique	**Fixed-Function-Counter Control Register. (R/W)** See Table 34-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Unique	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
38FH	911	IA32_PERF_GLOBAL_CTRL	Unique	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Unique	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
3F1H	1009	MSR_PEBS_ENABLE	Unique	See Table 34-2. See Section 18.4.4, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
400H	1024	IA32_MC0_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
402H	1026	IA32_MC0_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
408H	1032	IA32_MC2_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC3_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC3_ STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC3_ADDR	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC4_CTL	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC4_ STATUS	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
412H	1042	MSR_MC4_ADDR	Shared	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.1, “Basic VMX Information.”
481H	1153	IA32_VMX_PINBASED_CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Table 34-2. See Appendix A.3, “VM-Execution Controls.”
482H	1154	IA32_VMX_PROCBASED_CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, “VM-Execution Controls.”
483H	1155	IA32_VMX_EXIT_CTLS	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** See Table 34-2. See Appendix A.4, “VM-Exit Controls.”
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** See Table 34-2. See Appendix A.5, “VM-Entry Controls.”
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.6, “Miscellaneous Data.”
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”
487H	1159	IA32_VMX_CR0_FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.7, “VMX-Fixed Bits in CR0.”

Table 34-4. MSRs in Intel Atom Processor Family (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
488H	1160	IA32_VMX_CR4_FIXED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FIXED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_ VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Table 34-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Unique	**DS Save Area. (R/W).** See Table 34-2. See Section 18.9.4, "Debug Store (DS) Mechanism."
C000_ 0080H		IA32_EFER	Unique	**Extended Feature Enables.** See Table 34-2.
C000_ 0081H		IA32_STAR	Unique	**System Call Target Address. (R/W).** See Table 34-2.
C000_ 0082H		IA32_LSTAR	Unique	**IA-32e Mode System Call Target Address. (R/W).** See Table 34-2.
C000_ 0084H		IA32_FMASK	Unique	**System Call Flag Mask. (R/W).** See Table 34-2.
C000_ 0100H		IA32_FS_BASE	Unique	**Map of BASE Address of FS. (R/W).** See Table 34-2.
C000_ 0101H		IA32_GS_BASE	Unique	**Map of BASE Address of GS. (R/W).** See Table 34-2.
C000_ 0102H		IA32_KERNEL_GS BASE	Unique	**Swap Target of BASE Address of GS. (R/W).** See Table 34-2.

34.4 MSRS IN THE INTEL® MICROARCHITECTURE CODE NAME NEHALEM

Table 34-5 lists model-specific registers (MSRs) that are common for Intel® microarchitecture code name Nehalem. These include Intel Core i7 and i5 processor family. Architectural MSR addresses are also included in Table 34-5. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH, 06_1FH, 06_2EH, see Table 34-1. Additional MSRs specific to 06_1AH, 06_1EH, 06_1FH are listed in Table 34-6. Some MSRs listed in these tables are used by BIOS. More information about these MSR can be found at http://biosbits.org.

The column "Scope" represents the package/core/thread scope of individual bit field of an MSR. "Thread" means this bit field must be programmed on each logical processor independently. "Core" means the bit field must be programmed on each processor core independently, logical processors in the same core will be affected by change of this bit on the other logical processor in the same core. "Package" means the bit field must be programmed once for each physical package. Change of a bit filed with a package scope will affect all logical processors in that physical package.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 34.12, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 34.12, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 34-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.12, "Time-Stamp Counter," and see Table 34-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID. (R)** See Table 34-2.
17H	23	MSR_PLATFORM_ID	Package	**Model Specific Platform ID. (R)**
		49:0		Reserved.
		52:50		See Table 34-2.
		63:53		Reserved.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, “Local APIC Status and Location,” and Table 34-2.
34H	52	MSR_SMI_ COUNT	Thread	**SMI Counter. (R/O).**
		31:0		**SMI Count. (R/O)** Count SMIs
		63:32		Reserved.
3AH	58	IA32_FEATURE_ CONTROL	Thread	**Control Features in Intel 64Processor. (R/W).** See Table 34-2.
79H	121	IA32_BIOS_ UPDT_TRIG	Core	**BIOS Update Trigger Register. (W)** See Table 34-2.
8BH	139	IA32_BIOS_ SIGN_ID	Thread	**BIOS Update Signature ID. (RO)** See Table 34-2.
C1H	193	IA32_PMC0	Thread	**Performance counter register.** See Table 34-2.
C2H	194	IA32_PMC1	Thread	**Performance counter register.** See Table 34-2.
C3H	195	IA32_PMC2	Thread	**Performance counter register.** See Table 34-2.
C4H	196	IA32_PMC3	Thread	**Performance counter register.** See Table 34-2.
CEH	206	MSR_PLATFORM_I NFO	Package	see http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio. (R/O)** The is the ratio of the frequency that invariant TSC runs at. The invariant TSC frequency can be computed by multiplying this ratio by 133.33 MHz.
		27:16		Reserved.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		28	Package	**Programmable Ratio Limit for Turbo Mode. (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDC-TDP Limit for Turbo Mode. (R/O)** When set to 1, indicates that TDC/TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDC and TDP Limits for Turbo mode are not programmable.
		39:30		Reserved.
		47:40	Package	**Maximum Efficiency Ratio. (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 133.33MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	**C-State Configuration Control** (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See http://biosbits.org.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		2:0		**Package C-State limit. (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0 (no package C-sate support) 001b: C1 (Behavior is the same as 000b) 010b: C3 011b: C6 100b: C7 101b and 110b: Reserved 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		**Reserved.**
		10		**I/O MWAIT Redirection Enable. (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions.
		14:11		**Reserved.**
		15		**CFG Lock. (R/WO)** When set, lock bits 15:0 of this register until next reset.
		23:16		**Reserved.**
		24		**Interrupt filtering enable. (R/W)** When set, processor cores in a deep C-State will wake only when the event message is destined for that core. When 0, all processor cores in a deep C-State will wake for an event message.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		25		**C3 state auto demotion enable. (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable. (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		63:27		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See `http://biosbits.org`.
		15:0		**LVL_2 Base Address. (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.
		18:16		**C-state Range. (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count. (RW)** See Table 34-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count. (RW)** See Table 34-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 34-2.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
174H	372	IA32_SYSENTER_CS	Thread	See Table 34-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 34-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 34-2.
179H	377	IA32_MCG_CAP	Thread	See Table 34-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.
		1		**EIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP.** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_PERFEVTSEL0	Thread	See Table 34-2.
187H	391	IA32_PERFEVTSEL1	Thread	See Table 34-2.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
188H	392	IA32_ PERFEVTSEL2	Thread	See Table 34-2.
189H	393	IA32_ PERFEVTSEL3	Thread	See Table 34-2.
198H	408	IA32_PERF_STATUS	Core	See Table 34-2.
		15:0		Current Performance State Value.
		63:16		Reserved.
199H	409	IA32_PERF_CTL	Thread	See Table 34-2.
19AH	410	IA32_CLOCK_ MODULATION	Thread	**Clock Modulation. (R/W)** See Table 34-2. IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		0		Reserved.
		3:1		On demand Clock Modulation Duty Cycle (R/W).
		4		On demand Clock Modulation Enable (R/W).
		63:5		Reserved.
19BH	411	IA32_THERM_ INTERRUPT	Core	**Thermal Interrupt Control. (R/W)** See Table 34-2.
19CH	412	IA32_THERM_ STATUS	Core	**Thermal Monitor Status. (R/W)** See Table 34-2.
1A0	416	IA32_MISC_ ENABLE		**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0	Thread	**Fast-Strings Enable.** See Table 34-2.
		2:1		Reserved.
		3	Thread	**Automatic Thermal Control Circuit Enable. (R/W)** See Table 34-2.
		6:4		Reserved.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		7	Thread	**Performance Monitoring Available. (R)** See Table 34-2.
		10:8		Reserved.
		11	Thread	**Branch Trace Storage Unavailable. (RO)** See Table 34-2.
		12	Thread	**Precise Event Based Sampling Unavailable. (RO)** See Table 34-2.
		15:13		Reserved.
		16	Package	**Enhanced Intel SpeedStep Technology Enable. (R/W)** See Table 34-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 34-2.
		21:19		Reserved.
		22	Thread	**Limit CPUID Maxval. (R/W)** See Table 34-2.
		23	Thread	**xTPR Message Disable. (R/W)** See Table 34-2.
		33:24		Reserved.
		34	Thread	**XD Bit Disable. (R/W)** See Table 34-2.
		37:35		Reserved.
		38	Package	**Turbo Mode Disable. (R/W)** When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A2H	418	MSR_TEMPERATURE_TARGET	Thread	
		15:0		Reserved.
		23:16		**Temperature Target. (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.
		63:24		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	Offcore Response Event Select Register (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		See `http://biosbits.org`.
		0	Package	EIST Hardware Coordination Disable (R/W). When 0, enables hardware coordination of EIST request from processor cores; When 1, disables hardware coordination of EIST requests.
		1	Thread	**Energy/Performance Bias Enable. (R/W)** This bit makes the IA32_ENERGY_PERF_BIAS register (MSR 1B0h) visible to software with Ring 0 privileges. This bit's status (1 or 0) is also reflected by CPUID.(EAX=06h):ECX[3].
		63:2		Reserved.
1ACH	428	MSR_TURBO_POWER_CURRENT_LIMIT		See `http://biosbits.org`.
		14:0	Package	TDP Limit (R/W) TDP limit in 1/8 Watt granularity.
		15	Package	TDP Limit Override Enable (R/W) A value = 0 indicates override is not active, and a value = 1 indicates active.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		30:16	Package	**TDC Limit (R/W)** TDC limit in 1/8 Amp granularity.
		31	Package	**TDC Limit Override Enable (R/W)** A value = 0 indicates override is not active, and a value = 1 indicates active.
		63:32		Reserved.
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode.** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C.** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C.** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C.** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C.** Maximum turbo ratio limit of 4 core active.
		63:32		Reserved.
1C8H	456	MSR_LBR_SELECT	Core	**Last Branch Record Filtering Select Register** (R/W) see Section 17.6.2, “Filtering of Last Branch Records.”
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control. (R/W)** See Table 34-2.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1DDH	477	MSR_LER_FROM_LIP	Thread	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 34-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 34-2.
1FCH	508	MSR_POWER_CTL	Core	Power Control Register. See http://biosbits.org.
		0		Reserved.
		1	Package	**C1E Enable. (R/W)** When set to '1', will enable the CPU to switch to the Minimum Enhanced Intel SpeedStep Technology operating point when all execution cores enter MWAIT (C1).
		63:2		Reserved.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 34-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 34-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 34-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 34-2.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
204H	516	IA32_MTRR_PHYS BASE2	Thread	See Table 34-2.
205H	517	IA32_MTRR_PHYS MASK2	Thread	See Table 34-2.
206H	518	IA32_MTRR_PHYS BASE3	Thread	See Table 34-2.
207H	519	IA32_MTRR_PHYS MASK3	Thread	See Table 34-2.
208H	520	IA32_MTRR_PHYS BASE4	Thread	See Table 34-2.
209H	521	IA32_MTRR_PHYS MASK4	Thread	See Table 34-2.
20AH	522	IA32_MTRR_PHYS BASE5	Thread	See Table 34-2.
20BH	523	IA32_MTRR_PHYS MASK5	Thread	See Table 34-2.
20CH	524	IA32_MTRR_PHYS BASE6	Thread	See Table 34-2.
20DH	525	IA32_MTRR_PHYS MASK6	Thread	See Table 34-2.
20EH	526	IA32_MTRR_PHYS BASE7	Thread	See Table 34-2.
20FH	527	IA32_MTRR_PHYS MASK7	Thread	See Table 34-2.
210H	528	IA32_MTRR_PHYS BASE8	Thread	See Table 34-2.
211H	529	IA32_MTRR_PHYS MASK8	Thread	See Table 34-2.
212H	530	IA32_MTRR_PHYS BASE9	Thread	See Table 34-2.
213H	531	IA32_MTRR_PHYS MASK9	Thread	See Table 34-2.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 34-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 34-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 34-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 34-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 34-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 34-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 34-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 34-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 34-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 34-2.
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 34-2.
277H	631	IA32_PAT	Thread	See Table 34-2.
280H	640	IA32_MC0_CTL2	Package	See Table 34-2.
281H	641	IA32_MC1_CTL2	Package	See Table 34-2.
282H	642	IA32_MC2_CTL2	Core	See Table 34-2.
283H	643	IA32_MC3_CTL2	Core	See Table 34-2.
284H	644	IA32_MC4_CTL2	Core	See Table 34-2.
285H	645	IA32_MC5_CTL2	Core	See Table 34-2.
286H	646	IA32_MC6_CTL2	Package	See Table 34-2.
287H	647	IA32_MC7_CTL2	Package	See Table 34-2.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
288H	648	IA32_MC8_CTL2	Package	See Table 34-2.
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	**Default Memory Types. (R/W)** See Table 34-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0. (R/W)** See Table 34-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1. (R/W)** See Table 34-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2. (R/W)** See Table 34-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 34-2. See Section 17.4.1, “IA32_DEBUGCTL MSR.”
		5:0		LBR Format. See Table 34-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 34-2.
		11:8		PEBS_REC_FORMAT. See Table 34-2.
		12		SMM_FREEZE. See Table 34-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register. (R/W)** See Table 34-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Thread	See Table 34-2. See Section 18.4.2, “Global Counter Control Facilities.”
38EH	910	MSR_PERF_GLOBAL_STAUS	Thread	**(RO)**
		61		**UNC_Ovf**. Uncore overflowed if 1.
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 34-2. See Section 18.4.2, “Global Counter Control Facilities.”
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 34-2. See Section 18.4.2, “Global Counter Control Facilities.”

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
390H	912	MSR_PERF_GLOBAL_OVF_CTRL	Thread	**(R/W)**
		61		**CLR_UNC_Ovf**. Set 1 to clear UNC_Ovf.
3F1H	1009	MSR_PEBS_ENABLE	Thread	See See Section 18.6.1.1, “Precise Event Based Sampling (PEBS).”
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	See See Section 18.6.1.2, “Load Latency Performance Monitoring Facility.”
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
400H	1024	IA32_MC0_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
401H	1025	IA32_MC0_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	MSR_MC0_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	MSR_MC1_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
40BH	1035	MSR_MC2_MISC	Core	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
40CH	1036	MSR_MC3_CTL	Core	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
40DH	1037	MSR_MC3_ STATUS	Core	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
40EH	1038	MSR_MC3_ADDR	Core	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40FH	1039	MSR_MC3_MISC	Core	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
411H	1041	MSR_MC4_ STATUS	Core	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
412H	1042	MSR_MC4_ADDR	Core	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC4_MISC	Core	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
414H	1044	MSR_MC5_CTL	Core	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
415H	1045	MSR_MC5_ STATUS	Core	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
416H	1046	MSR_MC5_ADDR	Core	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
417H	1047	MSR_MC5_MISC	Core	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
419H	1049	MSR_MC6_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	MSR_MC7_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
480H	1152	IA32_VMX_BASIC	Thread	**Reporting Register of Basic VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBA SED_CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Table 34-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCB ASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_ CTLS	Thread	**Capability Reporting Register of VM-exit Controls. (R/O)** See Table 34-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls. (R/O)** See Table 34-2. See Appendix A.5, "VM-Entry Controls."

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_ FIXED0	Thread	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_ FIXED1	Thread	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FI XED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
489H	1161	IA32_VMX_CR4_FI XED1	Thread	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_ VMCS_ENUM	Thread	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Table 34-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCB ASED_CTLS2	Thread	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
600H	1536	IA32_DS_AREA	Thread	**DS Save Area. (R/W).** See Table 34-2. See Section 18.9.4, "Debug Store (DS) Mechanism."

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP. (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the source instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, "LBR Stack."
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	**Last Branch Record 1 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	**Last Branch Record 2 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	**Last Branch Record 3 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	**Last Branch Record 4 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	**Last Branch Record 5 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	**Last Branch Record 6 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
687H	1671	MSR_LASTBRANCH_7_FROM_IP	Thread	**Last Branch Record 7 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_LASTBRANCH_8_FROM_IP	Thread	**Last Branch Record 8 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
689H	1673	MSR_ LASTBRANCH_9_F ROM_IP	Thread	**Last Branch Record 9 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_ LASTBRANCH_10_ FROM_IP	Thread	**Last Branch Record 10 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_ LASTBRANCH_11_ FROM_IP	Thread	**Last Branch Record 11 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_ LASTBRANCH_12_ FROM_IP	Thread	**Last Branch Record 12 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_ LASTBRANCH_13_ FROM_IP	Thread	**Last Branch Record 13 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_ FROM_IP	Thread	**Last Branch Record 14 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_ LASTBRANCH_15_ FROM_IP	Thread	**Last Branch Record 15 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_ LASTBRANCH_0_ TO_LIP	Thread	**Last Branch Record 0 To IP. (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.
6C1H	1729	MSR_ LASTBRANCH_1_ TO_LIP	Thread	**Last Branch Record 1 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6C2H	1730	MSR_LASTBRANCH_2_TO_LIP	Thread	**Last Branch Record 2 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C3H	1731	MSR_LASTBRANCH_3_TO_LIP	Thread	**Last Branch Record 3 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C4H	1732	MSR_LASTBRANCH_4_TO_LIP	Thread	**Last Branch Record 4 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C5H	1733	MSR_LASTBRANCH_5_TO_LIP	Thread	**Last Branch Record 5 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C6H	1734	MSR_LASTBRANCH_6_TO_LIP	Thread	**Last Branch Record 6 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C7H	1735	MSR_LASTBRANCH_7_TO_LIP	Thread	**Last Branch Record 7 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C8H	1736	MSR_LASTBRANCH_8_TO_LIP	Thread	**Last Branch Record 8 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C9H	1737	MSR_LASTBRANCH_9_TO_LIP	Thread	**Last Branch Record 9 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CAH	1738	MSR_LASTBRANCH_10_TO_LIP	Thread	**Last Branch Record 10 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CBH	1739	MSR_LASTBRANCH_11_TO_LIP	Thread	**Last Branch Record 11 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CCH	1740	MSR_ LASTBRANCH_12_ TO_LIP	Thread	**Last Branch Record 12 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CDH	1741	MSR_ LASTBRANCH_13_ TO_LIP	Thread	**Last Branch Record 13 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CEH	1742	MSR_ LASTBRANCH_14_ TO_LIP	Thread	**Last Branch Record 14 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CFH	1743	MSR_ LASTBRANCH_15_ TO_LIP	Thread	**Last Branch Record 15 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
802H	2050	IA32_X2APIC_API CID	Thread	x2APIC ID register (R/O) See x2APIC Specification.
803H	2051	IA32_X2APIC_VER SION	Thread	x2APIC Version register (R/O)
808H	2056	IA32_X2APIC_TPR	Thread	x2APIC Task Priority register (R/W)
80AH	2058	IA32_X2APIC_PPR	Thread	x2APIC Processor Priority register (R/O)
80BH	2059	IA32_X2APIC_EOI	Thread	x2APIC EOI register (W/O)
80DH	2061	IA32_X2APIC_LDR	Thread	x2APIC Logical Destination register (R/O)
80FH	2063	IA32_X2APIC_SIV R	Thread	x2APIC Spurious Interrupt Vector register (R/W)
810H	2064	IA32_X2APIC_ISR 0	Thread	x2APIC In-Service register bits [31:0] (R/O)
811H	2065	IA32_X2APIC_ISR 1	Thread	x2APIC In-Service register bits [63:32] (R/O)
812H	2066	IA32_X2APIC_ISR 2	Thread	x2APIC In-Service register bits [95:64] (R/O)
813H	2067	IA32_X2APIC_ISR 3	Thread	x2APIC In-Service register bits [127:96] (R/O)

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
814H	2068	IA32_X2APIC_ISR4	Thread	x2APIC In-Service register bits [159:128] (R/O)
815H	2069	IA32_X2APIC_ISR5	Thread	x2APIC In-Service register bits [191:160] (R/O)
816H	2070	IA32_X2APIC_ISR6	Thread	x2APIC In-Service register bits [223:192] (R/O)
817H	2071	IA32_X2APIC_ISR7	Thread	x2APIC In-Service register bits [255:224] (R/O)
818H	2072	IA32_X2APIC_TMR0	Thread	x2APIC Trigger Mode register bits [31:0] (R/O)
819H	2073	IA32_X2APIC_TMR1	Thread	x2APIC Trigger Mode register bits [63:32] (R/O)
81AH	2074	IA32_X2APIC_TMR2	Thread	x2APIC Trigger Mode register bits [95:64] (R/O)
81BH	2075	IA32_X2APIC_TMR3	Thread	x2APIC Trigger Mode register bits [127:96] (R/O)
81CH	2076	IA32_X2APIC_TMR4	Thread	x2APIC Trigger Mode register bits [159:128] (R/O)
81DH	2077	IA32_X2APIC_TMR5	Thread	x2APIC Trigger Mode register bits [191:160] (R/O)
81EH	2078	IA32_X2APIC_TMR6	Thread	x2APIC Trigger Mode register bits [223:192] (R/O)
81FH	2079	IA32_X2APIC_TMR7	Thread	x2APIC Trigger Mode register bits [255:224] (R/O)
820H	2080	IA32_X2APIC_IRR0	Thread	x2APIC Interrupt Request register bits [31:0] (R/O)
821H	2081	IA32_X2APIC_IRR1	Thread	x2APIC Interrupt Request register bits [63:32] (R/O)
822H	2082	IA32_X2APIC_IRR2	Thread	x2APIC Interrupt Request register bits [95:64] (R/O)
823H	2083	IA32_X2APIC_IRR3	Thread	x2APIC Interrupt Request register bits [127:96] (R/O)

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
824H	2084	IA32_X2APIC_IRR4	Thread	x2APIC Interrupt Request register bits [159:128] (R/O)
825H	2085	IA32_X2APIC_IRR5	Thread	x2APIC Interrupt Request register bits [191:160] (R/O)
826H	2086	IA32_X2APIC_IRR6	Thread	x2APIC Interrupt Request register bits [223:192] (R/O)
827H	2087	IA32_X2APIC_IRR7	Thread	x2APIC Interrupt Request register bits [255:224] (R/O)
828H	2088	IA32_X2APIC_ESR	Thread	x2APIC Error Status register (R/W)
82FH	2095	IA32_X2APIC_LVT_CMCI	Thread	x2APIC LVT Corrected Machine Check Interrupt register (R/W)
830H	2096	IA32_X2APIC_ICR	Thread	x2APIC Interrupt Command register (R/W)
832H	2098	IA32_X2APIC_LVT_TIMER	Thread	x2APIC LVT Timer Interrupt register (R/W)
833H	2099	IA32_X2APIC_LVT_THERMAL	Thread	x2APIC LVT Thermal Sensor Interrupt register (R/W)
834H	2100	IA32_X2APIC_LVT_PMI	Thread	x2APIC LVT Performance Monitor register (R/W)
835H	2101	IA32_X2APIC_LVT_LINT0	Thread	x2APIC LVT LINT0 register (R/W)
836H	2102	IA32_X2APIC_LVT_LINT1	Thread	x2APIC LVT LINT1 register (R/W)
837H	2103	IA32_X2APIC_LVT_ERROR	Thread	x2APIC LVT Error register (R/W)
838H	2104	IA32_X2APIC_INIT_COUNT	Thread	x2APIC Initial Count register (R/W)
839H	2105	IA32_X2APIC_CUR_COUNT	Thread	x2APIC Current Count register (R/O)
83EH	2110	IA32_X2APIC_DIV_CONF	Thread	x2APIC Divide Configuration register (R/W)
83FH	2111	IA32_X2APIC_SELF_IPI	Thread	x2APIC Self IPI register (W/O)

Table 34-5. MSRs in Processors Based on Intel Microarchitecture Code Name Nehalem (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C000_ 0080H		IA32_EFER	Thread	**Extended Feature Enables.** See Table 34-2.
C000_ 0081H		IA32_STAR	Thread	**System Call Target Address. (R/W).** See Table 34-2.
C000_ 0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address. (R/W).** See Table 34-2.
C000_ 0084H		IA32_FMASK	Thread	**System Call Flag Mask. (R/W).** See Table 34-2.
C000_ 0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS. (R/W).** See Table 34-2.
C000_ 0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS. (R/W).** See Table 34-2.
C000_ 0102H		IA32_KERNEL_GS BASE	Thread	**Swap Target of BASE Address of GS. (R/W).** See Table 34-2.
C000_ 0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature. (R/W).** See Table 34-2 and Section 17.12.2, "IA32_TSC_AUX Register and RDTSCP Support."

34.4.1 Additional MSRs in the Intel® Xeon® Processor 5500 and 3400 Series

Intel Xeon Processor 5500 and 3400 series support additional model-specific registers listed in Table 34-6. These MSRs also apply to Intel Core i7 and i5 processor family CPUID signature with DisplayFamily_DisplayModel of 06_1AH, 06_1EH and 06_1FH, see Table 34-1.

Table 34-6. Additional MSRs in Intel Xeon Processor 5500 and 3400 Series

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Actual maximum turbo frequency is multiplied by 133.33MHz. (not available to model 06_2EH)
		7:0		**Maximum Turbo Ratio Limit 1C. (R/O)** maximum Turbo mode ratio limit with 1 core active.
		15:8		**Maximum Turbo Ratio Limit 2C. (R/O)** maximum Turbo mode ratio limit with 2cores active.
		23:16		**Maximum Turbo Ratio Limit 3C. (R/O)** maximum Turbo mode ratio limit with 3cores active.
		31:24		**Maximum Turbo Ratio Limit 4C. (R/O)** maximum Turbo mode ratio limit with 4 cores active.
		63:32		Reserved.
301H	769	MSR_GQ_SNOOP_MESF	Package	
		0		From M to S (R/W).
		1		From E to S (R/W).
		2		From S to S (R/W).
		3		From F to S (R/W).
		4		From M to I (R/W).
		5		From E to I (R/W).
		6		From S to I (R/W).
		7		From F to I (R/W).
		63:8		Reserved.
391H	913	MSR_UNCORE_PERF_GLOBAL_CTRL	Package	See Section 18.6.2.1, "Uncore Performance Monitoring Management Facility."
392H	914	MSR_UNCORE_PERF_GLOBAL_STATUS	Package	See Section 18.6.2.1, "Uncore Performance Monitoring Management Facility."

Table 34-6. Additional MSRs in Intel Xeon Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
393H	915	MSR_UNCORE_PERF_GLOBAL_OVF_CTRL	Package	See Section 18.6.2.1, "Uncore Performance Monitoring Management Facility."
394H	916	MSR_UNCORE_FIXED_CTR0	Package	See Section 18.6.2.1, "Uncore Performance Monitoring Management Facility."
395H	917	MSR_UNCORE_FIXED_CTR_CTRL	Package	See Section 18.6.2.1, "Uncore Performance Monitoring Management Facility."
396H	918	MSR_UNCORE_ADDR_OPCODE_MATCH	Package	See Section 18.6.2.3, "Uncore Address/Opcode Match MSR."
3B0H	960	MSR_UNCORE_PMC0	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B1H	961	MSR_UNCORE_PMC1	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B2H	962	MSR_UNCORE_PMC2	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B3H	963	MSR_UNCORE_PMC3	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B4H	964	MSR_UNCORE_PMC4	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B6H	966	MSR_UNCORE_PMC6	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3B7H	967	MSR_UNCORE_PMC7	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C0H	944	MSR_UNCORE_PERFEVTSEL0	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C1H	945	MSR_UNCORE_PERFEVTSEL1	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C2H	946	MSR_UNCORE_PERFEVTSEL2	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C3H	947	MSR_UNCORE_PERFEVTSEL3	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."

Table 34-6. Additional MSRs in Intel Xeon Processor 5500 and 3400 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3C4H	948	MSR_UNCORE_PERFEVTSEL4	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C5H	949	MSR_UNCORE_PERFEVTSEL5	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C6H	950	MSR_UNCORE_PERFEVTSEL6	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."
3C7H	951	MSR_UNCORE_PERFEVTSEL7	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."

34.4.2 Additional MSRs in the Intel® Xeon® Processor 7500 Series

Intel Xeon Processor 7500 series support MSRs listed in Table 34-5 (except MSR address 1ADH) and additional model-specific registers listed in Table 34-7.

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved. Attempt to read/write will cause #UD
289H	649	IA32_MC9_CTL2	Package	See Table 34-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 34-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 34-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 34-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 34-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 34-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 34-2.
290H	656	IA32_MC16_CTL2	Package	See Table 34-2.
291H	657	IA32_MC17_CTL2	Package	See Table 34-2.
292H	658	IA32_MC18_CTL2	Package	See Table 34-2.
293H	659	IA32_MC19_CTL2	Package	See Table 34-2.
294H	660	IA32_MC20_CTL2	Package	See Table 34-2.

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
295H	661	IA32_MC21_CTL2	Package	See Table 34-2.
394H	816	MSR_W_PMON_FIXED_CTR	Package	Uncore W-box perfmon fixed counter
395H	817	MSR_W_PMON_FIXED_CTR_CTL	Package	Uncore U-box perfmon fixed counter control MSR
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
425H	1061	MSR_MC9_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
426H	1062	MSR_MC9_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
427H	1063	MSR_MC9_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
429H	1065	MSR_MC10_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
42AH	1066	MSR_MC10_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
42BH	1067	MSR_MC10_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
42DH	1069	MSR_MC11_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
42EH	1070	MSR_MC11_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
42FH	1071	MSR_MC11_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
431H	1073	MSR_MC12_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
432H	1074	MSR_MC12_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
433H	1075	MSR_MC12_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
435H	1077	MSR_MC13_STATUS	Package	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS,” and Chapter 16.
436H	1078	MSR_MC13_ADDR	Package	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.”
437H	1079	MSR_MC13_MISC	Package	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.”
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
439H	1081	MSR_MC14_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	MSR_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	MSR_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	MSR_MC15_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	MSR_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	MSR_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
441H	1089	MSR_MC16_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	MSR_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	MSR_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	MSR_MC17_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
446H	1094	MSR_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	MSR_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	MSR_MC18_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44AH	1098	MSR_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	MSR_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	MSR_MC19_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44EH	1102	MSR_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	MSR_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
450H	1104	MSR_MC20_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
451H	1105	MSR_MC20_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
452H	1106	MSR_MC20_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
453H	1107	MSR_MC20_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
454H	1108	MSR_MC21_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
455H	1109	MSR_MC21_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
456H	1110	MSR_MC21_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
457H	1111	MSR_MC21_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
C00H	3072	MSR_U_PMON_GL OBAL_CTRL	Package	Uncore U-box perfmon global control MSR
C01H	3073	MSR_U_PMON_GL OBAL_STATUS	Package	Uncore U-box perfmon global status MSR
C02H	3074	MSR_U_PMON_GL OBAL_OVF_CTRL	Package	Uncore U-box perfmon global overflow control MSR
C10H	3088	MSR_U_PMON_EV NT_SEL	Package	Uncore U-box perfmon event select MSR
C11H	3089	MSR_U_PMON_CT R	Package	Uncore U-box perfmon counter MSR
C20H	3104	MSR_B0_PMON_B OX_CTRL	Package	Uncore B-box 0 perfmon local box control MSR
C21H	3105	MSR_B0_PMON_B OX_STATUS	Package	Uncore B-box 0 perfmon local box status MSR
C22H	3106	MSR_B0_PMON_B OX_OVF_CTRL	Package	Uncore B-box 0 perfmon local box overflow control MSR
C30H	3120	MSR_B0_PMON_E VNT_SEL0	Package	Uncore B-box 0 perfmon event select MSR
C31H	3121	MSR_B0_PMON_C TR0	Package	Uncore B-box 0 perfmon counter MSR
C32H	3122	MSR_B0_PMON_E VNT_SEL1	Package	Uncore B-box 0 perfmon event select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C33H	3123	MSR_B0_PMON_CTR1	Package	Uncore B-box 0 perfmon counter MSR
C34H	3124	MSR_B0_PMON_EVNT_SEL2	Package	Uncore B-box 0 perfmon event select MSR
C35H	3125	MSR_B0_PMON_CTR2	Package	Uncore B-box 0 perfmon counter MSR
C36H	3126	MSR_B0_PMON_EVNT_SEL3	Package	Uncore B-box 0 perfmon event select MSR
C37H	3127	MSR_B0_PMON_CTR3	Package	Uncore B-box 0 perfmon counter MSR
C40H	3136	MSR_S0_PMON_BOX_CTRL	Package	Uncore S-box 0 perfmon local box control MSR
C41H	3137	MSR_S0_PMON_BOX_STATUS	Package	Uncore S-box 0 perfmon local box status MSR
C42H	3138	MSR_S0_PMON_BOX_OVF_CTRL	Package	Uncore S-box 0 perfmon local box overflow control MSR
C50H	3152	MSR_S0_PMON_EVNT_SEL0	Package	Uncore S-box 0 perfmon event select MSR
C51H	3153	MSR_S0_PMON_CTR0	Package	Uncore S-box 0 perfmon counter MSR
C52H	3154	MSR_S0_PMON_EVNT_SEL1	Package	Uncore S-box 0 perfmon event select MSR
C53H	3155	MSR_S0_PMON_CTR1	Package	Uncore S-box 0 perfmon counter MSR
C54H	3156	MSR_S0_PMON_EVNT_SEL2	Package	Uncore S-box 0 perfmon event select MSR
C55H	3157	MSR_S0_PMON_CTR2	Package	Uncore S-box 0 perfmon counter MSR
C56H	3158	MSR_S0_PMON_EVNT_SEL3	Package	Uncore S-box 0 perfmon event select MSR
C57H	3159	MSR_S0_PMON_CTR3	Package	Uncore S-box 0 perfmon counter MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C60H	3168	MSR_B1_PMON_BOX_CTRL	Package	Uncore B-box 1 perfmon local box control MSR
C61H	3169	MSR_B1_PMON_BOX_STATUS	Package	Uncore B-box 1 perfmon local box status MSR
C62H	3170	MSR_B1_PMON_BOX_OVF_CTRL	Package	Uncore B-box 1 perfmon local box overflow control MSR
C70H	3184	MSR_B1_PMON_EVNT_SEL0	Package	Uncore B-box 1 perfmon event select MSR
C71H	3185	MSR_B1_PMON_CTR0	Package	Uncore B-box 1 perfmon counter MSR
C72H	3186	MSR_B1_PMON_EVNT_SEL1	Package	Uncore B-box 1 perfmon event select MSR
C73H	3187	MSR_B1_PMON_CTR1	Package	Uncore B-box 1 perfmon counter MSR
C74H	3188	MSR_B1_PMON_EVNT_SEL2	Package	Uncore B-box 1 perfmon event select MSR
C75H	3189	MSR_B1_PMON_CTR2	Package	Uncore B-box 1 perfmon counter MSR
C76H	3190	MSR_B1_PMON_EVNT_SEL3	Package	Uncore B-box 1vperfmon event select MSR
C77H	3191	MSR_B1_PMON_CTR3	Package	Uncore B-box 1 perfmon counter MSR
C80H	3120	MSR_W_PMON_BOX_CTRL	Package	Uncore W-box perfmon local box control MSR
C81H	3121	MSR_W_PMON_BOX_STATUS	Package	Uncore W-box perfmon local box status MSR
C82H	3122	MSR_W_PMON_BOX_OVF_CTRL	Package	Uncore W-box perfmon local box overflow control MSR
C90H	3136	MSR_W_PMON_EVNT_SEL0	Package	Uncore W-box perfmon event select MSR
C91H	3137	MSR_W_PMON_CTR0	Package	Uncore W-box perfmon counter MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C92H	3138	MSR_W_PMON_EVNT_SEL1	Package	Uncore W-box perfmon event select MSR
C93H	3139	MSR_W_PMON_CTR1	Package	Uncore W-box perfmon counter MSR
C94H	3140	MSR_W_PMON_EVNT_SEL2	Package	Uncore W-box perfmon event select MSR
C95H	3141	MSR_W_PMON_CTR2	Package	Uncore W-box perfmon counter MSR
C96H	3142	MSR_W_PMON_EVNT_SEL3	Package	Uncore W-box perfmon event select MSR
C97H	3143	MSR_W_PMON_CTR3	Package	Uncore W-box perfmon counter MSR
CA0H	3232	MSR_M0_PMON_BOX_CTRL	Package	Uncore M-box 0 perfmon local box control MSR
CA1H	3233	MSR_M0_PMON_BOX_STATUS	Package	Uncore M-box 0 perfmon local box status MSR
CA2H	3234	MSR_M0_PMON_BOX_OVF_CTRL	Package	Uncore M-box 0 perfmon local box overflow control MSR
CA4H	3236	MSR_M0_PMON_TIMESTAMP	Package	Uncore M-box 0 perfmon time stamp unit select MSR
CA5H	3237	MSR_M0_PMON_DSP	Package	Uncore M-box 0 perfmon DSP unit select MSR
CA6H	3238	MSR_M0_PMON_ISS	Package	Uncore M-box 0 perfmon ISS unit select MSR
CA7H	3239	MSR_M0_PMON_MAP	Package	Uncore M-box 0 perfmon MAP unit select MSR
CA8H	3240	MSR_M0_PMON_MSC_THR	Package	Uncore M-box 0 perfmon MIC THR select MSR
CA9H	3241	MSR_M0_PMON_PGT	Package	Uncore M-box 0 perfmon PGT unit select MSR
CAAH	3242	MSR_M0_PMON_PLD	Package	Uncore M-box 0 perfmon PLD unit select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CABH	3243	MSR_M0_PMON_ZDP	Package	Uncore M-box 0 perfmon ZDP unit select MSR
CB0H	3248	MSR_M0_PMON_EVNT_SEL0	Package	Uncore M-box 0 perfmon event select MSR
CB1H	3249	MSR_M0_PMON_CTR0	Package	Uncore M-box 0 perfmon counter MSR
CB2H	3250	MSR_M0_PMON_EVNT_SEL1	Package	Uncore M-box 0 perfmon event select MSR
CB3H	3251	MSR_M0_PMON_CTR1	Package	Uncore M-box 0 perfmon counter MSR
CB4H	3252	MSR_M0_PMON_EVNT_SEL2	Package	Uncore M-box 0 perfmon event select MSR
CB5H	3253	MSR_M0_PMON_CTR2	Package	Uncore M-box 0 perfmon counter MSR
CB6H	3254	MSR_M0_PMON_EVNT_SEL3	Package	Uncore M-box 0 perfmon event select MSR
CB7H	3255	MSR_M0_PMON_CTR3	Package	Uncore M-box 0 perfmon counter MSR
CB8H	3256	MSR_M0_PMON_EVNT_SEL4	Package	Uncore M-box 0 perfmon event select MSR
CB9H	3257	MSR_M0_PMON_CTR4	Package	Uncore M-box 0 perfmon counter MSR
CBAH	3258	MSR_M0_PMON_EVNT_SEL5	Package	Uncore M-box 0 perfmon event select MSR
CBBH	3259	MSR_M0_PMON_CTR5	Package	Uncore M-box 0 perfmon counter MSR
CC0H	3264	MSR_S1_PMON_BOX_CTRL	Package	Uncore S-box 1 perfmon local box control MSR
CC1H	3265	MSR_S1_PMON_BOX_STATUS	Package	Uncore S-box 1 perfmon local box status MSR
CC2H	3266	MSR_S1_PMON_BOX_OVF_CTRL	Package	Uncore S-box 1 perfmon local box overflow control MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CD0H	3280	MSR_S1_PMON_EVNT_SEL0	Package	Uncore S-box 1 perfmon event select MSR
CD1H	3281	MSR_S1_PMON_CTR0	Package	Uncore S-box 1 perfmon counter MSR
CD2H	3282	MSR_S1_PMON_EVNT_SEL1	Package	Uncore S-box 1 perfmon event select MSR
CD3H	3283	MSR_S1_PMON_CTR1	Package	Uncore S-box 1 perfmon counter MSR
CD4H	3284	MSR_S1_PMON_EVNT_SEL2	Package	Uncore S-box 1 perfmon event select MSR
CD5H	3285	MSR_S1_PMON_CTR2	Package	Uncore S-box 1 perfmon counter MSR
CD6H	3286	MSR_S1_PMON_EVNT_SEL3	Package	Uncore S-box 1 perfmon event select MSR
CD7H	3287	MSR_S1_PMON_CTR3	Package	Uncore S-box 1 perfmon counter MSR
CE0H	3296	MSR_M1_PMON_BOX_CTRL	Package	Uncore M-box 1 perfmon local box control MSR
CE1H	3297	MSR_M1_PMON_BOX_STATUS	Package	Uncore M-box 1 perfmon local box status MSR
CE2H	3298	MSR_M1_PMON_BOX_OVF_CTRL	Package	Uncore M-box 1 perfmon local box overflow control MSR
CE4H	3300	MSR_M1_PMON_TIMESTAMP	Package	Uncore M-box 1 perfmon time stamp unit select MSR
CE5H	3301	MSR_M1_PMON_DSP	Package	Uncore M-box 1 perfmon DSP unit select MSR
CE6H	3302	MSR_M1_PMON_ISS	Package	Uncore M-box 1 perfmon ISS unit select MSR
CE7H	3303	MSR_M1_PMON_MAP	Package	Uncore M-box 1 perfmon MAP unit select MSR
CE8H	3304	MSR_M1_PMON_MSC_THR	Package	Uncore M-box 1 perfmon MIC THR select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
CE9H	3305	MSR_M1_PMON_PGT	Package	Uncore M-box 1 perfmon PGT unit select MSR
CEAH	3306	MSR_M1_PMON_PLD	Package	Uncore M-box 1 perfmon PLD unit select MSR
CEBH	3307	MSR_M1_PMON_ZDP	Package	Uncore M-box 1 perfmon ZDP unit select MSR
CF0H	3312	MSR_M1_PMON_EVNT_SEL0	Package	Uncore M-box 1 perfmon event select MSR
CF1H	3313	MSR_M1_PMON_CTR0	Package	Uncore M-box 1 perfmon counter MSR
CF2H	3314	MSR_M1_PMON_EVNT_SEL1	Package	Uncore M-box 1 perfmon event select MSR
CF3H	3315	MSR_M1_PMON_CTR1	Package	Uncore M-box 1 perfmon counter MSR
CF4H	3316	MSR_M1_PMON_EVNT_SEL2	Package	Uncore M-box 1 perfmon event select MSR
CF5H	3317	MSR_M1_PMON_CTR2	Package	Uncore M-box 1 perfmon counter MSR
CF6H	3318	MSR_M1_PMON_EVNT_SEL3	Package	Uncore M-box 1 perfmon event select MSR
CF7H	3319	MSR_M1_PMON_CTR3	Package	Uncore M-box 1 perfmon counter MSR
CF8H	3320	MSR_M1_PMON_EVNT_SEL4	Package	Uncore M-box 1 perfmon event select MSR
CF9H	3321	MSR_M1_PMON_CTR4	Package	Uncore M-box 1 perfmon counter MSR
CFAH	3322	MSR_M1_PMON_EVNT_SEL5	Package	Uncore M-box 1 perfmon event select MSR
CFBH	3323	MSR_M1_PMON_CTR5	Package	Uncore M-box 1 perfmon counter MSR
D00H	3328	MSR_C0_PMON_BOX_CTRL	Package	Uncore C-box 0 perfmon local box control MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D01H	3329	MSR_C0_PMON_BOX_STATUS	Package	Uncore C-box 0 perfmon local box status MSR
D02H	3330	MSR_C0_PMON_BOX_OVF_CTRL	Package	Uncore C-box 0 perfmon local box overflow control MSR
D10H	3344	MSR_C0_PMON_EVNT_SEL0	Package	Uncore C-box 0 perfmon event select MSR
D11H	3345	MSR_C0_PMON_CTR0	Package	Uncore C-box 0 perfmon counter MSR
D12H	3346	MSR_C0_PMON_EVNT_SEL1	Package	Uncore C-box 0 perfmon event select MSR
D13H	3347	MSR_C0_PMON_CTR1	Package	Uncore C-box 0 perfmon counter MSR
D14H	3348	MSR_C0_PMON_EVNT_SEL2	Package	Uncore C-box 0 perfmon event select MSR
D15H	3349	MSR_C0_PMON_CTR2	Package	Uncore C-box 0 perfmon counter MSR
D16H	3350	MSR_C0_PMON_EVNT_SEL3	Package	Uncore C-box 0 perfmon event select MSR
D17H	3351	MSR_C0_PMON_CTR3	Package	Uncore C-box 0 perfmon counter MSR
D18H	3352	MSR_C0_PMON_EVNT_SEL4	Package	Uncore C-box 0 perfmon event select MSR
D19H	3353	MSR_C0_PMON_CTR4	Package	Uncore C-box 0 perfmon counter MSR
D1AH	3354	MSR_C0_PMON_EVNT_SEL5	Package	Uncore C-box 0 perfmon event select MSR
D1BH	3355	MSR_C0_PMON_CTR5	Package	Uncore C-box 0 perfmon counter MSR
D20H	3360	MSR_C4_PMON_BOX_CTRL	Package	Uncore C-box 4 perfmon local box control MSR
D21H	3361	MSR_C4_PMON_BOX_STATUS	Package	Uncore C-box 4 perfmon local box status MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D22H	3362	MSR_C4_PMON_BOX_OVF_CTRL	Package	Uncore C-box 4 perfmon local box overflow control MSR
D30H	3376	MSR_C4_PMON_EVNT_SEL0	Package	Uncore C-box 4 perfmon event select MSR
D31H	3377	MSR_C4_PMON_CTR0	Package	Uncore C-box 4 perfmon counter MSR
D32H	3378	MSR_C4_PMON_EVNT_SEL1	Package	Uncore C-box 4 perfmon event select MSR
D33H	3379	MSR_C4_PMON_CTR1	Package	Uncore C-box 4 perfmon counter MSR
D34H	3380	MSR_C4_PMON_EVNT_SEL2	Package	Uncore C-box 4 perfmon event select MSR
D35H	3381	MSR_C4_PMON_CTR2	Package	Uncore C-box 4 perfmon counter MSR
D36H	3382	MSR_C4_PMON_EVNT_SEL3	Package	Uncore C-box 4 perfmon event select MSR
D37H	3383	MSR_C4_PMON_CTR3	Package	Uncore C-box 4 perfmon counter MSR
D38H	3384	MSR_C4_PMON_EVNT_SEL4	Package	Uncore C-box 4 perfmon event select MSR
D39H	3385	MSR_C4_PMON_CTR4	Package	Uncore C-box 4 perfmon counter MSR
D3AH	3386	MSR_C4_PMON_EVNT_SEL5	Package	Uncore C-box 4 perfmon event select MSR
D3BH	3387	MSR_C4_PMON_CTR5	Package	Uncore C-box 4 perfmon counter MSR
D40H	3392	MSR_C2_PMON_BOX_CTRL	Package	Uncore C-box 2 perfmon local box control MSR
D41H	3393	MSR_C2_PMON_BOX_STATUS	Package	Uncore C-box 2 perfmon local box status MSR
D42H	3394	MSR_C2_PMON_BOX_OVF_CTRL	Package	Uncore C-box 2 perfmon local box overflow control MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D50H	3408	MSR_C2_PMON_EVNT_SEL0	Package	Uncore C-box 2 perfmon event select MSR
D51H	3409	MSR_C2_PMON_CTR0	Package	Uncore C-box 2 perfmon counter MSR
D52H	3410	MSR_C2_PMON_EVNT_SEL1	Package	Uncore C-box 2 perfmon event select MSR
D53H	3411	MSR_C2_PMON_CTR1	Package	Uncore C-box 2 perfmon counter MSR
D54H	3412	MSR_C2_PMON_EVNT_SEL2	Package	Uncore C-box 2 perfmon event select MSR
D55H	3413	MSR_C2_PMON_CTR2	Package	Uncore C-box 2 perfmon counter MSR
D56H	3414	MSR_C2_PMON_EVNT_SEL3	Package	Uncore C-box 2 perfmon event select MSR
D57H	3415	MSR_C2_PMON_CTR3	Package	Uncore C-box 2 perfmon counter MSR
D58H	3416	MSR_C2_PMON_EVNT_SEL4	Package	Uncore C-box 2 perfmon event select MSR
D59H	3417	MSR_C2_PMON_CTR4	Package	Uncore C-box 2 perfmon counter MSR
D5AH	3418	MSR_C2_PMON_EVNT_SEL5	Package	Uncore C-box 2 perfmon event select MSR
D5BH	3419	MSR_C2_PMON_CTR5	Package	Uncore C-box 2 perfmon counter MSR
D60H	3424	MSR_C6_PMON_BOX_CTRL	Package	Uncore C-box 6 perfmon local box control MSR
D61H	3425	MSR_C6_PMON_BOX_STATUS	Package	Uncore C-box 6 perfmon local box status MSR
D62H	3426	MSR_C6_PMON_BOX_OVF_CTRL	Package	Uncore C-box 6 perfmon local box overflow control MSR
D70H	3440	MSR_C6_PMON_EVNT_SEL0	Package	Uncore C-box 6 perfmon event select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D71H	3441	MSR_C6_PMON_CTR0	Package	Uncore C-box 6 perfmon counter MSR
D72H	3442	MSR_C6_PMON_EVNT_SEL1	Package	Uncore C-box 6 perfmon event select MSR
D73H	3443	MSR_C6_PMON_CTR1	Package	Uncore C-box 6 perfmon counter MSR
D74H	3444	MSR_C6_PMON_EVNT_SEL2	Package	Uncore C-box 6 perfmon event select MSR
D75H	3445	MSR_C6_PMON_CTR2	Package	Uncore C-box 6 perfmon counter MSR
D76H	3446	MSR_C6_PMON_EVNT_SEL3	Package	Uncore C-box 6 perfmon event select MSR
D77H	3447	MSR_C6_PMON_CTR3	Package	Uncore C-box 6 perfmon counter MSR
D78H	3448	MSR_C6_PMON_EVNT_SEL4	Package	Uncore C-box 6 perfmon event select MSR
D79H	3449	MSR_C6_PMON_CTR4	Package	Uncore C-box 6 perfmon counter MSR
D7AH	3450	MSR_C6_PMON_EVNT_SEL5	Package	Uncore C-box 6 perfmon event select MSR
D7BH	3451	MSR_C6_PMON_CTR5	Package	Uncore C-box 6 perfmon counter MSR
D80H	3456	MSR_C1_PMON_BOX_CTRL	Package	Uncore C-box 1 perfmon local box control MSR
D81H	3457	MSR_C1_PMON_BOX_STATUS	Package	Uncore C-box 1 perfmon local box status MSR
D82H	3458	MSR_C1_PMON_BOX_OVF_CTRL	Package	Uncore C-box 1 perfmon local box overflow control MSR
D90H	3472	MSR_C1_PMON_EVNT_SEL0	Package	Uncore C-box 1 perfmon event select MSR
D91H	3473	MSR_C1_PMON_CTR0	Package	Uncore C-box 1 perfmon counter MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
D92H	3474	MSR_C1_PMON_EVNT_SEL1	Package	Uncore C-box 1 perfmon event select MSR
D93H	3475	MSR_C1_PMON_CTR1	Package	Uncore C-box 1 perfmon counter MSR
D94H	3476	MSR_C1_PMON_EVNT_SEL2	Package	Uncore C-box 1 perfmon event select MSR
D95H	3477	MSR_C1_PMON_CTR2	Package	Uncore C-box 1 perfmon counter MSR
D96H	3478	MSR_C1_PMON_EVNT_SEL3	Package	Uncore C-box 1 perfmon event select MSR
D97H	3479	MSR_C1_PMON_CTR3	Package	Uncore C-box 1 perfmon counter MSR
D98H	3480	MSR_C1_PMON_EVNT_SEL4	Package	Uncore C-box 1 perfmon event select MSR
D99H	3481	MSR_C1_PMON_CTR4	Package	Uncore C-box 1 perfmon counter MSR
D9AH	3482	MSR_C1_PMON_EVNT_SEL5	Package	Uncore C-box 1 perfmon event select MSR
D9BH	3483	MSR_C1_PMON_CTR5	Package	Uncore C-box 1 perfmon counter MSR
DA0H	3488	MSR_C5_PMON_BOX_CTRL	Package	Uncore C-box 5 perfmon local box control MSR
DA1H	3489	MSR_C5_PMON_BOX_STATUS	Package	Uncore C-box 5 perfmon local box status MSR
DA2H	3490	MSR_C5_PMON_BOX_OVF_CTRL	Package	Uncore C-box 5 perfmon local box overflow control MSR
DB0H	3504	MSR_C5_PMON_EVNT_SEL0	Package	Uncore C-box 5 perfmon event select MSR
DB1H	3505	MSR_C5_PMON_CTR0	Package	Uncore C-box 5 perfmon counter MSR
DB2H	3506	MSR_C5_PMON_EVNT_SEL1	Package	Uncore C-box 5 perfmon event select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DB3H	3507	MSR_C5_PMON_CTR1	Package	Uncore C-box 5 perfmon counter MSR
DB4H	3508	MSR_C5_PMON_EVNT_SEL2	Package	Uncore C-box 5 perfmon event select MSR
DB5H	3509	MSR_C5_PMON_CTR2	Package	Uncore C-box 5 perfmon counter MSR
DB6H	3510	MSR_C5_PMON_EVNT_SEL3	Package	Uncore C-box 5 perfmon event select MSR
DB7H	3511	MSR_C5_PMON_CTR3	Package	Uncore C-box 5 perfmon counter MSR
DB8H	3512	MSR_C5_PMON_EVNT_SEL4	Package	Uncore C-box 5 perfmon event select MSR
DB9H	3513	MSR_C5_PMON_CTR4	Package	Uncore C-box 5 perfmon counter MSR
DBAH	3514	MSR_C5_PMON_EVNT_SEL5	Package	Uncore C-box 5 perfmon event select MSR
DBBH	3515	MSR_C5_PMON_CTR5	Package	Uncore C-box 5 perfmon counter MSR
DC0H	3520	MSR_C3_PMON_BOX_CTRL	Package	Uncore C-box 3 perfmon local box control MSR
DC1H	3521	MSR_C3_PMON_BOX_STATUS	Package	Uncore C-box 3 perfmon local box status MSR
DC2H	3522	MSR_C3_PMON_BOX_OVF_CTRL	Package	Uncore C-box 3 perfmon local box overflow control MSR
DD0H	3536	MSR_C3_PMON_EVNT_SEL0	Package	Uncore C-box 3 perfmon event select MSR
DD1H	3537	MSR_C3_PMON_CTR0	Package	Uncore C-box 3 perfmon counter MSR
DD2H	3538	MSR_C3_PMON_EVNT_SEL1	Package	Uncore C-box 3 perfmon event select MSR
DD3H	3539	MSR_C3_PMON_CTR1	Package	Uncore C-box 3 perfmon counter MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DD4H	3540	MSR_C3_PMON_EVNT_SEL2	Package	Uncore C-box 3 perfmon event select MSR
DD5H	3541	MSR_C3_PMON_CTR2	Package	Uncore C-box 3 perfmon counter MSR
DD6H	3542	MSR_C3_PMON_EVNT_SEL3	Package	Uncore C-box 3 perfmon event select MSR
DD7H	3543	MSR_C3_PMON_CTR3	Package	Uncore C-box 3 perfmon counter MSR
DD8H	3544	MSR_C3_PMON_EVNT_SEL4	Package	Uncore C-box 3 perfmon event select MSR
DD9H	3545	MSR_C3_PMON_CTR4	Package	Uncore C-box 3 perfmon counter MSR
DDAH	3546	MSR_C3_PMON_EVNT_SEL5	Package	Uncore C-box 3 perfmon event select MSR
DDBH	3547	MSR_C3_PMON_CTR5	Package	Uncore C-box 3 perfmon counter MSR
DE0H	3552	MSR_C7_PMON_BOX_CTRL	Package	Uncore C-box 7 perfmon local box control MSR
DE1H	3553	MSR_C7_PMON_BOX_STATUS	Package	Uncore C-box 7 perfmon local box status MSR
DE2H	3554	MSR_C7_PMON_BOX_OVF_CTRL	Package	Uncore C-box 7 perfmon local box overflow control MSR
DF0H	3568	MSR_C7_PMON_EVNT_SEL0	Package	Uncore C-box 7 perfmon event select MSR
DF1H	3569	MSR_C7_PMON_CTR0	Package	Uncore C-box 7 perfmon counter MSR
DF2H	3570	MSR_C7_PMON_EVNT_SEL1	Package	Uncore C-box 7 perfmon event select MSR
DF3H	3571	MSR_C7_PMON_CTR1	Package	Uncore C-box 7 perfmon counter MSR
DF4H	3572	MSR_C7_PMON_EVNT_SEL2	Package	Uncore C-box 7 perfmon event select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
DF5H	3573	MSR_C7_PMON_CTR2	Package	Uncore C-box 7 perfmon counter MSR
DF6H	3574	MSR_C7_PMON_EVNT_SEL3	Package	Uncore C-box 7 perfmon event select MSR
DF7H	3575	MSR_C7_PMON_CTR3	Package	Uncore C-box 7 perfmon counter MSR
DF8H	3576	MSR_C7_PMON_EVNT_SEL4	Package	Uncore C-box 7 perfmon event select MSR
DF9H	3577	MSR_C7_PMON_CTR4	Package	Uncore C-box 7 perfmon counter MSR
DFAH	3578	MSR_C7_PMON_EVNT_SEL5	Package	Uncore C-box 7 perfmon event select MSR
DFBH	3579	MSR_C7_PMON_CTR5	Package	Uncore C-box 7 perfmon counter MSR
E00H	3584	MSR_R0_PMON_BOX_CTRL	Package	Uncore R-box 0 perfmon local box control MSR
E01H	3585	MSR_R0_PMON_BOX_STATUS	Package	Uncore R-box 0 perfmon local box status MSR
E02H	3586	MSR_R0_PMON_BOX_OVF_CTRL	Package	Uncore R-box 0 perfmon local box overflow control MSR
E04H	3588	MSR_R0_PMON_IPERF0_P0	Package	Uncore R-box 0 perfmon IPERF0 unit Port 0 select MSR
E05H	3589	MSR_R0_PMON_IPERF0_P1	Package	Uncore R-box 0 perfmon IPERF0 unit Port 1 select MSR
E06H	3590	MSR_R0_PMON_IPERF0_P2	Package	Uncore R-box 0 perfmon IPERF0 unit Port 2 select MSR
E07H	3591	MSR_R0_PMON_IPERF0_P3	Package	Uncore R-box 0 perfmon IPERF0 unit Port 3 select MSR
E08H	3592	MSR_R0_PMON_IPERF0_P4	Package	Uncore R-box 0 perfmon IPERF0 unit Port 4 select MSR
E09H	3593	MSR_R0_PMON_IPERF0_P5	Package	Uncore R-box 0 perfmon IPERF0 unit Port 5 select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E0AH	3594	MSR_R0_PMON_IPERF0_P6	Package	Uncore R-box 0 perfmon IPERF0 unit Port 6 select MSR
E0BH	3595	MSR_R0_PMON_IPERF0_P7	Package	Uncore R-box 0 perfmon IPERF0 unit Port 7 select MSR
E0CH	3596	MSR_R0_PMON_QLX_P0	Package	Uncore R-box 0 perfmon QLX unit Port 0 select MSR
E0DH	3597	MSR_R0_PMON_QLX_P1	Package	Uncore R-box 0 perfmon QLX unit Port 1 select MSR
E0EH	3598	MSR_R0_PMON_QLX_P2	Package	Uncore R-box 0 perfmon QLX unit Port 2 select MSR
E0FH	3599	MSR_R0_PMON_QLX_P3	Package	Uncore R-box 0 perfmon QLX unit Port 3 select MSR
E10H	3600	MSR_R0_PMON_EVNT_SEL0	Package	Uncore R-box 0 perfmon event select MSR
E11H	3601	MSR_R0_PMON_CTR0	Package	Uncore R-box 0 perfmon counter MSR
E12H	3602	MSR_R0_PMON_EVNT_SEL1	Package	Uncore R-box 0 perfmon event select MSR
E13H	3603	MSR_R0_PMON_CTR1	Package	Uncore R-box 0 perfmon counter MSR
E14H	3604	MSR_R0_PMON_EVNT_SEL2	Package	Uncore R-box 0 perfmon event select MSR
E15H	3605	MSR_R0_PMON_CTR2	Package	Uncore R-box 0 perfmon counter MSR
E16H	3606	MSR_R0_PMON_EVNT_SEL3	Package	Uncore R-box 0 perfmon event select MSR
E17H	3607	MSR_R0_PMON_CTR3	Package	Uncore R-box 0 perfmon counter MSR
E18H	3608	MSR_R0_PMON_EVNT_SEL4	Package	Uncore R-box 0 perfmon event select MSR
E19H	3609	MSR_R0_PMON_CTR4	Package	Uncore R-box 0 perfmon counter MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E1AH	3610	MSR_R0_PMON_EVNT_SEL5	Package	Uncore R-box 0 perfmon event select MSR
E1BH	3611	MSR_R0_PMON_CTR5	Package	Uncore R-box 0 perfmon counter MSR
E1CH	3612	MSR_R0_PMON_EVNT_SEL6	Package	Uncore R-box 0 perfmon event select MSR
E1DH	3613	MSR_R0_PMON_CTR6	Package	Uncore R-box 0 perfmon counter MSR
E1EH	3614	MSR_R0_PMON_EVNT_SEL7	Package	Uncore R-box 0 perfmon event select MSR
E1FH	3615	MSR_R0_PMON_CTR7	Package	Uncore R-box 0 perfmon counter MSR
E20H	3616	MSR_R1_PMON_BOX_CTRL	Package	Uncore R-box 1 perfmon local box control MSR
E21H	3617	MSR_R1_PMON_BOX_STATUS	Package	Uncore R-box 1 perfmon local box status MSR
E22H	3618	MSR_R1_PMON_BOX_OVF_CTRL	Package	Uncore R-box 1 perfmon local box overflow control MSR
E24H	3620	MSR_R1_PMON_IPERF1_P8	Package	Uncore R-box 1 perfmon IPERF1 unit Port 8 select MSR
E25H	3621	MSR_R1_PMON_IPERF1_P9	Package	Uncore R-box 1 perfmon IPERF1 unit Port 9 select MSR
E26H	3622	MSR_R1_PMON_IPERF1_P10	Package	Uncore R-box 1 perfmon IPERF1 unit Port 10 select MSR
E27H	3623	MSR_R1_PMON_IPERF1_P11	Package	Uncore R-box 1 perfmon IPERF1 unit Port 11 select MSR
E28H	3624	MSR_R1_PMON_IPERF1_P12	Package	Uncore R-box 1 perfmon IPERF1 unit Port 12 select MSR
E29H	3625	MSR_R1_PMON_IPERF1_P13	Package	Uncore R-box 1 perfmon IPERF1 unit Port 13 select MSR
E2AH	3626	MSR_R1_PMON_IPERF1_P14	Package	Uncore R-box 1 perfmon IPERF1 unit Port 14 select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E2BH	3627	MSR_R1_PMON_IPERF1_P15	Package	Uncore R-box 1 perfmon IPERF1 unit Port 15 select MSR
E2CH	3628	MSR_R1_PMON_QLX_P4	Package	Uncore R-box 1 perfmon QLX unit Port 4 select MSR
E2DH	3629	MSR_R1_PMON_QLX_P5	Package	Uncore R-box 1 perfmon QLX unit Port 5 select MSR
E2EH	3630	MSR_R1_PMON_QLX_P6	Package	Uncore R-box 1 perfmon QLX unit Port 6 select MSR
E2FH	3631	MSR_R1_PMON_QLX_P7	Package	Uncore R-box 1 perfmon QLX unit Port 7 select MSR
E30H	3632	MSR_R1_PMON_EVNT_SEL8	Package	Uncore R-box 1 perfmon event select MSR
E31H	3633	MSR_R1_PMON_CTR8	Package	Uncore R-box 1 perfmon counter MSR
E32H	3634	MSR_R1_PMON_EVNT_SEL9	Package	Uncore R-box 1 perfmon event select MSR
E33H	3635	MSR_R1_PMON_CTR9	Package	Uncore R-box 1 perfmon counter MSR
E34H	3636	MSR_R1_PMON_EVNT_SEL10	Package	Uncore R-box 1 perfmon event select MSR
E35H	3637	MSR_R1_PMON_CTR10	Package	Uncore R-box 1 perfmon counter MSR
E36H	3638	MSR_R1_PMON_EVNT_SEL11	Package	Uncore R-box 1 perfmon event select MSR
E37H	3639	MSR_R1_PMON_CTR11	Package	Uncore R-box 1 perfmon counter MSR
E38H	3640	MSR_R1_PMON_EVNT_SEL12	Package	Uncore R-box 1 perfmon event select MSR
E39H	3641	MSR_R1_PMON_CTR12	Package	Uncore R-box 1 perfmon counter MSR
E3AH	3642	MSR_R1_PMON_EVNT_SEL13	Package	Uncore R-box 1 perfmon event select MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E3BH	3643	MSR_R1_PMON_CTR13	Package	Uncore R-box 1perfmon counter MSR
E3CH	3644	MSR_R1_PMON_EVNT_SEL14	Package	Uncore R-box 1 perfmon event select MSR
E3DH	3645	MSR_R1_PMON_CTR14	Package	Uncore R-box 1 perfmon counter MSR
E3EH	3646	MSR_R1_PMON_EVNT_SEL15	Package	Uncore R-box 1 perfmon event select MSR
E3FH	3647	MSR_R1_PMON_CTR15	Package	Uncore R-box 1 perfmon counter MSR
E45H	3653	MSR_B0_PMON_MATCH	Package	Uncore B-box 0 perfmon local box match MSR
E46H	3654	MSR_B0_PMON_MASK	Package	Uncore B-box 0 perfmon local box mask MSR
E49H	3657	MSR_S0_PMON_MATCH	Package	Uncore S-box 0 perfmon local box match MSR
E4AH	3658	MSR_S0_PMON_MASK	Package	Uncore S-box 0 perfmon local box mask MSR
E4DH	3661	MSR_B1_PMON_MATCH	Package	Uncore B-box 1 perfmon local box match MSR
E4EH	3662	MSR_B1_PMON_MASK	Package	Uncore B-box 1 perfmon local box mask MSR
E54H	3668	MSR_M0_PMON_MM_CONFIG	Package	Uncore M-box 0 perfmon local box address match/mask config MSR
E55H	3669	MSR_M0_PMON_ADDR_MATCH	Package	Uncore M-box 0 perfmon local box address match MSR
E56H	3670	MSR_M0_PMON_ADDR_MASK	Package	Uncore M-box 0 perfmon local box address mask MSR
E59H	3673	MSR_S1_PMON_MATCH	Package	Uncore S-box 1 perfmon local box match MSR
E5AH	3674	MSR_S1_PMON_MASK	Package	Uncore S-box 1 perfmon local box mask MSR
E5CH	3676	MSR_M1_PMON_MM_CONFIG	Package	Uncore M-box 1 perfmon local box address match/mask config MSR

Table 34-7. Additional MSRs in Intel Xeon Processor 7500 Series (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
E5DH	3677	MSR_M1_PMON_ADDR_MATCH	Package	Uncore M-box 1 perfmon local box address match MSR
E5EH	3678	MSR_M1_PMON_ADDR_MASK	Package	Uncore M-box 1 perfmon local box address mask MSR
3B5H	965	MSR_UNCORE_PMC5	Package	See Section 18.6.2.2, "Uncore Performance Event Configuration Facility."

34.5 MSRS IN THE INTEL XEON PROCESSOR 5600 SERIES (INTEL® MICROARCHITECTURE CODE NAME WESTMERE)

Intel Xeon processor 5600 series (Intel® microarchitecture code name Westmere) supports the MSR interfaces listed in Table 34-5, Table 34-6, plus additional MSR listed in Table 34-8. These MSRs also apply to Intel Core i7, i5 and i3 processor family with CPUID signature DisplayFamily_DisplayModel of 06_25H and 06_2CH, see Table 34-1.

Table 34-8. Additional MSRs Supported by Intel Processors (Intel Microarchitecture Code Name Westmere)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	**Maximum Ratio Limit of Turbo Mode.** RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C.** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C.** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C.** Maximum turbo ratio limit of 3 core active.

Table 34-8. Additional MSRs Supported by Intel Processors (Contd.)(Intel Microarchitecture Code Name Westmere)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		31:24	Package	**Maximum Ratio Limit for 4C.** Maximum turbo ratio limit of 4 core active.
		39:32	Package	**Maximum Ratio Limit for 5C.** Maximum turbo ratio limit of 5 core active.
		47:40	Package	**Maximum Ratio Limit for 6C.** Maximum turbo ratio limit of 6 core active.
		63:48		Reserved.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 34-2.

34.6 MSRS IN THE INTEL XEON PROCESSOR E7 FAMILY (INTEL® MICROARCHITECTURE CODE NAME WESTMERE)

Intel Xeon processor E7 family (Intel® microarchitecture code name Westmere) supports the MSR interfaces listed in Table 34-5 (except MSR address 1ADH), Table 34-6, plus additional MSR listed in Table 34-9.

Table 34-9. Additional MSRs Supported by Intel Xeon Processor E7 Family

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1A7H	423	MSR_OFFCORE_RSP_1	Thread	Offcore Response Event Select Register (R/W)
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Reserved. Attempt to read/write will cause #UD
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 34-2.
F40H	3904	MSR_C8_PMON_BOX_CTRL	Package	Uncore C-box 8 perfmon local box control MSR

Table 34-9. Additional MSRs Supported by Intel Xeon Processor E7 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
F41H	3905	MSR_C8_PMON_BOX_STATUS	Package	Uncore C-box 8 perfmon local box status MSR
F42H	3906	MSR_C8_PMON_BOX_OVF_CTRL	Package	Uncore C-box 8 perfmon local box overflow control MSR
F50H	3920	MSR_C8_PMON_EVNT_SEL0	Package	Uncore C-box 8 perfmon event select MSR
F51H	3921	MSR_C8_PMON_CTR0	Package	Uncore C-box 8 perfmon counter MSR
F52H	3922	MSR_C8_PMON_EVNT_SEL1	Package	Uncore C-box 8 perfmon event select MSR
F53H	3923	MSR_C8_PMON_CTR1	Package	Uncore C-box 8 perfmon counter MSR
F54H	3924	MSR_C8_PMON_EVNT_SEL2	Package	Uncore C-box 8 perfmon event select MSR
F55H	3925	MSR_C8_PMON_CTR2	Package	Uncore C-box 8 perfmon counter MSR
F56H	3926	MSR_C8_PMON_EVNT_SEL3	Package	Uncore C-box 8 perfmon event select MSR
F57H	3927	MSR_C8_PMON_CTR3	Package	Uncore C-box 8 perfmon counter MSR
F58H	3928	MSR_C8_PMON_EVNT_SEL4	Package	Uncore C-box 8 perfmon event select MSR
F59H	3929	MSR_C8_PMON_CTR4	Package	Uncore C-box 8 perfmon counter MSR
F5AH	3930	MSR_C8_PMON_EVNT_SEL5	Package	Uncore C-box 8 perfmon event select MSR
F5BH	3931	MSR_C8_PMON_CTR5	Package	Uncore C-box 8 perfmon counter MSR
FC0H	4032	MSR_C9_PMON_BOX_CTRL	Package	Uncore C-box 9 perfmon local box control MSR
FC1H	4033	MSR_C9_PMON_BOX_STATUS	Package	Uncore C-box 9 perfmon local box status MSR

Table 34-9. Additional MSRs Supported by Intel Xeon Processor E7 Family (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
FC2H	4034	MSR_C9_PMON_BOX_OVF_CTRL	Package	Uncore C-box 9 perfmon local box overflow control MSR
FD0H	4048	MSR_C9_PMON_EVNT_SEL0	Package	Uncore C-box 9 perfmon event select MSR
FD1H	4049	MSR_C9_PMON_CTR0	Package	Uncore C-box 9 perfmon counter MSR
FD2H	4050	MSR_C9_PMON_EVNT_SEL1	Package	Uncore C-box 9 perfmon event select MSR
FD3H	4051	MSR_C9_PMON_CTR1	Package	Uncore C-box 9 perfmon counter MSR
FD4H	4052	MSR_C9_PMON_EVNT_SEL2	Package	Uncore C-box 9 perfmon event select MSR
FD5H	4053	MSR_C9_PMON_CTR2	Package	Uncore C-box 9 perfmon counter MSR
FD6H	4054	MSR_C9_PMON_EVNT_SEL3	Package	Uncore C-box 9 perfmon event select MSR
FD7H	4055	MSR_C9_PMON_CTR3	Package	Uncore C-box 9 perfmon counter MSR
FD8H	4056	MSR_C9_PMON_EVNT_SEL4	Package	Uncore C-box 9 perfmon event select MSR
FD9H	4057	MSR_C9_PMON_CTR4	Package	Uncore C-box 9 perfmon counter MSR
FDAH	4058	MSR_C9_PMON_EVNT_SEL5	Package	Uncore C-box 9 perfmon event select MSR
FDBH	4059	MSR_C9_PMON_CTR5	Package	Uncore C-box 9 perfmon counter MSR

34.7 MSRS IN INTEL® PROCESSOR FAMILY (INTEL® MICROARCHITECTURE CODE NAME SANDY BRIDGE)

Table 34-10 lists model-specific registers (MSRs) that are common to Intel® processor family based on Intel® microarchitecture (Sandy Bridge). All architectural MSRs listed in Table 34-2 are supported. These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH, 06_2DH, see Table 34-1. Additional MSRs specific to 06_2AH are listed in Table 34-11.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
0H	0	IA32_P5_MC_ADDR	Thread	See Section 34.12, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	Thread	See Section 34.12, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_FILTER_SIZE	Thread	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and Table 34-2.
10H	16	IA32_TIME_STAMP_COUNTER	Thread	See Section 17.12, "Time-Stamp Counter," and see Table 34-2.
17H	23	IA32_PLATFORM_ID	Package	**Platform ID. (R)** See Table 34-2.
1BH	27	IA32_APIC_BASE	Thread	See Section 10.4.4, "Local APIC Status and Location," and Table 34-2.
34H	52	MSR_SMI_COUNT	Thread	**SMI Counter. (R/O).**
		31:0		**SMI Count. (R/O)** Count SMIs
		63:32		Reserved.
3AH	58	IA32_FEATURE_CONTROL	Thread	**Control Features in Intel 64Processor. (R/W).** See Table 34-2.
79H	121	IA32_BIOS_UPDT_TRIG	Core	**BIOS Update Trigger Register. (W)** See Table 34-2.
8BH	139	IA32_BIOS_SIGN_ID	Thread	**BIOS Update Signature ID. (RO)** See Table 34-2.
C1H	193	IA32_PMC0	Thread	**Performance counter register.** See Table 34-2.
C2H	194	IA32_PMC1	Thread	**Performance counter register.** See Table 34-2.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C3H	195	IA32_PMC2	Thread	**Performance counter register.** See Table 34-2.
C4H	196	IA32_PMC3	Thread	**Performance counter register.** See Table 34-2.
C5H	197	IA32_PMC4	Core	**Performance counter register.** See Table 34-2.
C6H	198	IA32_PMC5	Core	**Performance counter register.** See Table 34-2.
C7H	199	IA32_PMC6	Core	**Performance counter register.** See Table 34-2.
C8H	200	IA32_PMC7	Core	**Performance counter register.** See Table 34-2.
CEH	206	MSR_PLATFORM_INFO	Package	See http://biosbits.org.
		7:0		Reserved.
		15:8	Package	**Maximum Non-Turbo Ratio. (R/O)** The is the ratio of the frequency that invariant TSC runs at. Frequency = ratio * 100 MHz.
		27:16		Reserved.
		28	Package	**Programmable Ratio Limit for Turbo Mode. (R/O)** When set to 1, indicates that Programmable Ratio Limits for Turbo mode is enabled, and when set to 0, indicates Programmable Ratio Limits for Turbo mode is disabled.
		29	Package	**Programmable TDP Limit for Turbo Mode. (R/O)** When set to 1, indicates that TDP Limits for Turbo mode are programmable, and when set to 0, indicates TDP Limit for Turbo mode is not programmable.
		39:30		Reserved.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		47:40	Package	**Maximum Efficiency Ratio. (R/O)** The is the minimum ratio (maximum efficiency) that the processor can operates, in units of 100MHz.
		63:48		Reserved.
E2H	226	MSR_PKG_CST_CONFIG_CONTROL	Core	C-State Configuration Control (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States. See `http://biosbits.org`.
		2:0		**Package C-State limit. (R/W)** Specifies the lowest processor-specific C-state code name (consuming the least power). for the package. The default is set as factory-configured package C-state limit. The following C-state code name encodings are supported: 000b: C0/C1 (no package C-sate support) 001b: C2 010b: C6 no retention 011b: C6 retention 100b: C7 101b: C7s 111: No package C-state limit. Note: This field cannot be used to limit package C-state to C3.
		9:3		**Reserved.**
		10		**I/O MWAIT Redirection Enable. (R/W)** When set, will map IO_read instructions sent to IO register specified by MSR_PMG_IO_CAPTURE_BASE to MWAIT instructions
		14:11		**Reserved.**

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15		**CFG Lock. (R/WO)** When set, lock bits 15:0 of this register until next reset.
		24:16		**Reserved.**
		25		**C3 state auto demotion enable. (R/W)** When set, the processor will conditionally demote C6/C7 requests to C3 based on uncore auto-demote information.
		26		**C1 state auto demotion enable. (R/W)** When set, the processor will conditionally demote C3/C6/C7 requests to C1 based on uncore auto-demote information.
		27		**Enable C3 undemotion (R/W)** When set, enables undemotion from demoted C3.
		28		**Enable C1 undemotion (R/W)** When set, enables undemotion from demoted C1.
		63:29		Reserved.
E4H	228	MSR_PMG_IO_CAPTURE_BASE	Core	Power Management IO Redirection in C-state (R/W) See `http://biosbits.org`.
		15:0		**LVL_2 Base Address. (R/W)** Specifies the base address visible to software for IO redirection. If IO MWAIT Redirection is enabled, reads to this address will be consumed by the power management logic and decoded to MWAIT instructions. When IO port address redirection is enabled, this is the IO port address reported to the OS/software.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		18:16		**C-state Range. (R/W)** Specifies the encoding value of the maximum C-State code name to be included when IO read to MWAIT redirection is enabled by MSR_PMG_CST_CONFIG_CONTROL[bit10]: 000b - C3 is the max C-State to include 001b - C6 is the max C-State to include 010b - C7 is the max C-State to include
		63:19		Reserved.
E7H	231	IA32_MPERF	Thread	**Maximum Performance Frequency Clock Count. (RW)** See Table 34-2.
E8H	232	IA32_APERF	Thread	**Actual Performance Frequency Clock Count. (RW)** See Table 34-2.
FEH	254	IA32_MTRRCAP	Thread	See Table 34-2.
174H	372	IA32_SYSENTER_CS	Thread	See Table 34-2.
175H	373	IA32_SYSENTER_ESP	Thread	See Table 34-2.
176H	374	IA32_SYSENTER_EIP	Thread	See Table 34-2.
179H	377	IA32_MCG_CAP	Thread	See Table 34-2.
17AH	378	IA32_MCG_STATUS	Thread	
		0		**RIPV.** When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If cleared, the program cannot be reliably restarted.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		1		EIPV. When set, bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP.** When set, bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Thread	See Table 34-2.
187H	391	IA32_ PERFEVTSEL1	Thread	See Table 34-2.
188H	392	IA32_ PERFEVTSEL2	Thread	See Table 34-2.
189H	393	IA32_ PERFEVTSEL3	Thread	See Table 34-2.
18AH	394	IA32_ PERFEVTSEL4	Core	See Table 34-2; If CPUID.0AH:EAX[15:8] = 8
18BH	395	IA32_ PERFEVTSEL5	Core	See Table 34-2; If CPUID.0AH:EAX[15:8] = 8
18CH	396	IA32_ PERFEVTSEL6	Core	See Table 34-2; If CPUID.0AH:EAX[15:8] = 8
18DH	397	IA32_ PERFEVTSEL7	Core	See Table 34-2; If CPUID.0AH:EAX[15:8] = 8
198H	408	IA32_PERF_STAT US	Package	See Table 34-2.
		15:0		Current Performance State Value.
		63:16		Reserved.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
198H	408	MSR_PERF_STATUS	Package	
		47:32		Core Voltage (R/O) P-state core voltage can be computed by MSR_PERF_STATUS[37:32] * (float) 1/(2^13).
199H	409	IA32_PERF_CTL	Thread	See Table 34-2.
19AH	410	IA32_CLOCK_MODULATION	Thread	**Clock Modulation. (R/W)** see Table 34-2 IA32_CLOCK_MODULATION MSR was originally named IA32_THERM_CONTROL MSR.
		3:0		On demand Clock Modulation Duty Cycle (R/W). In 6.25% increment
		4		On demand Clock Modulation Enable (R/W).
		63:5		Reserved.
19BH	411	IA32_THERM_INTERRUPT	Core	**Thermal Interrupt Control. (R/W)** See Table 34-2.
19CH	412	IA32_THERM_STATUS	Core	**Thermal Monitor Status. (R/W)** See Table 34-2.
1A0	416	IA32_MISC_ENABLE		**Enable Misc. Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.
		0	Thread	**Fast-Strings Enable.** See Table 34-2
		6:1		Reserved.
		7	Thread	**Performance Monitoring Available. (R)** See Table 34-2.
		10:8		Reserved.
		11	Thread	**Branch Trace Storage Unavailable. (RO)** See Table 34-2.
		12	Thread	**Precise Event Based Sampling Unavailable. (RO)** See Table 34-2.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		15:13		Reserved.
		16	Package	**Enhanced Intel SpeedStep Technology Enable. (R/W)** See Table 34-2.
		18	Thread	ENABLE MONITOR FSM. (R/W) See Table 34-2.
		21:19		Reserved.
		22	Thread	**Limit CPUID Maxval. (R/W)** See Table 34-2.
		23	Thread	**xTPR Message Disable. (R/W)** See Table 34-2.
		33:24		Reserved.
		34	Thread	**XD Bit Disable. (R/W)** See Table 34-2.
		37:35		Reserved.
		38	Package	**Turbo Mode Disable. (R/W)** When set to 1 on processors that support Intel Turbo Boost Technology, the turbo mode feature is disabled and the IDA_Enable feature flag will be clear (CPUID.06H: EAX[1]=0). When set to a 0 on processors that support IDA, CPUID.06H: EAX[1] reports the processor's support of turbo mode is enabled. **Note:** the power-on default value is used by BIOS to detect hardware support of turbo mode. If power-on default value is 1, turbo mode is available in the processor. If power-on default value is 0, turbo mode is not available.
		63:39		Reserved.
1A2H	418	MSR_TEMPERATURE_TARGET	Unique	
		15:0		Reserved.
		23:16		**Temperature Target. (R)** The minimum temperature at which PROCHOT# will be asserted. The value is degree C.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:24		Reserved.
1A6H	422	MSR_OFFCORE_RSP_0	Thread	**Offcore Response Event Select Register** (R/W)
1AAH	426	MSR_MISC_PWR_MGMT		See http://biosbits.org.
1ACH	428	MSR_TURBO_PWR_CURRENT_LIMIT		See http://biosbits.org.
1B0H	432	IA32_ENERGY_PERF_BIAS	Package	See Table 34-2.
1B1H	433	IA32_PACKAGE_THERM_STATUS	Package	See Table 34-2.
1B2H	434	IA32_PACKAGE_THERM_INTERRUPT	Package	See Table 34-2.
1C8H	456	MSR_LBR_SELECT	Thread	**Last Branch Record Filtering Select Register** (R/W) See Section 17.6.2, "Filtering of Last Branch Records."
1C9H	457	MSR_LASTBRANCH_TOS	Thread	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 680H).
1D9H	473	IA32_DEBUGCTL	Thread	**Debug Control. (R/W)** See Table 34-2.
1DDH	477	MSR_LER_FROM_LIP	Thread	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Thread	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1F2H	498	IA32_SMRR_PHYSBASE	Core	See Table 34-2.
1F3H	499	IA32_SMRR_PHYSMASK	Core	See Table 34-2.
1FCH	508	MSR_POWER_CTL	Core	See http://biosbits.org.
200H	512	IA32_MTRR_PHYSBASE0	Thread	See Table 34-2.
201H	513	IA32_MTRR_PHYSMASK0	Thread	See Table 34-2.
202H	514	IA32_MTRR_PHYSBASE1	Thread	See Table 34-2.
203H	515	IA32_MTRR_PHYSMASK1	Thread	See Table 34-2.
204H	516	IA32_MTRR_PHYSBASE2	Thread	See Table 34-2.
205H	517	IA32_MTRR_PHYSMASK2	Thread	See Table 34-2.
206H	518	IA32_MTRR_PHYSBASE3	Thread	See Table 34-2.
207H	519	IA32_MTRR_PHYSMASK3	Thread	See Table 34-2.
208H	520	IA32_MTRR_PHYSBASE4	Thread	See Table 34-2.
209H	521	IA32_MTRR_PHYSMASK4	Thread	See Table 34-2.
20AH	522	IA32_MTRR_PHYSBASE5	Thread	See Table 34-2.
20BH	523	IA32_MTRR_PHYSMASK5	Thread	See Table 34-2.
20CH	524	IA32_MTRR_PHYSBASE6	Thread	See Table 34-2.
20DH	525	IA32_MTRR_PHYSMASK6	Thread	See Table 34-2.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
20EH	526	IA32_MTRR_PHYSBASE7	Thread	See Table 34-2.
20FH	527	IA32_MTRR_PHYSMASK7	Thread	See Table 34-2.
210H	528	IA32_MTRR_PHYSBASE8	Thread	See Table 34-2.
211H	529	IA32_MTRR_PHYSMASK8	Thread	See Table 34-2.
212H	530	IA32_MTRR_PHYSBASE9	Thread	See Table 34-2.
213H	531	IA32_MTRR_PHYSMASK9	Thread	See Table 34-2.
250H	592	IA32_MTRR_FIX64K_00000	Thread	See Table 34-2.
258H	600	IA32_MTRR_FIX16K_80000	Thread	See Table 34-2.
259H	601	IA32_MTRR_FIX16K_A0000	Thread	See Table 34-2.
268H	616	IA32_MTRR_FIX4K_C0000	Thread	See Table 34-2.
269H	617	IA32_MTRR_FIX4K_C8000	Thread	See Table 34-2.
26AH	618	IA32_MTRR_FIX4K_D0000	Thread	See Table 34-2.
26BH	619	IA32_MTRR_FIX4K_D8000	Thread	See Table 34-2.
26CH	620	IA32_MTRR_FIX4K_E0000	Thread	See Table 34-2.
26DH	621	IA32_MTRR_FIX4K_E8000	Thread	See Table 34-2.
26EH	622	IA32_MTRR_FIX4K_F0000	Thread	See Table 34-2.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
26FH	623	IA32_MTRR_FIX4K_F8000	Thread	See Table 34-2.
277H	631	IA32_PAT	Thread	See Table 34-2.
280H	640	IA32_MC0_CTL2	Core	See Table 34-2.
281H	641	IA32_MC1_CTL2	Core	See Table 34-2.
282H	642	IA32_MC2_CTL2	Core	See Table 34-2.
283H	643	IA32_MC3_CTL2	Core	See Table 34-2.
284H	644	MSR_MC4_CTL2	Package	Always 0 (CMCI not supported).
2FFH	767	IA32_MTRR_DEF_TYPE	Thread	**Default Memory Types. (R/W)** See Table 34-2.
309H	777	IA32_FIXED_CTR0	Thread	**Fixed-Function Performance Counter Register 0. (R/W)** See Table 34-2.
30AH	778	IA32_FIXED_CTR1	Thread	**Fixed-Function Performance Counter Register 1. (R/W)** See Table 34-2.
30BH	779	IA32_FIXED_CTR2	Thread	**Fixed-Function Performance Counter Register 2. (R/W)** See Table 34-2.
345H	837	IA32_PERF_CAPABILITIES	Thread	See Table 34-2. See Section 17.4.1, “IA32_DEBUGCTL MSR.”
		5:0		LBR Format. See Table 34-2.
		6		PEBS Record Format.
		7		PEBSSaveArchRegs. See Table 34-2.
		11:8		PEBS_REC_FORMAT. See Table 34-2.
		12		SMM_FREEZE. See Table 34-2.
		63:13		Reserved.
38DH	909	IA32_FIXED_CTR_CTRL	Thread	**Fixed-Function-Counter Control Register. (R/W)** See Table 34-2.
38EH	910	IA32_PERF_GLOBAL_STAUS	Thread	See Table 34-2. See Section 18.4.2, “Global Counter Control Facilities.”
38FH	911	IA32_PERF_GLOBAL_CTRL	Thread	See Table 34-2. See Section 18.4.2, “Global Counter Control Facilities.”

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
390H	912	IA32_PERF_GLOBAL_OVF_CTRL	Thread	See Table 34-2. See Section 18.4.2, "Global Counter Control Facilities."
391H	913	MSR_UNC_PERF_GLOBAL_CTRL	Package	Uncore PMU global control
		0		Core 0 select
		1		Core 1 select
		2		Core 2 select
		3		Core 3 select
		18:4		Reserved.
		29		Enable all uncore counters
		30		Enable PMI on overflow
		31		Enable Freezing counter when overflow
		63:32		Reserved.
392H	914	MSR_UNC_PERF_GLOBAL_STATUS	Package	Uncore PMU main status
		0		Fixed counter overflowed
		1		CBox counter overflowed
		63:2		Reserved.
394H	916	MSR_UNC_PERF_FIXED_CTRL	Package	Uncore fixed counter control (R/W)
		19:0		Reserved
		20		Enable overflow
		21		Reserved
		22		Enable counting
		63:23		Reserved.
395H	917	MSR_UNC_PERF_FIXED_CTR	Package	Uncore fixed counter
		47:0		Current count
		63:48		Reserved.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
3F1H	1009	MSR_PEBS_ENABLE	Thread	See See Section 18.6.1.1, "Precise Event Based Sampling (PEBS)."
		0		Enable PEBS on IA32_PMC0. (R/W)
		1		Enable PEBS on IA32_PMC1. (R/W)
		2		Enable PEBS on IA32_PMC2. (R/W)
		3		Enable PEBS on IA32_PMC3. (R/W)
		31:4		Reserved.
		32		Enable Load Latency on IA32_PMC0. (R/W)
		33		Enable Load Latency on IA32_PMC1. (R/W)
		34		Enable Load Latency on IA32_PMC2. (R/W)
		35		Enable Load Latency on IA32_PMC3. (R/W)
		63:36		Reserved.
3F6H	1014	MSR_PEBS_LD_LAT	Thread	see See Section 18.6.1.2, "Load Latency Performance Monitoring Facility."
		15:0		Minimum threshold latency value of tagged load operation that will be counted. (R/W)
		63:36		Reserved.
3F8H	1016	MSR_PKG_C3_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C3 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C3 states. Count at the same frequency as the TSC.
3F9H	1017	MSR_PKG_C6_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:0		Package C6 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C6 states. Count at the same frequency as the TSC.
3FAH	1018	MSR_PKG_C7_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C7 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C7 states. Count at the same frequency as the TSC.
3FCH	1020	MSR_CORE_C3_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C3 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C3 states. Count at the same frequency as the TSC.
3FDH	1021	MSR_CORE_C6_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C6 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C6 states. Count at the same frequency as the TSC.
3FEH	1022	MSR_CORE_C7_RESIDENCY	Core	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		CORE C7 Residency Counter. (R/O) Value since last reset that this core is in processor-specific C7 states. Count at the same frequency as the TSC.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
400H	1024	IA32_MC0_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_ STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
402H	1026	IA32_MC0_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
403H	1027	IA32_MC0_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
404H	1028	IA32_MC1_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
406H	1030	IA32_MC1_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
407H	1031	IA32_MC1_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
408H	1032	IA32_MC2_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40AH	1034	IA32_MC2_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40BH	1035	IA32_MC2_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
40CH	1036	IA32_MC3_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_ STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
40EH	1038	IA32_MC3_ADDR	Core	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
40FH	1039	IA32_MC3_MISC	Core	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
410H	1040	MSR_MC4_CTL	Core	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
		0		**PCU Hardware Error. (R/W)** When set, enables signaling of PCU hardware detected errors.
		1		**PCU Controller Error. (R/W)** When set, enables signaling of PCU controller detected errors
		2		**PCU Firmware Error. (R/W)** When set, enables signaling of PCU firmware detected errors
		63:2		Reserved.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
411H	1041	IA32_MC4_STATUS	Core	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
480H	1152	IA32_VMX_BASIC	Thread	**Reporting Register of Basic VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	Thread	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Table 34-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	Thread	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
483H	1155	IA32_VMX_EXIT_CTLS	Thread	**Capability Reporting Register of VM-exit Controls. (R/O)** See Table 34-2. See Appendix A.4, "VM-Exit Controls."
484H	1156	IA32_VMX_ENTRY_CTLS	Thread	**Capability Reporting Register of VM-entry Controls. (R/O)** See Table 34-2. See Appendix A.5, "VM-Entry Controls."
485H	1157	IA32_VMX_MISC	Thread	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Table 34-2. See Appendix A.6, "Miscellaneous Data."
486H	1158	IA32_VMX_CR0_FIXED0	Thread	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
487H	1159	IA32_VMX_CR0_FIXED1	Thread	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.7, "VMX-Fixed Bits in CR0."
488H	1160	IA32_VMX_CR4_FIXED0	Thread	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
489H	1161	IA32_VMX_CR4_FIXED1	Thread	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Table 34-2. See Appendix A.8, "VMX-Fixed Bits in CR4."
48AH	1162	IA32_VMX_VMCS_ENUM	Thread	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Table 34-2. See Appendix A.9, "VMCS Enumeration."
48BH	1163	IA32_VMX_PROCBASED_CTLS2	Thread	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls."
4C1H	1217	IA32_A_PMC0	Thread	See Table 34-2.
4C2H	1218	IA32_A_PMC1	Thread	See Table 34-2.
4C3H	1219	IA32_A_PMC2	Thread	See Table 34-2.
4C4H	1220	IA32_A_PMC3	Thread	See Table 34-2.
4C5H	1221	IA32_A_PMC4	Core	See Table 34-2.
4C6H	1222	IA32_A_PMC5	Core	See Table 34-2.
4C7H	1223	IA32_A_PMC6	Core	See Table 34-2.
C8H	200	IA32_A_PMC7	Core	See Table 34-2.
600H	1536	IA32_DS_AREA	Thread	**DS Save Area. (R/W).** See Table 34-2. See Section 18.9.4, "Debug Store (DS) Mechanism."
606H	1542	MSR_RAPL_POWER_UNIT	Package	Unit Multipliers used in RAPL Interfaces (R/O) See Section 14.7.1, "RAPL Interfaces."
60AH	1546	MSR_PKGC3_IRTL	Package	Package C3 Interrupt Response Limit (R/W) Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit. (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C3 state.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:10		**Time Unit. (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		**Reserved.**
		15		**Valid. (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60BH	1547	MSR_PKGC6_IRTL	Package	**Package C6 Interrupt Response Limit** (R/W) This MSR defines the budget allocated for the package to exit from C6 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit. (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C6 state.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:10		**Time Unit. (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		**Reserved.**
		15		**Valid. (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60CH	1548	MSR_PKGC7_IRTL	Package	**Package C7 Interrupt Response Limit** (R/W) This MSR defines the budget allocated for the package to exit from C7 to a C0 state, where interrupt request can be delivered to the core and serviced. Additional core-exit latency amy be applicable depending on the actual C-state the core is in. Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		9:0		**Interrupt response time limit. (R/W)** Specifies the limit that should be used to decide if the package should be put into a package C7 state.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		12:10		**Time Unit. (R/W)** Specifies the encoding value of time unit of the interrupt response time limit. The following time unit encodings are supported: 000b: 1 ns 001b: 32 ns 010b: 1024 ns 011b: 32768 ns 100b: 1048576 ns 101b: 33554432 ns
		14:13		**Reserved.**
		15		**Valid. (R/W)** Indicates whether the values in bits 12:0 are valid and can be used by the processor for package C-sate management.
		63:16		Reserved.
60DH	1549	MSR_PKG_C2_RESIDENCY	Package	Note: C-state values are processor specific C-state code names, unrelated to MWAIT extension C-state parameters or ACPI C-States.
		63:0		Package C2 Residency Counter. (R/O) Value since last reset that this package is in processor-specific C2 states. Count at the same frequency as the TSC.
610H	1552	MSR_PKG_RAPL_POWER_LIMIT	Package	**PKG RAPL Power Limit Control** (R/W) See Section 14.7.3, "Package RAPL Domain."
611H	1553	MSR_PKG_ENERY_STATUS	Package	**PKG Energy Status** (R/O) See Section 14.7.3, "Package RAPL Domain."
613H	1555	MSR_PKG_PERF_STATUS	Package	**PKG Performance Throttling Status** (R/O) See Section 14.7.3, "Package RAPL Domain."
614H	1556	MSR_PKG_POWER_INFO	Package	**PKG RAPL Parameters** (R/W) See Section 14.7.3, "Package RAPL Domain."
638H	1592	MSR_PP0_POWER_LIMIT	Package	**PP0 RAPL Power Limit Control** (R/W) See Section 14.7.4, "PP0/PP1 RAPL Domains."

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
639H	1593	MSR_PP0_ENERY_STATUS	Package	**PP0 Energy Status** (R/O) See Section 14.7.4, “PP0/PP1 RAPL Domains.”
63AH	1594	MSR_PP0_POLICY	Package	**PP0 Balance Policy** (R/W) See Section 14.7.4, “PP0/PP1 RAPL Domains.”
63BH	1595	MSR_PP0_PERF_STATUS	Package	**PP0 Performance Throttling Status** (R/O) See Section 14.7.4, “PP0/PP1 RAPL Domains.”
680H	1664	MSR_LASTBRANCH_0_FROM_IP	Thread	**Last Branch Record 0 From IP. (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the **source instruction** for one of the last sixteen branches, exceptions, or interrupts taken by the processor. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.6.1, “LBR Stack.”
681H	1665	MSR_LASTBRANCH_1_FROM_IP	Thread	**Last Branch Record 1 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
682H	1666	MSR_LASTBRANCH_2_FROM_IP	Thread	**Last Branch Record 2 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
683H	1667	MSR_LASTBRANCH_3_FROM_IP	Thread	**Last Branch Record 3 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
684H	1668	MSR_LASTBRANCH_4_FROM_IP	Thread	**Last Branch Record 4 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
685H	1669	MSR_LASTBRANCH_5_FROM_IP	Thread	**Last Branch Record 5 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
686H	1670	MSR_LASTBRANCH_6_FROM_IP	Thread	**Last Branch Record 6 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
687H	1671	MSR_ LASTBRANCH_7_F ROM_IP	Thread	**Last Branch Record 7 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
688H	1672	MSR_ LASTBRANCH_8_F ROM_IP	Thread	**Last Branch Record 8 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
689H	1673	MSR_ LASTBRANCH_9_F ROM_IP	Thread	**Last Branch Record 9 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68AH	1674	MSR_ LASTBRANCH_10_ FROM_IP	Thread	**Last Branch Record 10 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68BH	1675	MSR_ LASTBRANCH_11_ FROM_IP	Thread	**Last Branch Record 11 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68CH	1676	MSR_ LASTBRANCH_12_ FROM_IP	Thread	**Last Branch Record 12 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68DH	1677	MSR_ LASTBRANCH_13_ FROM_IP	Thread	**Last Branch Record 13 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68EH	1678	MSR_ LASTBRANCH_14_ FROM_IP	Thread	**Last Branch Record 14 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
68FH	1679	MSR_ LASTBRANCH_15_ FROM_IP	Thread	**Last Branch Record 15 From IP. (R/W)** See description of MSR_LASTBRANCH_0_FROM_IP.
6C0H	1728	MSR_ LASTBRANCH_0_ TO_LIP	Thread	**Last Branch Record 0 To IP. (R/W)** One of sixteen pairs of last branch record registers on the last branch record stack. This part of the stack contains pointers to the destination instruction for one of the last sixteen branches, exceptions, or interrupts taken by the processor.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	**Dec**			
6C1H	1729	MSR_ LASTBRANCH_1_ TO_LIP	Thread	**Last Branch Record 1 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C2H	1730	MSR_ LASTBRANCH_2_ TO_LIP	Thread	**Last Branch Record 2 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C3H	1731	MSR_ LASTBRANCH_3_ TO_LIP	Thread	**Last Branch Record 3 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C4H	1732	MSR_ LASTBRANCH_4_ TO_LIP	Thread	**Last Branch Record 4 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C5H	1733	MSR_ LASTBRANCH_5_ TO_LIP	Thread	**Last Branch Record 5 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C6H	1734	MSR_ LASTBRANCH_6_ TO_LIP	Thread	**Last Branch Record 6 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C7H	1735	MSR_ LASTBRANCH_7_ TO_LIP	Thread	**Last Branch Record 7 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C8H	1736	MSR_ LASTBRANCH_8_ TO_LIP	Thread	**Last Branch Record 8 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6C9H	1737	MSR_ LASTBRANCH_9_ TO_LIP	Thread	**Last Branch Record 9 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CAH	1738	MSR_ LASTBRANCH_10_ TO_LIP	Thread	**Last Branch Record 10 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CBH	1739	MSR_ LASTBRANCH_11_ TO_LIP	Thread	**Last Branch Record 11 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
6CCH	1740	MSR_ LASTBRANCH_12_ TO_LIP	Thread	**Last Branch Record 12 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CDH	1741	MSR_ LASTBRANCH_13_ TO_LIP	Thread	**Last Branch Record 13 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CEH	1742	MSR_ LASTBRANCH_14_ TO_LIP	Thread	**Last Branch Record 14 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6CFH	1743	MSR_ LASTBRANCH_15_ TO_LIP	Thread	**Last Branch Record 15 To IP. (R/W)** See description of MSR_LASTBRANCH_0_TO_LIP.
6E0H	1760	IA32_TSC_DEADLI NE	Thread	See Table 34-2.
700H	1792	MSR_UNC_CBO_0_ PERFEVTSEL0	Package	Uncore C-Box 0, counter 0 event select MSR
701H	1793	MSR_UNC_CBO_0_ PERFEVTSEL1	Package	Uncore C-Box 0, counter 1 event select MSR
705H	1797	MSR_UNC_CBO_0_ UNIT_STATUS	Package	Uncore C-Box 0, Overflow Status
706H	1798	MSR_UNC_CBO_0_ PER_CTR0	Package	Uncore C-Box 0, performance counter 0
707H	1799	MSR_UNC_CBO_0_ PER_CTR1	Package	Uncore C-Box 0, performance counter 1
710H	1808	MSR_UNC_CBO_1_ PERFEVTSEL0	Package	Uncore C-Box 1, counter 0 event select MSR
711H	1809	MSR_UNC_CBO_1_ PERFEVTSEL1	Package	Uncore C-Box 1, counter 1 event select MSR
715H	1813	MSR_UNC_CBO_1_ UNIT_STATUS	Package	Uncore C-Box 1, Overflow Status
716H	1814	MSR_UNC_CBO_1_ PER_CTR0	Package	Uncore C-Box 1, performance counter 0

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
717H	1815	MSR_UNC_CBO_1_PER_CTR1	Package	Uncore C-Box 1, performance counter 1
720H	1824	MSR_UNC_CBO_2_PERFEVTSEL0	Package	Uncore C-Box 2, counter 0 event select MSR
721H	1824	MSR_UNC_CBO_2_PERFEVTSEL1	Package	Uncore C-Box 2, counter 1 event select MSR
725H	1829	MSR_UNC_CBO_2_UNIT_STATUS	Package	Uncore C-Box 2, Overflow Status
726H	1830	MSR_UNC_CBO_2_PER_CTR0	Package	Uncore C-Box 2, performance counter 0
727H	1831	MSR_UNC_CBO_2_PER_CTR1	Package	Uncore C-Box 2, performance counter 1
730H	1840	MSR_UNC_CBO_3_PERFEVTSEL0	Package	Uncore C-Box 3, counter 0 event select MSR
731H	1841	MSR_UNC_CBO_3_PERFEVTSEL1	Package	Uncore C-Box 3, counter 1 event select MSR
725H	1845	MSR_UNC_CBO_3_UNIT_STATUS	Package	Uncore C-Box 3, Overflow Status
736H	1846	MSR_UNC_CBO_3_PER_CTR0	Package	Uncore C-Box 3, performance counter 0
737H	1847	MSR_UNC_CBO_3_PER_CTR1	Package	Uncore C-Box 3, performance counter 1
C000_0080H		IA32_EFER	Thread	**Extended Feature Enables.** See Table 34-2.
C000_0081H		IA32_STAR	Thread	**System Call Target Address. (R/W).** See Table 34-2.
C000_0082H		IA32_LSTAR	Thread	**IA-32e Mode System Call Target Address. (R/W).** See Table 34-2.
C000_0084H		IA32_FMASK	Thread	**System Call Flag Mask. (R/W).** See Table 34-2.
C000_0100H		IA32_FS_BASE	Thread	**Map of BASE Address of FS. (R/W).** See Table 34-2.

Table 34-10. MSRs Supported by Intel Processors Based on Intel Microarchitecture Code Name Sandy Bridge (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
C000_ 0101H		IA32_GS_BASE	Thread	**Map of BASE Address of GS. (R/W).** See Table 34-2.
C000_ 0102H		IA32_KERNEL_GS BASE	Thread	**Swap Target of BASE Address of GS. (R/W).** See Table 34-2.
C000_ 0103H		IA32_TSC_AUX	Thread	**AUXILIARY TSC Signature. (R/W).** See Table 34-2 and Section 17.12.2, "IA32_TSC_AUX Register and RDTSCP Support."

34.7.1 MSRs In Second Generation Intel® Core Processor Family (Intel® Microarchitecture Code Name Sandy Bridge)

Table 34-11 lists model-specific registers (MSRs) that are specific to second generation for Intel® Core processor family (Intel® microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2AH, see Table 34-1.

Table 34-11. MSRs Supported by Second Generation Intel Core Processors (Intel Microarchitecture Code Name Sandy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
1ADH	429	MSR_TURBO_RATIO_LIMIT	Package	Maximum Ratio Limit of Turbo Mode. RO if MSR_PLATFORM_INFO.[28] = 0, RW if MSR_PLATFORM_INFO.[28] = 1
		7:0	Package	**Maximum Ratio Limit for 1C.** Maximum turbo ratio limit of 1 core active.
		15:8	Package	**Maximum Ratio Limit for 2C.** Maximum turbo ratio limit of 2 core active.
		23:16	Package	**Maximum Ratio Limit for 3C.** Maximum turbo ratio limit of 3 core active.
		31:24	Package	**Maximum Ratio Limit for 4C.** Maximum turbo ratio limit of 4 core active.

Table 34-11. MSRs Supported by Second Generation Intel Core Processors (Contd.)(Intel Microarchitecture Code Name Sandy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
		63:32		Reserved.
640H	1600	MSR_PP1_POWER _LIMIT	Package	**PP1 RAPL Power Limit Control** (R/W) See Section 14.7.4, "PP0/PP1 RAPL Domains."
641H	1601	MSR_PP1_ENERY_ STATUS	Package	**PP1 Energy Status** (R/O) See Section 14.7.4, "PP0/PP1 RAPL Domains."
642H	1602	MSR_PP1_POLICY	Package	**PP1 Balance Policy** (R/W) See Section 14.7.4, "PP0/PP1 RAPL Domains."

34.7.2 MSRs In Next Generation Intel® Xeon Processor Family (Intel® Microarchitecture Code Name Sandy Bridge)

Table 34-12 lists selected model-specific registers (MSRs) that are specific to the next generation Intel® Xeon processor family (Intel® microarchitecture code name Sandy Bridge). These processors have a CPUID signature with DisplayFamily_DisplayModel of 06_2DH, see Table 34-1.

Table 34-12. Selected MSRs Supported by Next Generation Intel Xeon Processors (Intel Microarchitecture Code Name Sandy Bridge)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
285H	645	IA32_MC5_CTL2	Package	See Table 34-2.
286H	646	IA32_MC6_CTL2	Package	See Table 34-2.
287H	647	IA32_MC7_CTL2	Package	See Table 34-2.
288H	648	IA32_MC8_CTL2	Package	See Table 34-2.
289H	649	IA32_MC9_CTL2	Package	See Table 34-2.
28AH	650	IA32_MC10_CTL2	Package	See Table 34-2.
28BH	651	IA32_MC11_CTL2	Package	See Table 34-2.
28CH	652	IA32_MC12_CTL2	Package	See Table 34-2.
28DH	653	IA32_MC13_CTL2	Package	See Table 34-2.
28EH	654	IA32_MC14_CTL2	Package	See Table 34-2.
28FH	655	IA32_MC15_CTL2	Package	See Table 34-2.

Table 34-12. Selected MSRs Supported by Next Generation Intel Xeon Processors (Intel Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
290H	656	IA32_MC16_CTL2	Package	See Table 34-2.
291H	657	IA32_MC17_CTL2	Package	See Table 34-2.
292H	658	IA32_MC18_CTL2	Package	See Table 34-2.
293H	659	IA32_MC19_CTL2	Package	See Table 34-2.
414H	1044	MSR_MC5_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
415H	1045	MSR_MC5_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
416H	1046	MSR_MC5_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
417H	1047	MSR_MC5_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
418H	1048	MSR_MC6_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
419H	1049	MSR_MC6_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41AH	1050	MSR_MC6_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41BH	1051	MSR_MC6_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
41CH	1052	MSR_MC7_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
41DH	1053	MSR_MC7_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
41EH	1054	MSR_MC7_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
41FH	1055	MSR_MC7_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
420H	1056	MSR_MC8_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
421H	1057	MSR_MC8_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
422H	1058	MSR_MC8_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
423H	1059	MSR_MC8_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
424H	1060	MSR_MC9_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
425H	1061	MSR_MC9_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
426H	1062	MSR_MC9_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
427H	1063	MSR_MC9_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
428H	1064	MSR_MC10_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 34-12. Selected MSRs Supported by Next Generation Intel Xeon Processors (Intel Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
429H	1065	MSR_MC10_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42AH	1066	MSR_MC10_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42BH	1067	MSR_MC10_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
42CH	1068	MSR_MC11_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
42DH	1069	MSR_MC11_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
42EH	1070	MSR_MC11_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
42FH	1071	MSR_MC11_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
430H	1072	MSR_MC12_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
431H	1073	MSR_MC12_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
432H	1074	MSR_MC12_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
433H	1075	MSR_MC12_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
434H	1076	MSR_MC13_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
435H	1077	MSR_MC13_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
436H	1078	MSR_MC13_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
437H	1079	MSR_MC13_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
438H	1080	MSR_MC14_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
439H	1081	MSR_MC14_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43AH	1082	MSR_MC14_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43BH	1083	MSR_MC14_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
43CH	1084	MSR_MC15_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
43DH	1085	MSR_MC15_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
43EH	1086	MSR_MC15_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
43FH	1087	MSR_MC15_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
440H	1088	MSR_MC16_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 34-12. Selected MSRs Supported by Next Generation Intel Xeon Processors (Intel Microarchitecture Code Name Sandy Bridge) (Contd.)

Register Address		Register Name	Scope	Bit Description
Hex	Dec			
441H	1089	MSR_MC16_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
442H	1090	MSR_MC16_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
443H	1091	MSR_MC16_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
444H	1092	MSR_MC17_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
445H	1093	MSR_MC17_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
446H	1094	MSR_MC17_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
447H	1095	MSR_MC17_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
448H	1096	MSR_MC18_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
449H	1097	MSR_MC18_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44AH	1098	MSR_MC18_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44BH	1099	MSR_MC18_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
44CH	1100	MSR_MC19_CTL	Package	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
44DH	1101	MSR_MC19_ STATUS	Package	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS," and Chapter 16.
44EH	1102	MSR_MC19_ADDR	Package	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs."
44FH	1103	MSR_MC19_MISC	Package	See Section 15.3.2.4, "IA32_MCi_MISC MSRs."
618H	1560	MSR_DRAM_POWE R_LIMIT	Package	**DRAM RAPL Power Limit Control** (R/W) See Section 14.7.5, "DRAM RAPL Domain."
619H	1561	MSR_DRAM_ENER Y_STATUS	Package	**DRAM Energy Status** (R/O) See Section 14.7.5, "DRAM RAPL Domain."
61BH	1563	MSR_DRAM_PERF _STATUS	Package	**DRAM Performance Throttling Status** (R/O) See Section 14.7.5, "DRAM RAPL Domain."
61CH	1564	MSR_DRAM_POWE R_INFO	Package	**DRAM RAPL Parameters** (R/W) See Section 14.7.5, "DRAM RAPL Domain."

34.8 MSRS IN THE PENTIUM® 4 AND INTEL® XEON® PROCESSORS

Table 34-13 lists MSRs (architectural and model-specific) that are defined across processor generations based on Intel NetBurst microarchitecture. The processor can be identified by its CPUID signatures of DisplayFamily encoding of 0FH, see Table 34-1.

- MSRs with an "IA32_" prefix are designated as "architectural." This means that the functions of these MSRs and their addresses remain the same for succeeding families of IA-32 processors.
- MSRs with an "MSR_" prefix are model specific with respect to address functionalities. The column "Model Availability" lists the model encoding value(s) within the Pentium 4 and Intel Xeon processor family at the specified register address. The model encoding value of a processor can be queried using CPUID. See "CPUID—CPU Identification" in Chapter 3 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A*.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors

Register Address		Register Name Fields and Flags	Model Availability	Shared/ Unique[1]	Bit Description
Hex	Dec				
0H	0	IA32_P5_MC_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 34.12, "MSRs in Pentium Processors."
1H	1	IA32_P5_MC_TYPE	0, 1, 2, 3, 4, 6	Shared	See Section 34.12, "MSRs in Pentium Processors."
6H	6	IA32_MONITOR_ FILTER_LINE_SIZE	3, 4, 6	Shared	See Section 8.10.5, "Monitor/Mwait Address Range Determination."
10H	16	IA32_TIME_STAMP_ COUNTER	0, 1, 2, 3, 4, 6	Unique	**Time Stamp Counter.** See Table 34-2.
					On earlier processors, only the lower 32 bits are writable. On any write to the lower 32 bits, the upper 32 bits are cleared. For processor family 0FH, models 3 and 4: all 64 bits are writable.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
17H	23	IA32_PLATFORM_ID	0, 1, 2, 3, 4, 6	Shared	**Platform ID. (R).** See Table 34-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	0, 1, 2, 3, 4, 6	Unique	**APIC Location and Status. (R/W)** See Table 34-2. See Section 10.4.4, "Local APIC Status and Location."
2AH	42	MSR_EBC_HARD_POWERON	0, 1, 2, 3, 4, 6	Shared	**Processor Hard Power-On Configuration.** (R/W) Enables and disables processor features; (R) indicates current processor configuration.
		0			**Output Tri-state Enabled. (R)** Indicates whether tri-state output is enabled (1) or disabled (0) as set by the strapping of SMI#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		1			**Execute BIST. (R)** Indicates whether the execution of the BIST is enabled (1) or disabled (0) as set by the strapping of INIT#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		2			**In Order Queue Depth. (R)** Indicates whether the in order queue depth for the system bus is 1 (1) or up to 12 (0) as set by the strapping of A7#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		3			**MCERR# Observation Disabled. (R)** Indicates whether MCERR# observation is enabled (0) or disabled (1) as determined by the strapping of A9#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		4			**BINIT# Observation Enabled. (R)** Indicates whether BINIT# observation is enabled (0) or disabled (1) as determined by the strapping of A10#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		6:5			**APIC Cluster ID. (R)** Contains the logical APIC cluster ID value as set by the strapping of A12# and A11#. The logical cluster ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		7			**Bus Park Disable. (R)** Indicates whether bus park is enabled (0) or disabled (1) as set by the strapping of A15#. The value in this bit is written on the deassertion of RESET#; the bit is set to 1 when the address bus signal is asserted.
		11:8			Reserved.
		13:12			**Agent ID. (R)** Contains the logical agent ID value as set by the strapping of BR[3:0]. The logical ID value is written into the field on the deassertion of RESET#; the field is set to 1 when the address bus signal is asserted.
		63:14			Reserved.
2BH	43	MSR_EBC_SOFT_ POWERON	0, 1, 2, 3, 4, 6	Shared	**Processor Soft Power-On Configuration. (R/W)** Enables and disables processor features.
		0			**RCNT/SCNT On Request Encoding Enable. (R/W)** Controls the driving of RCNT/SCNT on the request encoding. Set to enable (1); clear to disabled (0, default).
		1			**Data Error Checking Disable. (R/W)** Set to disable system data bus parity checking; clear to enable parity checking.
		2			**Response Error Checking Disable. (R/W)** Set to disable (default); clear to enable.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		3			**Address/Request Error Checking Disable. (R/W)** Set to disable (default); clear to enable.
		4			**Initiator MCERR# Disable. (R/W)** Set to disable MCERR# driving for initiator bus requests (default); clear to enable.
		5			**Internal MCERR# Disable. (R/W)** Set to disable MCERR# driving for initiator internal errors (default); clear to enable.
		6			**BINIT# Driver Disable.** (R/W) Set to disable BINIT# driver (default); clear to enable driver.
		63:7			Reserved.
2CH	44	MSR_EBC_ FREQUENCY_ID	2,3, 4, 6	Shared	**Processor Frequency Configuration.** The bit field layout of this MSR varies according to the MODEL value in the CPUID version information. The following bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding equal or greater than 2. (R) The field Indicates the current processor frequency configuration.
		15:0			Reserved.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		18:16			**Scalable Bus Speed. (R/W)** Indicates the intended scalable bus speed: Encoding Scalable Bus Speed 000B 100 MHz (Model 2) 000B 266 MHz (Model 3 or 4) 001B 133 MHz 010B 200 MHz 011B 166 MHz 100B 333 MHz (Model 6) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 011B. 266.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 000B and model encoding = 3 or 4. 333.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 100B and model encoding = 6. All other values are reserved.
		23:19			Reserved.
		31:24			**Core Clock Frequency to System Bus Frequency Ratio. (R)** The processor core clock frequency to system bus frequency ratio observed at the de-assertion of the reset pin.
		63:25			Reserved.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
2CH	44	MSR_EBC_ FREQUENCY_ID	0, 1	Shared	**Processor Frequency Configuration. (R)** The bit field layout of this MSR varies according to the MODEL value of the CPUID version information. This bit field layout applies to Pentium 4 and Xeon Processors with MODEL encoding less than 2. Indicates current processor frequency configuration.
		20:0			Reserved.
		23:21			**Scalable Bus Speed. (R/W)** **Indicates the intended scalable bus speed:** Encoding Scalable Bus Speed 000B 100 MHz All others values reserved.
		63:24			Reserved.
3AH	58	IA32_FEATURE_ CONTROL	3, 4, 6	Unique	**Control Features in IA-32 Processor. (R/W).** See Table 34-2 (If CPUID.01H:ECX.[bit 5])
79H	121	IA32_BIOS_UPDT_ TRIG	0, 1, 2, 3, 4, 6	Shared	**BIOS Update Trigger Register. (W)** See Table 34-2.
8BH	139	IA32_BIOS_SIGN_ID	0, 1, 2, 3, 4, 6	Unique	**BIOS Update Signature ID. (R/W)** See Table 34-2.
9BH	155	IA32_SMM_MONITOR_ CTL	3, 4, 6	Unique	**SMM Monitor Configuration. (R/W).** See Table 34-2.
FEH	254	IA32_MTRRCAP	0, 1, 2, 3, 4, 6	Unique	**MTRR Information.** See Section 11.11.1, "MTRR Feature Identification.".

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
174H	372	IA32_SYSENTER_CS	0, 1, 2, 3, 4, 6	Unique	**CS register target for CPL 0 code. (R/W).** See Table 34-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
175H	373	IA32_SYSENTER_ESP	0, 1, 2, 3, 4, 6	Unique	**Stack pointer for CPL 0 stack. (R/W).** See Table 34-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
176H	374	IA32_SYSENTER_EIP	0, 1, 2, 3, 4, 6	Unique	**CPL 0 code entry point. (R/W).** See Table 34-2. See Section 5.8.7, "Performing Fast Calls to System Procedures with the SYSENTER and SYSEXIT Instructions."
179H	377	IA32_MCG_CAP	0, 1, 2, 3, 4, 6	Unique	**Machine Check Capabilities. (R)** See Table 34-2. See Section 15.3.1.1, "IA32_MCG_CAP MSR."
17AH	378	IA32_MCG_STATUS	0, 1, 2, 3, 4, 6	Unique	**Machine Check Status. (R).** See Table 34-2. **See Section 15.3.1.2, "IA32_MCG_STATUS MSR."**
17BH	379	IA32_MCG_CTL			**Machine Check Feature Enable. (R/W).** See Table 34-2. See Section 15.3.1.3, "IA32_MCG_CTL MSR."
180H	384	MSR_MCG_RAX	0, 1, 2, 3, 4, 6	Unique	**Machine Check EAX/RAX Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
181H	385	MSR_MCG_RBX	0, 1, 2, 3, 4, 6	Unique	**Machine Check EBX/RBX Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
182H	386	MSR_MCG_RCX	0, 1, 2, 3, 4, 6	Unique	**Machine Check ECX/RCX Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
183H	387	MSR_MCG_RDX	0, 1, 2, 3, 4, 6	Unique	**Machine Check EDX/RDX Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
184H	388	MSR_MCG_RSI	0, 1, 2, 3, 4, 6	Unique	**Machine Check ESI/RSI Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
185H	389	MSR_MCG_RDI	0, 1, 2, 3, 4, 6	Unique	**Machine Check EDI/RDI Save State.** See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
186H	390	MSR_MCG_RBP	0, 1, 2, 3, 4, 6	Unique	**Machine Check EBP/RBP Save State.** See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
187H	391	MSR_MCG_RSP	0, 1, 2, 3, 4, 6	Unique	**Machine Check ESP/RSP Save State.** See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
188H	392	MSR_MCG_RFLAGS	0, 1, 2, 3, 4, 6	Unique	**Machine Check EFLAGS/RFLAG Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
189H	393	MSR_MCG_RIP	0, 1, 2, 3, 4, 6	Unique	**Machine Check EIP/RIP Save State.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63:0			Contains register state at time of machine check error. When in non-64-bit modes at the time of the error, bits 63-32 do not contain valid data.
18AH	394	MSR_MCG_MISC	0, 1, 2, 3, 4, 6	Unique	**Machine Check Miscellaneous.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		0			DS. When set, the bit indicates that a page assist or page fault occurred during DS normal operation. The processors response is to shut down. The bit is used as an aid for debugging DS handling code. It is the responsibility of the user (BIOS or operating system) to clear this bit for normal operation.
		63:1			Reserved.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
18BH - 18FH	395	MSR_MCG_ RESERVED1 - MSR_MCG_ RESERVED5			Reserved.
190H	400	MSR_MCG_R8	0, 1, 2, 3, 4, 6	Unique	**Machine Check R8.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
191H	401	MSR_MCG_R9	0, 1, 2, 3, 4, 6	Unique	**Machine Check R9D/R9.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
192H	402	MSR_MCG_R10	0, 1, 2, 3, 4, 6	Unique	**Machine Check R10.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
193H	403	MSR_MCG_R11	0, 1, 2, 3, 4, 6	Unique	**Machine Check R11.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
194H	404	MSR_MCG_R12	0, 1, 2, 3, 4, 6	Unique	**Machine Check R12.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
195H	405	MSR_MCG_R13	0, 1, 2, 3, 4, 6	Unique	**Machine Check R13.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
196H	406	MSR_MCG_R14	0, 1, 2, 3, 4, 6	Unique	**Machine Check R14.** See Section 15.3.2.6, "IA32_MCG Extended Machine Check State MSRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
197H	407	MSR_MCG_R15	0, 1, 2, 3, 4, 6	Unique	**Machine Check R15.** See Section 15.3.2.6, “IA32_MCG Extended Machine Check State MSRs.”
		63-0			Registers R8-15 (and the associated state-save MSRs) exist only in Intel 64 processors. These registers contain valid information only when the processor is operating in 64-bit mode at the time of the error.
198H	408	IA32_PERF_STATUS	3, 4, 6	Unique	See Table 34-2. See Section 14.1, “Enhanced Intel Speedstep® Technology.”
199H	409	IA32_PERF_CTL	3, 4, 6	Unique	See Table 34-2. See Section 14.1, “Enhanced Intel Speedstep® Technology.”
19AH	410	IA32_CLOCK_MODULATION	0, 1, 2, 3, 4, 6	Unique	**Thermal Monitor Control. (R/W)** See Table 34-2. See Section 14.5.3, “Software Controlled Clock Modulation.”
19BH	411	IA32_THERM_INTERRUPT	0, 1, 2, 3, 4, 6	Unique	**Thermal Interrupt Control. (R/W)** See Section 14.5.2, “Thermal Monitor,” and see Table 34-2.
19CH	412	IA32_THERM_STATUS	0, 1, 2, 3, 4, 6	Shared	**Thermal Monitor Status. (R/W)** See Section 14.5.2, “Thermal Monitor,” and see Table 34-2.
19DH	413	MSR_THERM2_CTL			Thermal Monitor 2 Control.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	**Dec**				
			3,	Shared	For Family F, Model 3 processors: When read, specifies the value of the target TM2 transition last written. When set, it sets the next target value for TM2 transition.
			4, 6	Shared	For Family F, Model 4 and Model 6 processors: When read, specifies the value of the target TM2 transition last written. Writes may cause #GP exceptions.
1A0H	416	IA32_MISC_ENABLE	0, 1, 2, 3, 4, 6	Shared	**Enable Miscellaneous Processor Features. (R/W)**
		0			Fast-Strings Enable. See Table 34-2.
		1			Reserved.
		2			**x87 FPU Fopcode Compatibility Mode Enable.**
		3			**Thermal Monitor 1 Enable.** See Section 14.5.2, "Thermal Monitor," and see Table 34-2.
		4			**Split-Lock Disable.** When set, the bit causes an #AC exception to be issued instead of a split-lock cycle. Operating systems that set this bit must align system structures to avoid split-lock scenarios. When the bit is clear (default), normal split-locks are issued to the bus. This debug feature is specific to the Pentium 4 processor.
		5			Reserved.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	**Dec**				
		6			**Third-Level Cache Disable.** (R/W) When set, the third-level cache is disabled; when clear (default) the third-level cache is enabled. This flag is reserved for processors that do not have a third-level cache. Note that the bit controls only the third-level cache; and only if overall caching is enabled through the CD flag of control register CR0, the page-level cache controls, and/or the MTRRs. See Section 11.5.4, "Disabling and Enabling the L3 Cache."
		7			**Performance Monitoring Available. (R).** See Table 34-2.
		8			**Suppress Lock Enable.** When set, assertion of LOCK on the bus is suppressed during a Split Lock access. When clear (default), LOCK is not suppressed.
		9			**Prefetch Queue Disable.** When set, disables the prefetch queue. When clear (default), enables the prefetch queue.
		10			**FERR# Interrupt Reporting Enable. (R/W)** When set, interrupt reporting through the FERR# pin is enabled; when clear, this interrupt reporting function is disabled.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When this flag is set and the processor is in the stop-clock state (STPCLK# is asserted), asserting the FERR# pin signals to the processor that an interrupt (such as, INIT#, BINIT#, INTR, NMI, SMI#, or RESET#) is pending and that the processor should return to normal operation to handle the interrupt. This flag does not affect the normal operation of the FERR# pin (to indicate an unmasked floating-point error) when the STPCLK# pin is not asserted.
		11			**Branch Trace Storage Unavailable (BTS_UNAVILABLE). (R).** See Table 34-2. When set, the processor does not support branch trace storage (BTS); when clear, BTS is supported.
		12			**PEBS_UNAVILABLE: Precise Event Based Sampling Unavailable. (R).** See Table 34-2. When set, the processor does not support precise event-based sampling (PEBS); when clear, PEBS is supported.
		13	3		**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		17:14			Reserved.
		18	3, 4, 6		**ENABLE MONITOR FSM. (R/W)** See Table 34-2.
		19			**Adjacent Cache Line Prefetch Disable. (R/W)** When set to 1, the processor fetches the cache line of the 128-byte sector containing currently required data. When set to 0, the processor fetches both cache lines in the sector.
					Single processor platforms should not set this bit. Server platforms should set or clear this bit based on platform performance observed in validation and testing. BIOS may contain a setup option that controls the setting of this bit.
		21:20			Reserved.
		22	3, 4, 6		**Limit CPUID MAXVAL. (R/W)** See Table 34-2.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					Setting this can cause unexpected behavior to software that depends on the availability of CPUID leaves greater than 3.
		23		Shared	**xTPR Message Disable. (R/W)** See Table 34-2.
		24			**L1 Data Cache Context Mode. (R/W)** When set, the L1 data cache is placed in shared mode; when clear (default), the cache is placed in adaptive mode. This bit is only enabled for IA-32 processors that support Intel Hyper-Threading Technology. See Section 11.5.6, "L1 Data Cache Context Mode." When L1 is running in adaptive mode and CR3s are identical, data in L1 is shared across logical processors. Otherwise, L1 is not shared and cache use is competitive. If the Context ID feature flag (ECX[10]) is set to 0 after executing CPUID with EAX = 1, the ability to switch modes is not supported. BIOS must not alter the contents of IA32_MISC_ENABLE[24].
		33:25			Reserved.
		34		Unique	**XD Bit Disable. (R/W)** See Table 34-2.
		63:35			Reserved.
1A1H	417	MSR_PLATFORM_BRV	3, 4, 6	Shared	**Platform Feature Requirements. (R)**
		17:0			Reserved.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		18			**PLATFORM Requirements.** When set to 1, indicates the processor has specific platform requirements. The details of the platform requirements are listed in the respective data sheets of the processor.
		63:19			Reserved.
1D7H	471	MSR_LER_FROM_LIP	0, 1, 2, 3, 4, 6	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.8.3, "Last Exception Records."
		31:0			**From Linear IP.** Linear address of the last branch instruction.
		63:32			Reserved.
1D7H	471	63:0		Unique	**From Linear IP.** Linear address of the last branch instruction (If IA-32e mode is active).
1D8H	472	MSR_LER_TO_LIP	0, 1, 2, 3, 4, 6	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.8.3, "Last Exception Records."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		31:0			**From Linear IP.** Linear address of the target of the last branch instruction.
		63:32			Reserved.
1D8H	472	63:0		Unique	**From Linear IP.** Linear address of the target of the last branch instruction (If IA-32e mode is active).
1D9H	473	MSR_DEBUGCTLA	0, 1, 2, 3, 4, 6	Unique	**Debug Control. (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.8.1, “MSR_DEBUGCTLA MSR.”
1DAH	474	MSR_LASTBRANCH _TOS	0, 1, 2, 3, 4, 6	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (0-3 or 0-15) that points to the top of the last branch record stack (that is, that points the index of the MSR containing the most recent branch record). See Section 17.8.2, “LBR Stack for Processors Based on Intel NetBurst® Microarchitecture”; and addresses 1DBH-1DEH and 680H-68FH.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	**Dec**				
1DBH	475	MSR_LASTBRANCH_0	0, 1, 2	Unique	**Last Branch Record 0. (R/W)** One of four last branch record registers on the last branch record stack. It contains pointers to the source and destination instruction for one of the last four branches, exceptions, or interrupts that the processor took. MSR_LASTBRANCH_0 through MSR_LASTBRANCH_3 at 1DBH-1DEH are available only on family 0FH, models 0H-02H. They have been replaced by the MSRs at 680H-68FH and 6C0H-6CFH. See Section 17.8, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."
1DDH	477	MSR_LASTBRANCH_2	0, 1, 2	Unique	**Last Branch Record 2.** See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
1DEH	478	MSR_LASTBRANCH_3	0, 1, 2	Unique	**Last Branch Record 3.** See description of the MSR_LASTBRANCH_0 MSR at 1DBH.
200H	512	IA32_MTRR_PHYS BASE0	0, 1, 2, 3, 4, 6	Shared	**Variable Range Base MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
201H	513	IA32_MTRR_ PHYSMASK0	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
202H	514	IA32_MTRR_ PHYSBASE1	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
203H	515	IA32_MTRR_ PHYSMASK1	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
204H	516	IA32_MTRR_ PHYSBASE2	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
205H	517	IA32_MTRR_ PHYSMASK2	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs".
206H	518	IA32_MTRR_ PHYSBASE3	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
207H	519	IA32_MTRR_ PHYSMASK3	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
208H	520	IA32_MTRR_ PHYSBASE4	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
209H	521	IA32_MTRR_ PHYSMASK4	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
20AH	522	IA32_MTRR_ PHYSBASE5	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
20BH	523	IA32_MTRR_ PHYSMASK5	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
20CH	524	IA32_MTRR_ PHYSBASE6	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
20DH	525	IA32_MTRR_ PHYSMASK6	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
20EH	526	IA32_MTRR_ PHYSBASE7	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
20FH	527	IA32_MTRR_ PHYSMASK7	0, 1, 2, 3, 4, 6	Shared	**Variable Range Mask MTRR.** See Section 11.11.2.3, "Variable Range MTRRs."
250H	592	IA32_MTRR_FIX64K_ 00000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
258H	600	IA32_MTRR_FIX16K_ 80000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
259H	601	IA32_MTRR_FIX16K_ A0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
268H	616	IA32_MTRR_FIX4K_ C0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
269H	617	IA32_MTRR_FIX4K_ C8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs".
26AH	618	IA32_MTRR_FIX4K_ D0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs".
26BH	619	IA32_MTRR_FIX4K_ D8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
26CH	620	IA32_MTRR_FIX4K_ E0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
26DH	621	IA32_MTRR_FIX4K_ E8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
26EH	622	IA32_MTRR_FIX4K_ F0000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
26FH	623	IA32_MTRR_FIX4K_ F8000	0, 1, 2, 3, 4, 6	Shared	**Fixed Range MTRR.** See Section 11.11.2.2, "Fixed Range MTRRs."
277H	631	IA32_PAT	0, 1, 2, 3, 4, 6	Unique	**Page Attribute Table.** See Section 11.11.2.2, "Fixed Range MTRRs."
2FFH	767	IA32_MTRR_DEF_ TYPE	0, 1, 2, 3, 4, 6	Shared	**Default Memory Types. (R/W)** see Table 34-2 See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
300H	768	MSR_BPU_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
301H	769	MSR_BPU_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
302H	770	MSR_BPU_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
303H	771	MSR_BPU_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
304H	772	MSR_MS_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
305H	773	MSR_MS_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
306H	774	MSR_MS_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
307H	775	MSR_MS_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
308H	776	MSR_FLAME_ COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
309H	777	MSR_FLAME_ COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
30AH	778	MSR_FLAME_ COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
30BH	779	MSR_FLAME_ COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
30CH	780	MSR_IQ_COUNTER0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
30DH	781	MSR_IQ_COUNTER1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
30EH	782	MSR_IQ_COUNTER2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
30FH	783	MSR_IQ_COUNTER3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
310H	784	MSR_IQ_COUNTER4	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
311H	785	MSR_IQ_COUNTER5	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.2, "Performance Counters."
360H	864	MSR_BPU_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
361H	865	MSR_BPU_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
362H	866	MSR_BPU_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
363H	867	MSR_BPU_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
364H	868	MSR_MS_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
365H	869	MSR_MS_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
366H	870	MSR_MS_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
367H	871	MSR_MS_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
368H	872	MSR_FLAME_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
369H	873	MSR_FLAME_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
36AH	874	MSR_FLAME_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
36BH	875	MSR_FLAME_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
36CH	876	MSR_IQ_CCCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
36DH	877	MSR_IQ_CCCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
36EH	878	MSR_IQ_CCCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
36FH	879	MSR_IQ_CCCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
370H	880	MSR_IQ_CCCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
371H	881	MSR_IQ_CCCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.3, "CCCR MSRs."
3A0H	928	MSR_BSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A1H	929	MSR_BSU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A2H	930	MSR_FSB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A3H	931	MSR_FSB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A4H	932	MSR_FIRM_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A5H	933	MSR_FIRM_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A6H	934	MSR_FLAME_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A7H	935	MSR_FLAME_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3A8H	936	MSR_DAC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3A9H	937	MSR_DAC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3AAH	938	MSR_MOB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3ABH	939	MSR_MOB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3ACH	940	MSR_PMH_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3ADH	941	MSR_PMH_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3AEH	942	MSR_SAAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3AFH	943	MSR_SAAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B0H	944	MSR_U2L_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B1H	945	MSR_U2L_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B2H	946	MSR_BPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B3H	947	MSR_BPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B4H	948	MSR_IS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B5H	949	MSR_IS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B6H	950	MSR_ITLB_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B7H	951	MSR_ITLB_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3B8H	952	MSR_CRU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3B9H	953	MSR_CRU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3BAH	954	MSR_IQ_ESCR0	0, 1, 2	Shared	See Section 18.9.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BBH	955	MSR_IQ_ESCR1	0, 1, 2	Shared	See Section 18.9.1, "ESCR MSRs." This MSR is not available on later processors. It is only available on processor family 0FH, models 01H-02H.
3BCH	956	MSR_RAT_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3BDH	957	MSR_RAT_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3BEH	958	MSR_SSU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C0H	960	MSR_MS_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C1H	961	MSR_MS_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C2H	962	MSR_TBPU_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C3H	963	MSR_TBPU_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C4H	964	MSR_TC_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C5H	965	MSR_TC_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C8H	968	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."
3C9H	969	MSR_IX_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, "ESCR MSRs."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
3CAH	970	MSR_ALF_ESCR0	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3CBH	971	MSR_ALF_ESCR1	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3CCH	972	MSR_CRU_ESCR2	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3CDH	973	MSR_CRU_ESCR3	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3E0H	992	MSR_CRU_ESCR4	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3E1H	993	MSR_CRU_ESCR5	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3F0H	1008	MSR_TC_PRECISE _EVENT	0, 1, 2, 3, 4, 6	Shared	See Section 18.9.1, “ESCR MSRs.”
3F1H	1009	MSR_PEBS_ENABLE	0, 1, 2, 3, 4, 6	Shared	**Precise Event-Based Sampling (PEBS). (R/W)** Controls the enabling of precise event sampling and replay tagging.
		12:0			See Table 19-18.
		23:13			Reserved.
		24			**UOP Tag.** Enables replay tagging when set.
		25			**ENABLE_PEBS_MY_THR. (R/W)** Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.10.3, “IA32_PEBS_ENABLE MSR,” for an explanation of the target logical processor. This bit is called ENABLE_PEBS in IA-32 processors that do not support Intel Hyper-Threading Technology.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
		26			**ENABLE_PEBS_OTH_THR. (R/W)** Enables PEBS for the target logical processor when set; disables PEBS when clear (default). See Section 18.10.3, “IA32_PEBS_ENABLE MSR,” for an explanation of the target logical processor. This bit is reserved for IA-32 processors that do not support Intel Hyper-Threading Technology.
		63:27			Reserved.
3F2H	1010	MSR_PEBS_MATRIX _VERT	0, 1, 2, 3, 4, 6	Shared	See Table 19-18.
400H	1024	IA32_MC0_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, “IA32_MCi_CTL MSRs.”
401H	1025	IA32_MC0_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, “IA32_MCi_STATUS MSRS.”
402H	1026	IA32_MC0_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, “IA32_MCi_ADDR MSRs.” The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
403H	1027	IA32_MC0_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, “IA32_MCi_MISC MSRs.” The IA32_MC0_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC0_STATUS register is clear.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
407H	1031	IA32_MC1_MISC		Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC1_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
40AH	1034	IA32_MC2_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40BH	1035	IA32_MC2_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	IA32_MC3_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	IA32_MC3_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	IA32_MC3_ADDR	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
40FH	1039	IA32_MC3_MISC	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC3_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	IA32_MC4_CTL	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	IA32_MC4_STATUS	0, 1, 2, 3, 4, 6	Shared	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	IA32_MC4_ADDR			See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	IA32_MC4_MISC			See Section 15.3.2.4, "IA32_MCi_MISC MSRs." The IA32_MC2_MISC MSR is either not implemented or does not contain additional information if the MISCV flag in the IA32_MC4_STATUS register is clear.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
480H	1152	IA32_VMX_BASIC	3, 4, 6	Unique	**Reporting Register of Basic VMX Capabilities. (R/O).** See Table 34-2. See Appendix A.1, "Basic VMX Information."
481H	1153	IA32_VMX_PINBASED_CTLS	3, 4, 6	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O).** See Table 34-2. See Appendix A.3, "VM-Execution Controls."
482H	1154	IA32_VMX_PROCBASED_CTLS	3, 4, 6	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls," and see Table 34-2.
483H	1155	IA32_VMX_EXIT_CTLS	3, 4, 6	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** See Appendix A.4, "VM-Exit Controls," and see Table 34-2.
484H	1156	IA32_VMX_ENTRY_CTLS	3, 4, 6	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** See Appendix A.5, "VM-Entry Controls," and see Table 34-2.
485H	1157	IA32_VMX_MISC	3, 4, 6	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Appendix A.6, "Miscellaneous Data," and see Table 34-2.
486H	1158	IA32_VMX_CR0_FIXED0	3, 4, 6	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 34-2.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
487H	1159	IA32_VMX_CR0_ FIXED1	3, 4, 6	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0," and see Table 34-2.
488H	1160	IA32_VMX_CR4_ FIXED0	3, 4, 6	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 34-2.
489H	1161	IA32_VMX_CR4_ FIXED1	3, 4, 6	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Appendix A.8, "VMX-Fixed Bits in CR4," and see Table 34-2.
48AH	1162	IA32_VMX_VMCS_ ENUM	3, 4, 6	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Appendix A.9, "VMCS Enumeration," and see Table 34-2.
48BH	1163	IA32_VMX_ PROCBASED_CTLS2	3, 4, 6	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls," and see Table 34-2.
600H	1536	IA32_DS_AREA	0, 1, 2, 3, 4, 6	Unique	**DS Save Area. (R/W).** See Table 34-2. See Section 18.9.4, "Debug Store (DS) Mechanism."
680H	1664	MSR_LASTBRANCH _0_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 0. (R/W)** One of 16 pairs of last branch record registers on the last branch record stack (680H-68FH). This part of the stack contains pointers to the source instruction for one of the last 16 branches, exceptions, or interrupts taken by the processor.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
					The MSRs at 680H-68FH, 6C0H-6CfH are not available in processor releases before family 0FH, model 03H. These MSRs replace MSRs previously located at 1DBH-1DEH.which performed the same function for early releases.
					See Section 17.8, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."
681H	1665	MSR_LASTBRANCH _1_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 1.** See description of MSR_LASTBRANCH_0 at 680H.
682H	1666	MSR_LASTBRANCH _2_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 2.** See description of MSR_LASTBRANCH_0 at 680H.
683H	1667	MSR_LASTBRANCH _3_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 3.** See description of MSR_LASTBRANCH_0 at 680H.
684H	1668	MSR_LASTBRANCH _4_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 4.** See description of MSR_LASTBRANCH_0 at 680H.
685H	1669	MSR_LASTBRANCH _5_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 5.** See description of MSR_LASTBRANCH_0 at 680H.
686H	1670	MSR_LASTBRANCH _6_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 6.** See description of MSR_LASTBRANCH_0 at 680H.
687H	1671	MSR_LASTBRANCH _7_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 7.** See description of MSR_LASTBRANCH_0 at 680H.
688H	1672	MSR_LASTBRANCH _8_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 8.** See description of MSR_LASTBRANCH_0 at 680H.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
689H	1673	MSR_LASTBRANCH _9_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 9.** See description of MSR_LASTBRANCH_0 at 680H.
68AH	1674	MSR_LASTBRANCH _10_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 10.** See description of MSR_LASTBRANCH_0 at 680H.
68BH	1675	MSR_LASTBRANCH _11_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 11.** See description of MSR_LASTBRANCH_0 at 680H.
68CH	1676	MSR_LASTBRANCH _12_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 12.** See description of MSR_LASTBRANCH_0 at 680H.
68DH	1677	MSR_LASTBRANCH _13_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 13.** See description of MSR_LASTBRANCH_0 at 680H.
68EH	1678	MSR_LASTBRANCH _14_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 14.** See description of MSR_LASTBRANCH_0 at 680H.
68FH	1679	MSR_LASTBRANCH _15_FROM_LIP	3, 4, 6	Unique	**Last Branch Record 15.** See description of MSR_LASTBRANCH_0 at 680H.
6C0H	1728	MSR_LASTBRANCH _0_TO_LIP	3, 4, 6	Unique	**Last Branch Record 0. (R/W)** One of 16 pairs of last branch record registers on the last branch record stack (6C0H-6CFH). This part of the stack contains pointers to the destination instruction for one of the last 16 branches, exceptions, or interrupts that the processor took. See Section 17.8, "Last Branch, Interrupt, and Exception Recording (Processors based on Intel NetBurst® Microarchitecture)."

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
6C1H	1729	MSR_LASTBRANCH _1_TO_LIP	3, 4, 6	Unique	**Last Branch Record 1.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C2H	1730	MSR_LASTBRANCH _2_TO_LIP	3, 4, 6	Unique	**Last Branch Record 2.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C3H	1731	MSR_LASTBRANCH _3_TO_LIP	3, 4, 6	Unique	**Last Branch Record 3.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C4H	1732	MSR_LASTBRANCH _4_TO_LIP	3, 4, 6	Unique	**Last Branch Record 4.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C5H	1733	MSR_LASTBRANCH _5_TO_LIP	3, 4, 6	Unique	**Last Branch Record 5.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C6H	1734	MSR_LASTBRANCH _6_TO_LIP	3, 4, 6	Unique	**Last Branch Record 6.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C7H	1735	MSR_LASTBRANCH _7_TO_LIP	3, 4, 6	Unique	**Last Branch Record 7.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C8H	1736	MSR_LASTBRANCH _8_TO_LIP	3, 4, 6	Unique	**Last Branch Record 8.** See description of MSR_LASTBRANCH_0 at 6C0H.
6C9H	1737	MSR_LASTBRANCH _9_TO_LIP	3, 4, 6	Unique	**Last Branch Record 9.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CAH	1738	MSR_LASTBRANCH _10_TO_LIP	3, 4, 6	Unique	**Last Branch Record 10.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CBH	1739	MSR_LASTBRANCH _11_TO_LIP	3, 4, 6	Unique	**Last Branch Record 11.** See description of MSR_LASTBRANCH_0 at 6C0H.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				
6CCH	1740	MSR_LASTBRANCH _12_TO_LIP	3, 4, 6	Unique	**Last Branch Record 12.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CDH	1741	MSR_LASTBRANCH _13_TO_LIP	3, 4, 6	Unique	**Last Branch Record 13.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CEH	1742	MSR_LASTBRANCH _14_TO_LIP	3, 4, 6	Unique	**Last Branch Record 14.** See description of MSR_LASTBRANCH_0 at 6C0H.
6CFH	1743	MSR_LASTBRANCH _15_TO_LIP	3, 4, 6	Unique	**Last Branch Record 15.** See description of MSR_LASTBRANCH_0 at 6C0H.
C000_ 0080H		IA32_EFER	3, 4, 6	Unique	**Extended Feature Enables.** See Table 34-2.
C000_ 0081H		IA32_STAR	3, 4, 6	Unique	**System Call Target Address. (R/W)** See Table 34-2.
C000_ 0082H		IA32_LSTAR	3, 4, 6	Unique	**IA-32e Mode System Call Target Address. (R/W)** See Table 34-2.
C000_ 0084H		IA32_FMASK	3, 4, 6	Unique	**System Call Flag Mask. (R/W)** See Table 34-2.
C000_ 0100H		IA32_FS_BASE	3, 4, 6	Unique	**Map of BASE Address of FS. (R/W)** See Table 34-2.
C000_ 0101H		IA32_GS_BASE	3, 4, 6	Unique	**Map of BASE Address of GS. (R/W)** See Table 34-2.
C000_ 0102H		IA32_KERNEL_ GSBASE	3, 4, 6	Unique	**Swap Target of BASE Address of GS. (R/W)** See Table 34-2.

Table 34-13. MSRs in the Pentium 4 and Intel Xeon Processors (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique[1]	Bit Description
Hex	Dec				

NOTES

1. For HT-enabled processors, there may be more than one logical processors per physical unit. If an MSR is Shared, this means that one MSR is shared between logical processors. If an MSR is unique, this means that each logical processor has its own MSR.

34.8.1 MSRs Unique to Intel Xeon Processor MP with L3 Cache

The MSRs listed in Table 34-14 apply to Intel Xeon Processor MP with up to 8MB level three cache. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 3 or 4 (see CPUID instruction for more details).

Table 34-14. MSRs Unique to 64-bit Intel Xeon Processor MP with Up to an 8 MB L3 Cache

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique	Bit Description
107CCH		MSR_IFSB_BUSQ0	3, 4	Shared	**IFSB BUSQ Event Control and Counter Register. (R/W)** See Section 18.14, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."
107CDH		MSR_IFSB_BUSQ1	3, 4	Shared	**IFSB BUSQ Event Control and Counter Register. (R/W)**
107CEH		MSR_IFSB_SNPQ0	3, 4	Shared	**IFSB SNPQ Event Control and Counter Register. (R/W)** See Section 18.14, "Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache."

Table 34-14. MSRs Unique to 64-bit Intel Xeon Processor MP with Up to an 8 MB L3 Cache (Contd.)

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique	Bit Description
107CFH		MSR_IFSB_SNPQ1	3, 4	Shared	**IFSB SNPQ Event Control and Counter Register. (R/W)**
107D0H		MSR_EFSB_DRDY0	3, 4	Shared	**EFSB DRDY Event Control and Counter Register. (R/W)** See Section 18.14, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache” for details.
107D1H		MSR_EFSB_DRDY1	3, 4	Shared	**EFSB DRDY Event Control and Counter Register. (R/W)**
107D2H		MSR_IFSB_CTL6	3, 4	Shared	**IFSB Latency Event Control Register. (R/W)** See Section 18.14, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache” for details.
107D3H		MSR_IFSB_CNTR7	3, 4	Shared	**IFSB Latency Event Counter Register. (R/W)** See Section 18.14, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache.”

The MSRs listed in Table 34-15 apply to Intel Xeon Processor 7100 series. These processors can be detected by enumerating the deterministic cache parameter leaf of CPUID instruction (with EAX = 4 as input) to detect the presence of the third level cache, and with CPUID reporting family encoding 0FH, model encoding 6 (See CPUID instruction for more details.). The performance monitoring MSRs listed in Table 34-15 are shared between logical processors in the same core, but are replicated for each core.

Table 34-15. MSRs Unique to Intel Xeon Processor 7100 Series

Register Address		Register Name Fields and Flags	Model Avail-ability	Shared/ Unique	Bit Description
107CCH		MSR_EMON_L3_CTR_CTL0	6	Shared	**GBUSQ Event Control and Counter Register. (R/W)** See Section 18.14, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache.”
107CDH		MSR_EMON_L3_CTR_CTL1	6	Shared	**GBUSQ Event Control and Counter Register. (R/W)**
107CEH		MSR_EMON_L3_CTR_CTL2	6	Shared	**GSNPQ Event Control and Counter Register. (R/W)** See Section 18.14, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache.”
107CFH		MSR_EMON_L3_CTR_CTL3	6	Shared	**GSNPQ Event Control and Counter Register (R/W)**
107D0H		MSR_EMON_L3_CTR_CTL4	6	Shared	**FSB Event Control and Counter Register. (R/W)** See Section 18.14, “Performance Monitoring on 64-bit Intel Xeon Processor MP with Up to 8-MByte L3 Cache” for details.
107D1H		MSR_EMON_L3_CTR_CTL5	6	Shared	**FSB Event Control and Counter Register. (R/W)**
107D2H		MSR_EMON_L3_CTR_CTL6	6	Shared	**FSB Event Control and Counter Register. (R/W)**
107D3H		MSR_EMON_L3_CTR_CTL7	6	Shared	**FSB Event Control and Counter Register. (R/W)**

34.9 MSRS IN INTEL® CORE™ SOLO AND INTEL® CORE™ DUO PROCESSORS

Model-specific registers (MSRs) for Intel Core Solo, Intel Core Duo processors, and Dual-core Intel Xeon processor LV are listed in Table 34-16. The column "Shared/Unique" applies to Intel Core Duo processor. "Unique" means each processor core has a separate MSR, or a bit field in an MSR governs only a core independently. "Shared" means the MSR or the bit field in an MSR address governs the operation of both processor cores.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
0H	0	P5_MC_ADDR	Unique	See Section 34.12, "MSRs in Pentium Processors," and see Table 34-2.
1H	1	P5_MC_TYPE	Unique	See Section 34.12, "MSRs in Pentium Processors," and see Table 34-2.
6H	6	IA32_MONITOR_ FILTER_SIZE	Unique	See Section 8.10.5, "Monitor/Mwait Address Range Determination," and see Table 34-2.
10H	16	IA32_TIME_ STAMP_COUNTER	Unique	See Section 17.12, "Time-Stamp Counter," and see Table 34-2.
17H	23	IA32_PLATFORM_ ID	Shared	**Platform ID. (R)** See Table 34-2. The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
1BH	27	IA32_APIC_BASE	Unique	See Section 10.4.4, "Local APIC Status and Location," and see Table 34-2.
2AH	42	MSR_EBL_CR_ POWERON	Shared	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0		Reserved.
		1		**Data Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		2		**Response Error Checking Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		3		**MCERR# Drive Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		4		**Address Parity Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		6: 5		Reserved
		7		**BINIT# Driver Enable. (R/W)** 1 = Enabled; 0 = Disabled Note: Not all processor implements R/W.
		8		**Output Tri-state Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		9		**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled
		10		**MCERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		11		Reserved
		12		**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		13		Reserved
		14		**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes
		15		Reserved
		17:16		APIC Cluster ID. (R/O)
		18		**System Bus Frequency. (R/O)** 0 = 100 MHz 1 = Reserved
		19		Reserved.
		21: 20		**Symmetric Arbitration ID. (R/O)**
		26:22		**Clock Frequency Ratio. (R/O)**

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
3AH	58	IA32_FEATURE_ CONTROL	Unique	**Control Features in IA-32 Processor. (R/W)** See Table 34-2.
40H	64	MSR_ LASTBRANCH_0	Unique	**Last Branch Record 0. (R/W)** One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the 'to' address. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."
41H	65	MSR_ LASTBRANCH_1	Unique	**Last Branch Record 1. (R/W)** See description of MSR_LASTBRANCH_0.
42H	66	MSR_ LASTBRANCH_2	Unique	**Last Branch Record 2. (R/W)** See description of MSR_LASTBRANCH_0.
43H	67	MSR_ LASTBRANCH_3	Unique	**Last Branch Record 3. (R/W)** See description of MSR_LASTBRANCH_0.
44H	68	MSR_ LASTBRANCH_4	Unique	**Last Branch Record 4. (R/W)** See description of MSR_LASTBRANCH_0.
45H	69	MSR_ LASTBRANCH_5	Unique	**Last Branch Record 5. (R/W)** See description of MSR_LASTBRANCH_0.
46H	70	MSR_ LASTBRANCH_6	Unique	**Last Branch Record 6. (R/W)** See description of MSR_LASTBRANCH_0.
47H	71	MSR_ LASTBRANCH_7	Unique	**Last Branch Record 7. (R/W)** See description of MSR_LASTBRANCH_0.
79H	121	IA32_BIOS_ UPDT_TRIG	Unique	**BIOS Update Trigger Register (W).** See Table 34-2.
8BH	139	IA32_BIOS_ SIGN_ID	Unique	**BIOS Update Signature ID (RO).** See Table 34-2.
C1H	193	IA32_PMC0	Unique	**Performance counter register.** See Table 34-2.
C2H	194	IA32_PMC1	Unique	**Performance counter register.** See Table 34-2.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
CDH	205	MSR_FSB_FREQ	Shared	**Scaleable Bus Speed. (RO)** This field indicates the scaleable bus clock speed:
		2:0		▪ 101B: 100 MHz (FSB 400) ▪ 001B: 133 MHz (FSB 533) ▪ 011B: 167 MHz (FSB 667) 133.33 MHz should be utilized if performing calculation with System Bus Speed when encoding is 101B. 166.67 MHz should be utilized if performing calculation with System Bus Speed when encoding is 001B.
		63:3		Reserved.
E7H	231	IA32_MPERF	Unique	**Maximum Performance Frequency Clock Count. (RW).** See Table 34-2.
E8H	232	IA32_APERF	Unique	**Actual Performance Frequency Clock Count. (RW).** See Table 34-2.
FEH	254	IA32_MTRRCAP	Unique	See Table 34-2.
11EH	281	MSR_BBL_CR_CTL3	Shared	
		0		**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		7:1		Reserved.
		8		**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9		Reserved.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		23		**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24		Reserved.
174H	372	IA32_SYSENTER_CS	Unique	See Table 34-2.
175H	373	IA32_SYSENTER_ESP	Unique	See Table 34-2.
176H	374	IA32_SYSENTER_EIP	Unique	See Table 34-2.
179H	377	IA32_MCG_CAP	Unique	See Table 34-2.
17AH	378	IA32_MCG_STATUS	Unique	
		0		**RIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1		**EIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.
		2		**MCIP.** When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		63:3		Reserved.
186H	390	IA32_ PERFEVTSEL0	Unique	See Table 34-2 .
187H	391	IA32_ PERFEVTSEL1	Unique	See Table 34-2.
198H	408	IA32_PERF_STAT US	Shared	See Table 34-2.
199H	409	IA32_PERF_CTL	Unique	See Table 34-2.
19AH	410	IA32_CLOCK_ MODULATION	Unique	**Clock Modulation. (R/W)** See Table 34-2.
19BH	411	IA32_THERM_ INTERRUPT	Unique	**Thermal Interrupt Control. (R/W)** See Table 34-2 . See Section 14.5.2, "Thermal Monitor."
19CH	412	IA32_THERM_ STATUS	Unique	**Thermal Monitor Status. (R/W)** See Table 34-2. See Section 14.5.2, "Thermal Monitor".
19DH	413	MSR_THERM2_ CTL	Unique	
		15:0		Reserved.
		16		**TM_SELECT. (R/W)** Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16		Reserved.
1A0	416	IA32_MISC_ ENABLE		**Enable Miscellaneous Processor Features.** (R/W) Allows a variety of processor functions to be enabled and disabled.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
		2:0		Reserved.
		3	Unique	**Automatic Thermal Control Circuit Enable. (R/W)** See Table 34-2.
		6:4		Reserved.
		7	Shared	**Performance Monitoring Available. (R).** See Table 34-2.
		9:8		Reserved.
		10	Shared	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
		11	Shared	**Branch Trace Storage Unavailable. (RO).** See Table 34-2.
		12		Reserved.
		13	Shared	**TM2 Enable. (R/W)** When this bit is set (1) and the thermal sensor indicates that the die temperature is at the pre-determined threshold, the Thermal Monitor 2 mechanism is engaged. TM2 will reduce the bus to core ratio and voltage according to the value last written to MSR_THERM2_CTL bits 15:0.

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
				When this bit is clear (0, default), the processor does not change the VID signals or the bus to core ratio when the processor enters a thermal managed state. If the TM2 feature flag (ECX[8]) is not set to 1 after executing CPUID with EAX = 1, then this feature is not supported and BIOS must not alter the contents of this bit location. The processor is operating out of spec if both this bit and the TM1 bit are set to disabled states.
		15:14		Reserved.
		16	Shared	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 1 = Enhanced Intel SpeedStep Technology enabled
		18	Shared	**ENABLE MONITOR FSM. (R/W)** See Table 34-2.
		19		**Reserved.**
		22	Shared	**Limit CPUID Maxval. (R/W)** See Table 34-2. Setting this bit may cause behavior in software that depends on the availability of CPUID leaves greater than 3.
		33:23		Reserved.
		34	Shared	**XD Bit Disable. (R/W)** See Table 34-2.
		63:35		Reserved.
1C9H	457	MSR_ LASTBRANCH_ TOS	Unique	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See MSR_LASTBRANCH_0_FROM_IP (at 40H).

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
1D9H	473	IA32_DEBUGCTL	Unique	**Debug Control. (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section.
1DDH	477	MSR_LER_FROM_ LIP	Unique	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1DEH	478	MSR_LER_TO_LIP	Unique	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled.
1E0H	480	ROB_CR_ BKUPTMPDR6	Unique	
		1:0		Reserved.
		2		Fast String Enable bit. (Default, enabled)
200H	512	MTRRphysBase0	Unique	
201H	513	MTRRphysMask0	Unique	
202H	514	MTRRphysBase1	Unique	
203H	515	MTRRphysMask1	Unique	
204H	516	MTRRphysBase2	Unique	
205H	517	MTRRphysMask2	Unique	
206H	518	MTRRphysBase3	Unique	
207H	519	MTRRphysMask3	Unique	
208H	520	MTRRphysBase4	Unique	
209H	521	MTRRphysMask4	Unique	
20AH	522	MTRRphysBase5	Unique	
20BH	523	MTRRphysMask5	Unique	

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
20CH	524	MTRRphysBase6	Unique	
20DH	525	MTRRphysMask6	Unique	
20EH	526	MTRRphysBase7	Unique	
20FH	527	MTRRphysMask7	Unique	
250H	592	MTRRfix64K_ 00000	Unique	
258H	600	MTRRfix16K_ 80000	Unique	
259H	601	MTRRfix16K_ A0000	Unique	
268H	616	MTRRfix4K_ C0000	Unique	
269H	617	MTRRfix4K_ C8000	Unique	
26AH	618	MTRRfix4K_ D0000	Unique	
26BH	619	MTRRfix4K_ D8000	Unique	
26CH	620	MTRRfix4K_ E0000	Unique	
26DH	621	MTRRfix4K_ E8000	Unique	
26EH	622	MTRRfix4K_ F0000	Unique	
26FH	623	MTRRfix4K_ F8000	Unique	
2FFH	767	IA32_MTRR_DEF_ TYPE	Unique	**Default Memory Types. (R/W). see Table 34-2.** See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MC0_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
401H	1025	IA32_MC0_ STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_ STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
406H	1030	IA32_MC1_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_ STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	Unique	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_ STATUS	Unique	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
40EH	1038	MSR_MC4_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL		See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_ STATUS		See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	Unique	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
413H	1043	MSR_MC3_MISC	Unique	
414H	1044	MSR_MC5_CTL	Unique	
415H	1045	MSR_MC5_ STATUS	Unique	
416H	1046	MSR_MC5_ADDR	Unique	
417H	1047	MSR_MC5_MISC	Unique	
480H	1152	IA32_VMX_BASIC	Unique	**Reporting Register of Basic VMX Capabilities. (R/O).** See Table 34-2. See Appendix A.1, "Basic VMX Information" (If CPUID.01H:ECX.[bit 9])
481H	1153	IA32_VMX_PINBA SED_CTLS	Unique	**Capability Reporting Register of Pin-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9])

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
482H	1154	IA32_VMX_PROCBASED_CTLS	Unique	**Capability Reporting Register of Primary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9])
483H	1155	IA32_VMX_EXIT_CTLS	Unique	**Capability Reporting Register of VM-exit Controls. (R/O)** See Appendix A.4, "VM-Exit Controls" (If CPUID.01H:ECX.[bit 9])
484H	1156	IA32_VMX_ENTRY_CTLS	Unique	**Capability Reporting Register of VM-entry Controls. (R/O)** See Appendix A.5, "VM-Entry Controls" (If CPUID.01H:ECX.[bit 9])
485H	1157	IA32_VMX_MISC	Unique	**Reporting Register of Miscellaneous VMX Capabilities. (R/O)** See Appendix A.6, "Miscellaneous Data" (If CPUID.01H:ECX.[bit 9])
486H	1158	IA32_VMX_CR0_FIXED0	Unique	**Capability Reporting Register of CR0 Bits Fixed to 0. (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0" (If CPUID.01H:ECX.[bit 9])
487H	1159	IA32_VMX_CR0_FIXED1	Unique	**Capability Reporting Register of CR0 Bits Fixed to 1. (R/O)** See Appendix A.7, "VMX-Fixed Bits in CR0" (If CPUID.01H:ECX.[bit 9])
488H	1160	IA32_VMX_CR4_FIXED0	Unique	**Capability Reporting Register of CR4 Bits Fixed to 0. (R/O)** See Appendix A.8, "VMX-Fixed Bits in CR4" (If CPUID.01H:ECX.[bit 9])
489H	1161	IA32_VMX_CR4_FIXED1	Unique	**Capability Reporting Register of CR4 Bits Fixed to 1. (R/O)** See Appendix A.8, "VMX-Fixed Bits in CR4" (If CPUID.01H:ECX.[bit 9])

Table 34-16. MSRs in Intel Core Solo, Intel Core Duo Processors, and Dual-Core Intel Xeon Processor LV (Contd.)

Register Address		Register Name	Shared/ Unique	Bit Description
Hex	Dec			
48AH	1162	IA32_VMX_ VMCS_ENUM	Unique	**Capability Reporting Register of VMCS Field Enumeration. (R/O).** See Appendix A.9, "VMCS Enumeration" (If CPUID.01H:ECX.[bit 9])
48BH	1163	IA32_VMX_PROCB ASED_CTLS2	Unique	**Capability Reporting Register of Secondary Processor-based VM-execution Controls. (R/O)** See Appendix A.3, "VM-Execution Controls" (If CPUID.01H:ECX.[bit 9] and IA32_VMX_PROCBASED_CTLS[bit 63])
600H	1536	IA32_DS_AREA	Unique	**DS Save Area. (R/W)** See Table 34-2. See Section 18.9.4, "Debug Store (DS) Mechanism."
		31:0		**DS Buffer Management Area.** Linear address of the first byte of the DS buffer management area.
		63:32		Reserved.
C000_ 0080H		IA32_EFER	Unique	See Table 34-2.
		10:0		Reserved.
		11		**Execute Disable Bit Enable.**
		63:12		Reserved

34.10 MSRS IN THE PENTIUM M PROCESSOR

Model-specific registers (MSRs) for the Pentium M processor are similar to those described in Section 34.11 for P6 family processors. The following table describes new MSRs and MSRs whose behavior has changed on the Pentium M processor.

Table 34-17. MSRs in Pentium M Processors

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 34.12, “MSRs in Pentium Processors.”
1H	1	P5_MC_TYPE	See Section 34.12, “MSRs in Pentium Processors.”
10H	16	IA32_TIME_STAMP_ COUNTER	See Section 17.12, “Time-Stamp Counter,” and see Table 34-2.
17H	23	IA32_PLATFORM_ID	**Platform ID. (R).** See Table 34-2. The operating system can use this MSR to determine “slot” information for the processor and the proper microcode update to load.
2AH	42	MSR_EBL_CR_POWERON	**Processor Hard Power-On Configuration.** (R/W) Enables and disables processor features. (R) Indicates current processor configuration.
		0	Reserved.
		1	**Data Error Checking Enable. (R)** 0 = Disabled Always 0 on the Pentium M processor.
		2	**Response Error Checking Enable.** (R) 0 = Disabled Always 0 on the Pentium M processor.
		3	**MCERR# Drive Enable. (R)** 0 = Disabled Always 0 on the Pentium M processor.
		4	**Address Parity Enable. (R)** 0 = Disabled Always 0 on the Pentium M processor.
		6:5	Reserved.
		7	**BINIT# Driver Enable. (R)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		8	**Output Tri-state Enabled. (R/O)** 1 = Enabled; 0 = Disabled
		9	**Execute BIST. (R/O)** 1 = Enabled; 0 = Disabled

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		10	**MCERR# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		11	Reserved.
		12	**BINIT# Observation Enabled. (R/O)** 1 = Enabled; 0 = Disabled Always 0 on the Pentium M processor.
		13	**Reserved**
		14	**1 MByte Power on Reset Vector. (R/O)** 1 = 1 MByte; 0 = 4 GBytes Always 0 on the Pentium M processor.
		15	Reserved.
		17:16	APIC Cluster ID. (R/O) Always 00B on the Pentium M processor.
		18	**System Bus Frequency. (R/O)** 0 = 100 MHz 1 = Reserved Always 0 on the Pentium M processor.
		19	Reserved.
		21: 20	Symmetric Arbitration ID. (R/O) Always 00B on the Pentium M processor.
		26:22	Clock Frequency Ratio (R/O)
40H	64	MSR_LASTBRANCH_0	**Last Branch Record 0. (R/W)** One of 8 last branch record registers on the last branch record stack: bits 31-0 hold the 'from' address and bits 63-32 hold the to address. See also: ▪ Last Branch Record Stack TOS at 1C9H ▪ Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"
41H	65	MSR_LASTBRANCH_1	**Last Branch Record 1. (R/W)** See description of MSR_LASTBRANCH_0.

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
42H	66	MSR_LASTBRANCH_2	**Last Branch Record 2. (R/W)** See description of MSR_LASTBRANCH_0.
43H	67	MSR_LASTBRANCH_3	**Last Branch Record 3. (R/W)** See description of MSR_LASTBRANCH_0.
44H	68	MSR_LASTBRANCH_4	**Last Branch Record 4. (R/W)** See description of MSR_LASTBRANCH_0.
45H	69	MSR_LASTBRANCH_5	**Last Branch Record 5. (R/W)** See description of MSR_LASTBRANCH_0.
46H	70	MSR_LASTBRANCH_6	**Last Branch Record 6. (R/W)** See description of MSR_LASTBRANCH_0.
47H	71	MSR_LASTBRANCH_7	**Last Branch Record 7. (R/W)** See description of MSR_LASTBRANCH_0.
119H	281	MSR_BBL_CR_CTL	
		63:0	Reserved.
11EH	281	MSR_BBL_CR_CTL3	
		0	**L2 Hardware Enabled. (RO)** 1 = If the L2 is hardware-enabled 0 = Indicates if the L2 is hardware-disabled
		4:1	Reserved.
		5	**ECC Check Enable. (RO)** This bit enables ECC checking on the cache data bus. ECC is always generated on write cycles. 0 = Disabled (default) 1 = Enabled For the Pentium M processor, ECC checking on the cache data bus is always enabled.
		7:6	Reserved.

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		8	**L2 Enabled. (R/W)** 1 = L2 cache has been initialized 0 = Disabled (default) Until this bit is set the processor will not respond to the WBINVD instruction or the assertion of the FLUSH# input.
		22:9	Reserved.
		23	**L2 Not Present. (RO)** 0 = L2 Present 1 = L2 Not Present
		63:24	Reserved.
179H	377	IA32_MCG_CAP	
		7:0	**Count. (RO)** Indicates the number of hardware unit error reporting banks available in the processor.
		8	**IA32_MCG_CTL Present. (RO)** 1 = Indicates that the processor implements the MSR_MCG_CTL register found at MSR 17BH. 0 = Not supported.
		63:9	Reserved.
17AH	378	IA32_MCG_STATUS	
		0	**RIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) can be used to restart the program. If this bit is cleared, the program cannot be reliably restarted.
		1	**EIPV.** When set, this bit indicates that the instruction addressed by the instruction pointer pushed on the stack (when the machine check was generated) is directly associated with the error.

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2	**MCIP.** When set, this bit indicates that a machine check has been generated. If a second machine check is detected while this bit is still set, the processor enters a shutdown state. Software should write this bit to 0 after processing a machine check exception.
		63:3	Reserved.
198H	408	IA32_PERF_STATUS	See Table 34-2.
199H	409	IA32_PERF_CTL	See Table 34-2.
19AH	410	IA32_CLOCK_ MODULATION	**Clock Modulation. (R/W).** See Table 34-2. See Section 14.5.3, “Software Controlled Clock Modulation.”
19BH	411	IA32_THERM_ INTERRUPT	**Thermal Interrupt Control. (R/W).** See Table 34-2. See Section 14.5.2, “Thermal Monitor.”
19CH	412	IA32_THERM_ STATUS	**Thermal Monitor Status. (R/W).** See Table 34-2. See Section 14.5.2, “Thermal Monitor.”
19DH	413	MSR_THERM2_CTL	
		15:0	Reserved.
		16	**TM_SELECT.** (R/W) Mode of automatic thermal monitor: 0 = Thermal Monitor 1 (thermally-initiated on-die modulation of the stop-clock duty cycle) 1 = Thermal Monitor 2 (thermally-initiated frequency transitions) If bit 3 of the IA32_MISC_ENABLE register is cleared, TM_SELECT has no effect. Neither TM1 nor TM2 will be enabled.
		63:16	Reserved.
1A0	416	IA32_MISC_ENABLE	**Enable Miscellaneous Processor Features. (R/W)** Allows a variety of processor functions to be enabled and disabled.

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2:0	Reserved.
		3	**Automatic Thermal Control Circuit Enable. (R/W)** 1 = Setting this bit enables the thermal control circuit (TCC) portion of the Intel Thermal Monitor feature. This allows processor clocks to be automatically modulated based on the processor's thermal sensor operation. 0 = Disabled (default). The automatic thermal control circuit enable bit determines if the thermal control circuit (TCC) will be activated when the processor's internal thermal sensor determines the processor is about to exceed its maximum operating temperature. When the TCC is activated and TM1 is enabled, the processors clocks will be forced to a 50% duty cycle. BIOS must enable this feature. The bit should not be confused with the on-demand thermal control circuit enable bit.
		6:4	Reserved.
		7	**Performance Monitoring Available. (R)** 1 = Performance monitoring enabled 0 = Performance monitoring disabled
		9:8	Reserved.
		10	**FERR# Multiplexing Enable. (R/W)** 1 = FERR# asserted by the processor to indicate a pending break event within the processor 0 = Indicates compatible FERR# signaling behavior This bit must be set to 1 to support XAPIC interrupt model usage.
			Branch Trace Storage Unavailable. (RO) 1 = Processor doesn't support branch trace storage (BTS) 0 = BTS is supported

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		12	**Precise Event Based Sampling Unavailable. (RO)** 1 = Processor does not support precise event-based sampling (PEBS); 0 = PEBS is supported. The Pentium M processor does not support PEBS.
		15:13	Reserved.
		16	**Enhanced Intel SpeedStep Technology Enable. (R/W)** 1 = Enhanced Intel SpeedStep Technology enabled. On the Pentium M processor, this bit may be configured to be read-only.
		22:17	Reserved.
		23	**xTPR Message Disable. (R/W)** When set to 1, xTPR messages are disabled. xTPR messages are optional messages that allow the processor to inform the chipset of its priority. The default is processor specific.
		63:24	Reserved.
1C9H	457	MSR_LASTBRANCH_TOS	**Last Branch Record Stack TOS. (R)** Contains an index (bits 0-3) that points to the MSR containing the most recent branch record. See also: ▪ MSR_LASTBRANCH_0_FROM_IP (at 40H) ▪ Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)"
1D9H	473	MSR_DEBUGCTLB	**Debug Control. (R/W)** Controls how several debug features are used. Bit definitions are discussed in the referenced section. See Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)."

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
1DDH	477	MSR_LER_TO_LIP	**Last Exception Record To Linear IP. (R)** This area contains a pointer to the target of the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.11.2, "Last Branch and Last Exception MSRs."
1DEH	478	MSR_LER_FROM_LIP	**Last Exception Record From Linear IP. (R)** Contains a pointer to the last branch instruction that the processor executed prior to the last exception that was generated or the last interrupt that was handled. See Section 17.10, "Last Branch, Interrupt, and Exception Recording (Pentium M Processors)" and Section 17.11.2, "Last Branch and Last Exception MSRs."
2FFH	767	IA32_MTRR_DEF_ TYPE	**Default Memory Types. (R/W)** Sets the memory type for the regions of physical memory that are not mapped by the MTRRs. See Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."
400H	1024	IA32_MC0_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
401H	1025	IA32_MC0_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
402H	1026	IA32_MC0_ADDR	See Section 14.3.2.3., "IA32_MCi_ADDR MSRs". The IA32_MC0_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC0_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
404H	1028	IA32_MC1_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
405H	1029	IA32_MC1_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
406H	1030	IA32_MC1_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC1_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC1_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
408H	1032	IA32_MC2_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
409H	1033	IA32_MC2_STATUS	See Chapter 15.3.2.2, "IA32_MCi_STATUS MSRS."
40AH	1034	IA32_MC2_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The IA32_MC2_ADDR register is either not implemented or contains no address if the ADDRV flag in the IA32_MC2_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
40CH	1036	MSR_MC4_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
40DH	1037	MSR_MC4_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
40EH	1038	MSR_MC4_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC4_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC4_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.
410H	1040	MSR_MC3_CTL	See Section 15.3.2.1, "IA32_MCi_CTL MSRs."
411H	1041	MSR_MC3_STATUS	See Section 15.3.2.2, "IA32_MCi_STATUS MSRS."
412H	1042	MSR_MC3_ADDR	See Section 15.3.2.3, "IA32_MCi_ADDR MSRs." The MSR_MC3_ADDR register is either not implemented or contains no address if the ADDRV flag in the MSR_MC3_STATUS register is clear. When not implemented in the processor, all reads and writes to this MSR will cause a general-protection exception.

Table 34-17. MSRs in Pentium M Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
600H	1536	IA32_DS_AREA	**DS Save Area. (R/W).** See Table 34-2. Points to the DS buffer management area, which is used to manage the BTS and PEBS buffers. See Section 18.9.4, "Debug Store (DS) Mechanism."
		31:0	**DS Buffer Management Area.** Linear address of the first byte of the DS buffer management area.
		63:32	Reserved.

34.11 MSRS IN THE P6 FAMILY PROCESSORS

The following MSRs are defined for the P6 family processors. The MSRs in this table that are shaded are available only in the Pentium II and Pentium III processors. Beginning with the Pentium 4 processor, some of the MSRs in this list have been designated as "architectural" and have had their names changed. See Table 34-2 for a list of the architectural MSRs.

Table 34-18. MSRs in the P6 Family Processors

Register Address		Register Name	Bit Description
Hex	Dec		
0H	0	P5_MC_ADDR	See Section 34.12, "MSRs in Pentium Processors."
1H	1	P5_MC_TYPE	See Section 34.12, "MSRs in Pentium Processors."
10H	16	TSC	See Section 17.12, "Time-Stamp Counter."
17H	23	IA32_PLATFORM_ID	**Platform ID. (R)** The operating system can use this MSR to determine "slot" information for the processor and the proper microcode update to load.
		49:0	Reserved.

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		52:50	**Platform Id. (R)** Contains information concerning the intended platform for the processor. 52 51 50 0 0 0 Processor Flag 0 0 0 1 Processor Flag 1 0 1 0 Processor Flag 2 0 1 1 Processor Flag 3 1 0 0 Processor Flag 4 1 0 1 Processor Flag 5 1 1 0 Processor Flag 6 1 1 1 Processor Flag 7
		56:53	L2 Cache Latency Read.
		59:57	Reserved.
		60	Clock Frequency Ratio Read.
		63:61	**Reserved.**
1BH	27	APIC_BASE	Section 10.4.4, "Local APIC Status and Location."
		7:0	Reserved.
		8	**Boot Strap Processor indicator Bit.** 1 = BSP
		10:9	Reserved.
		11	**APIC Global Enable Bit - Permanent till reset.** 1 = Enabled 0 = Disabled
		31:12	APIC Base Address.
		63:32	Reserved.
2AH	42	EBL_CR_POWERON	**Processor Hard Power-On Configuration. (R/W)** Enables and disables processor features; (R) indicates current processor configuration.
		0	Reserved.[1]
		1	**Data Error Checking Enable. (R/W)** 1 = Enabled 0 = Disabled

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2	**Response Error Checking Enable FRCERR Observation Enable. (R/W)** 1 = Enabled 0 = Disabled
		3	**AERR# Drive Enable. (R/W)** 1 = Enabled 0 = Disabled
		4	**BERR# Enable for Initiator Bus Requests. (R/W)** 1 = Enabled 0 = Disabled
		5	Reserved.
		6	**BERR# Driver Enable for Initiator Internal Errors. (R/W)** 1 = Enabled 0 = Disabled
		7	**BINIT# Driver Enable. (R/W)** 1 = Enabled 0 = Disabled
		8	**Output Tri-state Enabled. (R)** 1 = Enabled 0 = Disabled
		9	**Execute BIST. (R)** 1 = Enabled 0 = Disabled
		10	**AERR# Observation Enabled. (R)** 1 = Enabled 0 = Disabled
		11	Reserved.
		12	**BINIT# Observation Enabled. (R)** 1 = Enabled 0 = Disabled

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		13	**In Order Queue Depth. (R)** 1 = 1 0 = 8
		14	**1-MByte Power on Reset Vector. (R)** 1 = 1MByte 0 = 4GBytes
		15	**FRC Mode Enable. (R)** 1 = Enabled 0 = Disabled
		17:16	**APIC Cluster ID. (R)**
		19:18	**System Bus Frequency. (R)** 00 = 66MHz 10 = 100Mhz 01 = 133MHz 11 = Reserved
		21: 20	**Symmetric Arbitration ID. (R)**
		25:22	**Clock Frequency Ratio. (R)**
		26	**Low Power Mode Enable. (R/W)**
		27	**Clock Frequency Ratio.**
		63:28	Reserved.[1]
33H	51	TEST_CTL	**Test Control Register.**
		29:0	Reserved.
		30	**Streaming Buffer Disable.**
		31	**Disable LOCK#.** Assertion for split locked access.
79H	121	BIOS_UPDT_TRIG	BIOS Update Trigger Register.
88	136	BBL_CR_D0[63:0]	Chunk 0 data register D[63:0]: used to write to and read from the L2
89	137	BBL_CR_D1[63:0]	Chunk 1 data register D[63:0]: used to write to and read from the L2
8A	138	BBL_CR_D2[63:0]	Chunk 2 data register D[63:0]: used to write to and read from the L2

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
8BH	139	BIOS_SIGN/BBL_CR_D3[63:0]	**BIOS Update Signature Register or Chunk 3 data register D[63:0].** Used to write to and read from the L2 depending on the usage model.
C1H	193	PerfCtr0 (PERFCTR0)	
C2H	194	PerfCtr1 (PERFCTR1)	
FEH	254	MTRRcap	
116	278	BBL_CR_ADDR [63:0]	Address register: used to send specified address (A31-A3) to L2 during cache initialization accesses.
		BBL_CR_ADDR [63:32]	Reserved,
		BBL_CR_ADDR [31:3]	Address bits [35:3]
		BBL_CR_ADDR [2:0]	Reserved Set to 0.
118	280	BBL_CR_DECC[63:0]	Data ECC register D[7:0]: used to write ECC and read ECC to/from L2
119	281	BBL_CR_CTL	Control register: used to program L2 commands to be issued via cache configuration accesses mechanism. Also receives L2 lookup response
		BL_CR_CTL[63:22]	Reserved

		BBL_CR_CTL[21]	Processor number[2]
			Disable = 1
			Enable = 0
			Reserved

		BBL_CR_CTL[20:19]	User supplied ECC
		BBL_CR_CTL[18]	Reserved
		BBL_CR_CTL[17]	L2 Hit
		BBL_CR_CTL[16]	Reserved
		BBL_CR_CTL[15:14]	State from L2
		BBL_CR_CTL[13:12]	Modified - 11,Exclusive - 10, Shared - 01, Invalid - 00
		BBL_CR_CTL[11:10]	Way from L2 Way 0 - 00, Way 1 - 01, Way 2 - 10, Way 3 - 11
		BBL_CR_CTL[9:8]	Way to L2
		BBL_CR_CTL[7]	Reserved
		BBL_CR_CTL[6:5]	State to L2

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		BBL_CR_CTL[4:0] 01100 01110 01111 00010 00011 010 + MESI encode 111 + MESI encode 100 + MESI encode	L2 Command Data Read w/ LRU update (RLU) Tag Read w/ Data Read (TRR) Tag Inquire (TI) L2 Control Register Read (CR) L2 Control Register Write (CW) Tag Write w/ Data Read (TWR) Tag Write w/ Data Write (TWW) Tag Write (TW)
11A	282	BBL_CR_TRIG	Trigger register: used to initiate a cache configuration accesses access, Write only with Data = 0.
11B	283	BBL_CR_BUSY	Busy register: indicates when a cache configuration accesses L2 command is in progress. D[0] = 1 = BUSY
11E	286	BBL_CR_CTL3	Control register 3: used to configure the L2 Cache
		BBL_CR_CTL3[63:26]	Reserved
		BBL_CR_CTL3[25]	Cache bus fraction (read only)
		BBL_CR_CTL3[24]	Reserved
		BBL_CR_CTL3[23]	L2 Hardware Disable (read only)
		BBL_CR_CTL3[22:20] 111 110 101 100 011 010 001 000	L2 Physical Address Range support 64GBytes 32GBytes 16GBytes 8GBytes 4GBytes 2GBytes 1GBytes 512MBytes
		BBL_CR_CTL3[19]	Reserved
		BBL_CR_CTL3[18]	Cache State error checking enable (read/write)

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		BBL_CR_CTL3[17:13 00001 00010 00100 01000 10000	Cache size per bank (read/write) 256KBytes 512KBytes 1MByte 2MByte 4MBytes
		BBL_CR_CTL3[12:11]	Number of L2 banks (read only)
		BBL_CR_CTL3[10:9] 00 01 10 11	L2 Associativity (read only) Direct Mapped 2 Way 4 Way Reserved
		BBL_CR_CTL3[8]	L2 Enabled (read/write)
		BBL_CR_CTL3[7]	CRTN Parity Check Enable (read/write)
		BBL_CR_CTL3[6]	Address Parity Check Enable (read/write)
		BBL_CR_CTL3[5]	ECC Check Enable (read/write)
		BBL_CR_CTL3[4:1]	L2 Cache Latency (read/write)
		BBL_CR_CTL3[0]	L2 Configured (read/write)
174H	372	SYSENTER_CS_MSR	CS register target for CPL 0 code
175H	373	SYSENTER_ESP_MSR	Stack pointer for CPL 0 stack
176H	374	SYSENTER_EIP_MSR	CPL 0 code entry point
179H	377	MCG_CAP	
17AH	378	MCG_STATUS	
17BH	379	MCG_CTL	
186H	390	PerfEvtSel0 (EVNTSEL0)	
		7:0	**Event Select.** Refer to Performance Counter section for a list of event encodings.
		15:8	**UMASK (Unit Mask).** Unit mask register set to 0 to enable all count options.

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		16	**USER.** Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	**OS.** Controls the counting of events at Privilege level of 0.
		18	**E.** Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	**PC.** Enabled the signaling of performance counter overflow via BP0 pin
		20	**INT.** Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		22	**ENABLE.** Enables the counting of performance events in both counters 1 = Enable 0 = Disable
		23	**INV.** Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	CMASK (Counter Mask).

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
187H	391	PerfEvtSel1 (EVNTSEL1)	
		7:0	**Event Select.** Refer to Performance Counter section for a list of event encodings.
		15:8	**UMASK (Unit Mask).** Unit mask register set to 0 to enable all count options.
		16	**USER.** Controls the counting of events at Privilege levels of 1, 2, and 3.
		17	**OS.** Controls the counting of events at Privilege level of 0
		18	**E.** Occurrence/Duration Mode Select 1 = Occurrence 0 = Duration
		19	**PC.** Enabled the signaling of performance counter overflow via BP0 pin.
		20	**INT.** Enables the signaling of counter overflow via input to APIC 1 = Enable 0 = Disable
		23	**INV.** Inverts the result of the CMASK condition 1 = Inverted 0 = Non-Inverted
		31:24	**CMASK (Counter Mask).**
1D9H	473	DEBUGCTLMSR	
		0	Enable/Disable Last Branch Records
		1	Branch Trap Flag

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
		2	Performance Monitoring/Break Point Pins
		3	Performance Monitoring/Break Point Pins
		4	Performance Monitoring/Break Point Pins
		5	Performance Monitoring/Break Point Pins
		6	Enable/Disable Execution Trace Messages
		31:7	Reserved
1DBH	475	LASTBRANCHFROMIP	
1DCH	476	LASTBRANCHTOIP	
1DDH	477	LASTINTFROMIP	
1DEH	478	LASTINTTOIP	
1E0H	480	ROB_CR_BKUPTMPDR6	
		1:0	Reserved
		2	Fast String Enable bit. Default is enabled
200H	512	MTRRphysBase0	
201H	513	MTRRphysMask0	
202H	514	MTRRphysBase1	
203H	515	MTRRphysMask1	
204H	516	MTRRphysBase2	
205H	517	MTRRphysMask2	
206H	518	MTRRphysBase3	
207H	519	MTRRphysMask3	
208H	520	MTRRphysBase4	
209H	521	MTRRphysMask4	
20AH	522	MTRRphysBase5	
20BH	523	MTRRphysMask5	
20CH	524	MTRRphysBase6	
20DH	525	MTRRphysMask6	
20EH	526	MTRRphysBase7	
20FH	527	MTRRphysMask7	

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
250H	592	MTRRfix64K_00000	
258H	600	MTRRfix16K_80000	
259H	601	MTRRfix16K_A0000	
268H	616	MTRRfix4K_C0000	
269H	617	MTRRfix4K_C8000	
26AH	618	MTRRfix4K_D0000	
26BH	619	MTRRfix4K_D8000	
26CH	620	MTRRfix4K_E0000	
26DH	621	MTRRfix4K_E8000	
26EH	622	MTRRfix4K_F0000	
26FH	623	MTRRfix4K_F8000	
2FFH	767	MTRRdefType	
		2:0	Default memory type
		10	Fixed MTRR enable
		11	MTRR Enable
400H	1024	MC0_CTL	
401H	1025	MC0_STATUS	
		15:0	MC_STATUS_MCACOD
		31:16	MC_STATUS_MSCOD
		57	MC_STATUS_DAM
		58	MC_STATUS_ADDRV
		59	MC_STATUS_MISCV
		60	MC_STATUS_EN. (Note: For MC0_STATUS only, this bit is hardcoded to 1.)
		61	MC_STATUS_UC
		62	MC_STATUS_O
		63	MC_STATUS_V
402H	1026	MC0_ADDR	
403H	1027	MC0_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

Table 34-18. MSRs in the P6 Family Processors (Contd.)

Register Address		Register Name	Bit Description
Hex	Dec		
404H	1028	MC1_CTL	
405H	1029	MC1_STATUS	Bit definitions same as MC0_STATUS.
406H	1030	MC1_ADDR	
407H	1031	MC1_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
408H	1032	MC2_CTL	
409H	1033	MC2_STATUS	Bit definitions same as MC0_STATUS.
40AH	1034	MC2_ADDR	
40BH	1035	MC2_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
40CH	1036	MC4_CTL	
40DH	1037	MC4_STATUS	Bit definitions same as MC0_STATUS, except bits 0, 4, 57, and 61 are hardcoded to 1.
40EH	1038	MC4_ADDR	Defined in MCA architecture but not implemented in P6 Family processors.
40FH	1039	MC4_MISC	Defined in MCA architecture but not implemented in the P6 family processors.
410H	1040	MC3_CTL	
411H	1041	MC3_STATUS	Bit definitions same as MC0_STATUS.
412H	1042	MC3_ADDR	
413H	1043	MC3_MISC	Defined in MCA architecture but not implemented in the P6 family processors.

NOTES

1. Bit 0 of this register has been redefined several times, and is no longer used in P6 family processors.
2. The processor number feature may be disabled by setting bit 21 of the BBL_CR_CTL MSR (model-specific register address 119h) to “1”. Once set, bit 21 of the BBL_CR_CTL may not be cleared. This bit is write-once. The processor number feature will be disabled until the processor is reset.
3. The Pentium III processor will prevent FSB frequency overclocking with a new shutdown mechanism. If the FSB frequency selected is greater than the internal FSB frequency the processor will shutdown. If the FSB selected is less than the internal FSB frequency the BIOS may choose to use bit 11 to implement its own shutdown policy.

34.12 MSRS IN PENTIUM PROCESSORS

The following MSRs are defined for the Pentium processors. The P5_MC_ADDR, P5_MC_TYPE, and TSC MSRs (named IA32_P5_MC_ADDR, IA32_P5_MC_TYPE, and IA32_TIME_STAMP_COUNTER in the Pentium 4 processor) are architectural; that is, code that accesses these registers will run on Pentium 4 and P6 family processors without generating exceptions (see Section 34.1, "Architectural MSRs"). The CESR, CTR0, and CTR1 MSRs are unique to Pentium processors; code that accesses these registers will generate exceptions on Pentium 4 and P6 family processors.

Table 34-19. MSRs in the Pentium Processor

Register Address			
Hex	**Dec**	**Register Name**	**Bit Description**
0H	0	P5_MC_ADDR	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
1H	1	P5_MC_TYPE	See Section 15.10.2, "Pentium Processor Machine-Check Exception Handling."
10H	16	TSC	See Section 17.12, "Time-Stamp Counter."
11H	17	CESR	See Section 18.17.1, "Control and Event Select Register (CESR)."
12H	18	CTR0	Section 18.17.3, "Events Counted."
13H	19	CTR1	Section 18.17.3, "Events Counted."

APPENDIX A VMX CAPABILITY REPORTING FACILITY

The ability of a processor to support VMX operation and related instructions is indicated by CPUID.1:ECX.VMX[bit 5] = 1. A value 1 in this bit indicates support for VMX features.

Support for specific features detailed in Chapter 24 and other VMX chapters is determined by reading values from a set of capability MSRs. These MSRs are indexed starting at MSR address 480H. VMX capability MSRs are read-only; an attempt to write them (with WRMSR) produces a general-protection exception (#GP(0)). They do not exist on processors that do not support VMX operation; an attempt to read them (with RDMSR) on such processors produces a general-protection exception (#GP(0)).

A.1 BASIC VMX INFORMATION

The IA32_VMX_BASIC MSR (index 480H) consists of the following fields:

- Bits 31:0 contain the 32-bit VMCS revision identifier used by the processor. Logical processors that use the same VMCS revision identifier use the same size for VMCS regions (see next item)
- Bits 44:32 report the number of bytes that software should allocate for the VMXON region and any VMCS region. It is a value greater than 0 and at most 4096 (bit 44 is set if and only if bits 43:32 are clear).
- Bit 48 indicates the width of the physical addresses that may be used for the VMXON region, each VMCS, and data structures referenced by pointers in a VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions). If the bit is 0, these addresses are limited to the processor's physical-address width.[1] If the bit is 1, these addresses are limited to 32 bits. This bit is always 0 for processors that support Intel 64 architecture.
- If bit 49 is read as 1, the logical processor supports the dual-monitor treatment of system-management interrupts and system-management mode. See Section 29.15 for details of this treatment.
- Bits 53:50 report the memory type that the logical processor uses to access the VMCS for VMREAD and VMWRITE and to access the VMCS, data structures referenced by pointers in the VMCS (I/O bitmaps, virtual-APIC page, MSR areas for VMX transitions), and the MSEG header during VM entries, VM exits, and in VMX non-root operation.[2]

1. On processors that support Intel 64 architecture, the pointer must not set bits beyond the processor's physical address width.

The first processors to support VMX operation use the write-back type. The values used are given in Table A-1.

Table A-1. Memory Types Used For VMCS Access

Value(s)	Field
0	Uncacheable (UC)
1-5	Not used
6	Write Back (WB)
7-15	Not used

If software needs to access these data structures (e.g., to modify the contents of the MSR bitmaps), it can configure the paging structures to map them into the linear-address space. If it does so, it should establish mappings that use the memory type reported in this MSR.[1]

- If bit 54 is read as 1, the logical processor reports information in the VM-exit instruction-information field on VM exits due to execution of the INS and OUTS instructions. This reporting is done only if this bit is read as 1.
- Bit 55 is read as 1 if any VMX controls that default to 1 may be cleared to 0. See Appendix A.2 for details. It also reports support for the VMX capability MSRs IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3.1, Appendix A.3.2, Appendix A.4, and Appendix A.5 for details.
- The values of bits 47:45 and bits 63:56 are reserved and are read as 0.

A.2 RESERVED CONTROLS AND DEFAULT SETTINGS

As noted in Chapter 24, "Virtual-Machine Control Structures", certain VMX controls are reserved and must be set to a specific value (0 or 1) determined by the processor. The specific value to which a reserved control must be set is its **default setting**.

2. If the MTRRs are disabled by clearing the E bit (bit 11) in the IA32_MTRR_DEF_TYPE MSR, the logical processor uses the UC memory type to access the indicated data structures, regardless of the value reported in bits 53:50 in the IA32_VMX_BASIC MSR. The processor will also use the UC memory type if the setting of CR0.CD on this logical processor (or another logical processor on the same physical processor) would cause it to do so for all memory accesses. The values of IA32_MTRR_DEF_TYPE.E and CR0.CD do not affect the value reported in IA32_VMX_BASIC[53:50].

1. Alternatively, software may map any of these regions or structures with the UC memory type. (This may be necessary for the MSEG header.) Doing so is discouraged unless necessary as it will cause the performance of software accesses to those structures to suffer. The processor will continue to use the memory type reported in the VMX capability MSR IA32_VMX_BASIC with the exceptions noted.

Software can discover the default setting of a reserved control by consulting the appropriate VMX capability MSR (see Appendix A.3 through Appendix A.5).

Future processors may define new functionality for one or more reserved controls. Such processors would allow each newly defined control to be set either to 0 or to 1. Software that does not desire a control’s new functionality should set the control to its default setting. For that reason, it is useful for software to know the default settings of the reserved controls.

Default settings partition the various controls into the following classes:

- **Always-flexible**. These have never been reserved.
- **Default0**. These are (or have been) reserved with a default setting of 0.
- **Default1**. They are (or have been) reserved with a default setting of 1.

As noted in Appendix A.1, a logical processor uses bit 55 of the IA32_VMX_BASIC MSR to indicate whether any of the default1 controls may be 0:

- If bit 55 of the IA32_VMX_BASIC MSR is read as 0, all the default1 controls are reserved and must be 1. VM entry will fail if any of these controls are 1 (see Section 26.2.1).
- If bit 55 of the IA32_VMX_BASIC MSR is read as 1, not all the default1 controls are reserved, and some (but not necessarily all) may be 0. The CPU supports four (4) new VMX capability MSRs: IA32_VMX_TRUE_PINBASED_CTLS, IA32_VMX_TRUE_PROCBASED_CTLS, IA32_VMX_TRUE_EXIT_CTLS, and IA32_VMX_TRUE_ENTRY_CTLS. See Appendix A.3 through Appendix A.5 for details. (These MSRs are not supported if bit 55 of the IA32_VMX_BASIC MSR is read as 0.)

See Section 30.5.1 for recommended software algorithms for proper capability detection of the default1 controls.

A.3 VM-EXECUTION CONTROLS

There are separate capability MSRs for the pin-based VM-execution controls, the primary processor-based VM-execution controls, and the secondary processor-based VM-execution controls. These are described in Appendix A.3.1, Appendix A.3.2, and Appendix A.3.3, respectively.

A.3.1 Pin-Based VM-Execution Controls

The IA32_VMX_PINBASED_CTLS MSR (index 481H) reports on the allowed settings of **most** of the pin-based VM-execution controls (see Section 24.6.1):

- Bits 31:0 indicate the **allowed 0-settings** of these controls. VM entry allows control X (bit X of the pin-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the pin-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 2, and 4; the corresponding bits of the IA32_VMX_PINBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  - If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any pin-based VM-execution control in the default1 class is 0.
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (see below) reports which of the pin-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the **allowed 1-settings** of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PINBASED_CTLS MSR (index 48DH) reports on the allowed settings of **all** of the pin-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the pin-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_PINBASED_CTLS MSR. (The IA32_VMX_TRUE_PINBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the pin-based VM-execution controls is contained in the IA32_VMX_TRUE_PINBASED_CTLS MSR. Assuming that software knows that the default1 class of pin-based VM-execution controls contains bits 1, 2, and 4, there is no need for software to consult the IA32_VMX_PINBASED_CTLS MSR.

A.3.2 Primary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS MSR (index 482H) reports on the allowed settings of **most** of the primary processor-based VM-execution controls (see Section 24.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the primary processor-based VM-execution controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

Exceptions are made for the primary processor-based VM-execution controls in the default1 class (see Appendix A.2). These are bits 1, 4–6, 8, 13–16, and 26; the corresponding bits of the IA32_VMX_PROCBASED_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  - If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any of the primary processor-based VM-execution controls in the default1 class is 0.
  - If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_PROCBASED_CTLS MSR (see below) reports which of the primary processor-based VM-execution controls in the default1 class can be 0 on VM entry.

- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1,
the IA32_VMX_TRUE_PROCBASED_CTLS MSR (index 48EH) reports on the allowed settings of **all** of the primary processor-based VM-execution controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the primary processor-based VM-execution controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the primary processor-based VM-execution controls is contained in the IA32_VMX_PROCBASED_CTLS MSR. (The IA32_VMX_TRUE_PROCBASED_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the processor-based VM-execution controls is contained in the IA32_VMX_TRUE_PROCBASED_CTLS MSR. Assuming that software knows that the default1 class of processor-based VM-execution controls contains bits 1, 4–6, 8, 13–16, and 26, there is no need for software to consult the IA32_VMX_PROCBASED_CTLS MSR.

A.3.3 Secondary Processor-Based VM-Execution Controls

The IA32_VMX_PROCBASED_CTLS2 MSR (index 48BH) reports on the allowed settings of the secondary processor-based VM-execution controls (see Section 24.6.2). VM entries perform the following checks:

- Bits 31:0 indicate the allowed 0-settings of these controls. These bits are always 0. This fact indicates that VM entry allows each bit of the secondary processor-based VM-execution controls to be 0 (reserved bits must be 0)
- Bits 63:32 indicate the allowed 1-settings of these controls; the 1-setting is not allowed for any reserved bit. VM entry allows control X (bit X of the secondary processor-based VM-execution controls) to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X and the "activate secondary controls" primary processor-based VM-execution control are both 1.

The IA32_VMX_PROCBASED_CTLS2 MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1).

A.4 VM-EXIT CONTROLS

The IA32_VMX_EXIT_CTLS MSR (index 483H) reports on the allowed settings of **most** of the VM-exit controls (see Section 24.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-exit controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

  Exceptions are made for the VM-exit controls in the default1 class (see Appendix A.2). These are bits 0–8, 10, 11, 13, 14, 16, and 17; the corresponding bits of the IA32_VMX_EXIT_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-exit control in the default1 class is 0.

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (see below) reports which of the VM-exit controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_EXIT_CTLS MSR (index 48FH) reports on the allowed settings of **all** of the VM-exit controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control X to be 1 if bit 32+X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-exit controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-exit controls is contained in the IA32_VMX_EXIT_CTLS MSR. (The IA32_VMX_TRUE_EXIT_CTLS MSR is not supported.)
- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-exit controls is contained in the IA32_VMX_TRUE_EXIT_CTLS MSR. Assuming that software knows that the default1 class of VM-exit controls contains bits 0–8, 10, 11, 13, 14, 16, and 17, there is no need for software to consult the IA32_VMX_EXIT_CTLS MSR.

A.5 VM-ENTRY CONTROLS

The IA32_VMX_ENTRY_CTLS MSR (index 484H) reports on the allowed settings of **most** of the VM-entry controls (see Section 24.8.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X (bit X of the VM-entry controls) to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0.

  Exceptions are made for the VM-entry controls in the default1 class (see Appendix A.2). These are bits 0–8 and 12; the corresponding bits of the IA32_VMX_ENTRY_CTLS MSR are always read as 1. The treatment of these controls by VM entry is determined by bit 55 in the IA32_VMX_BASIC MSR:

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 0, VM entry fails if any VM-entry control in the default1 class is 0.

  — If bit 55 in the IA32_VMX_BASIC MSR is read as 1, the IA32_VMX_TRUE_ENTRY_CTLS MSR (see below) reports which of the VM-entry controls in the default1 class can be 0 on VM entry.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X is 1 in the VM-entry controls and bit 32+X is 0 in this MSR.

If bit 55 in the IA32_VMX_BASIC MSR is read as 1,
the IA32_VMX_TRUE_ENTRY_CTLS MSR (index 490H) reports on the allowed settings of **all** of the VM-entry controls:

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry allows control X to be 0 if bit X in the MSR is cleared to 0; if bit X in the MSR is set to 1, VM entry fails if control X is 0. There are no exceptions.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry allows control 32+X to be 1 if bit X in the MSR is set to 1; if bit 32+X in the MSR is cleared to 0, VM entry fails if control X is 1.

It is necessary for software to consult only one of the capability MSRs to determine the allowed settings of the VM-entry controls:

- If bit 55 in the IA32_VMX_BASIC MSR is read as 0, all information about the allowed settings of the VM-entry controls is contained in the

IA32_VMX_ENTRY_CTLS MSR. (The IA32_VMX_TRUE_ENTRY_CTLS MSR is not supported.)

- If bit 55 in the IA32_VMX_BASIC MSR is read as 1, all information about the allowed settings of the VM-entry controls is contained in the IA32_VMX_TRUE_ENTRY_CTLS MSR. Assuming that software knows that the default1 class of VM-entry controls contains bits 0–8 and 12, there is no need for software to consult the IA32_VMX_ENTRY_CTLS MSR.

A.6 MISCELLANEOUS DATA

The IA32_VMX_MISC MSR (index 485H) consists of the following fields:

- Bits 4:0 report a value X that specifies the relationship between the rate of the VMX-preemption timer and that of the timestamp counter (TSC). Specifically, the VMX-preemption timer (if it is active) counts down by 1 every time bit X in the TSC changes due to a TSC increment.
- If bit 5 is read as 1, VM exits store the value of IA32_EFER.LMA into the “IA-32e mode guest” VM-entry control; see Section 27.2 for more details. This bit is read as 1 on any logical processor that supports the 1-setting of the “unrestricted guest” VM-execution control.
- Bits 8:6 report, as a bitmap, the activity states supported by the implementation:
  - Bit 6 reports (if set) the support for activity state 1 (HLT).
  - Bit 7 reports (if set) the support for activity state 2 (shutdown).
  - Bit 8 reports (if set) the support for activity state 3 (wait-for-SIPI).

  If an activity state is not supported, the implementation causes a VM entry to fail if it attempts to establish that activity state. All implementations support VM entry to activity state 0 (active).
- Bits 24:16 indicate the number of CR3-target values supported by the processor. This number is a value between 0 and 256, inclusive (bit 24 is set if and only if bits 23:16 are clear).
- Bits 27:25 is used to compute the recommended maximum number of MSRs that should appear in the VM-exit MSR-store list, the VM-exit MSR-load list, or the VM-entry MSR-load list. Specifically, if the value bits 27:25 of IA32_VMX_MISC is N, then 512 * (N + 1) is the recommended maximum number of MSRs to be included in each list. If the limit is exceeded, undefined processor behavior may result (including a machine check during the VMX transition).
- If bit 28 is read as 1, bit 2 of the IA32_SMM_MONITOR_CTL can be set to 1. VMXOFF unblocks SMIs unless IA32_SMM_MONITOR_CTL[bit 2] is 1 (see Section 29.14.4).
- Bits 63:32 report the 32-bit MSEG revision identifier used by the processor.
- Bits 15:9 and bits 31:29 are reserved and are read as 0.

A.7 VMX-FIXED BITS IN CR0

The IA32_VMX_CR0_FIXED0 MSR (index 486H) and IA32_VMX_CR0_FIXED1 MSR (index 487H) indicate how bits in CR0 may be set in VMX operation. They report on bits in CR0 that are allowed to be 0 and to be 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit of CR0 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR0_FIXED0, then that bit is also 1 in IA32_VMX_CR0_FIXED1; if bit X is 0 in IA32_VMX_CR0_FIXED1, then that bit is also 0 in IA32_VMX_CR0_FIXED0. Thus, each bit in CR0 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR0_FIXED0 and 1 in IA32_VMX_CR0_FIXED1).

A.8 VMX-FIXED BITS IN CR4

The IA32_VMX_CR4_FIXED0 MSR (index 488H) and IA32_VMX_CR4_FIXED1 MSR (index 489H) indicate how bits in CR4 may be set in VMX operation. They report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit of CR4 is fixed to 0 in VMX operation. It is always the case that, if bit X is 1 in IA32_VMX_CR4_FIXED0, then that bit is also 1 in IA32_VMX_CR4_FIXED1; if bit X is 0 in IA32_VMX_CR4_FIXED1, then that bit is also 0 in IA32_VMX_CR4_FIXED0. Thus, each bit in CR4 is either fixed to 0 (with value 0 in both MSRs), fixed to 1 (1 in both MSRs), or flexible (0 in IA32_VMX_CR4_FIXED0 and 1 in IA32_VMX_CR4_FIXED1).

A.9 VMCS ENUMERATION

The IA32_VMX_VMCS_ENUM MSR (index 48AH) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 24.10.2, each field in the VMCS is associated with a 32-bit encoding which is structured as follows:

- Bits 31:15 are reserved (must be 0).
- Bits 14:13 indicate the field's width.
- Bit 12 is reserved (must be 0).
- Bits 11:10 indicate the field's type.
- Bits 9:1 is an index field that distinguishes different fields with the same width and type.
- Bit 0 indicates access type.

IA32_VMX_VMCS_ENUM indicates to software the highest index value used in the encoding of any field supported by the processor:

- Bits 9:1 contain the highest index value used for any VMCS encoding.
- Bit 0 and bits 63:10 are reserved and are read as 0.

A.10 VPID AND EPT CAPABILITIES

The IA32_VMX_EPT_VPID_CAP MSR (index 48CH) reports information about the capabilities of the logical processor with regard to virtual-processor identifiers (VPIDs, Section 28.1) and extended page tables (EPT, Section 28.2):

- If bit 0 is read as 1, the logical processor allows software to configure EPT paging-structure entries in which bits 2:0 have value 100b (indicating an execute-only translation).
- Bit 6 indicates support for a page-walk length of 4.
- If bit 8 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be uncacheable (UC); see Section 24.6.11.
- If bit 14 is read as 1, the logical processor allows software to configure the EPT paging-structure memory type to be write-back (WB).
- If bit 16 is read as 1, the logical processor allows software to configure a EPT PDE to map a 2-Mbyte page (by setting bit 7 in the EPT PDE).
- If bit 17 is read as 1, the logical processor allows software to configure a EPT PDPTE to map a 1-Gbyte page (by setting bit 7 in the EPT PDPTE).
- Support for the INVEPT instruction (see Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* and Section 28.3.3.1).
  - If bit 20 is read as 1, the INVEPT instruction is supported.
  - If bit 25 is read as 1, the single-context INVEPT type is supported.
  - If bit 26 is read as 1, the all-context INVEPT type is supported.
- Support for the INVVPID instruction (see Chapter 33 of the *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3C* and Section 28.3.3.1).
  - If bit 32 is read as 1, the INVVPID instruction is supported.
  - If bit 40 is read as 1, the individual-address INVVPID type is supported.
  - If bit 41 is read as 1, the single-context INVVPID type is supported.
  - If bit 42 is read as 1, the all-context INVVPID type is supported.
  - If bit 43 is read as 1, the single-context-retaining-globals INVVPID type is supported.
- Bits 5:1, bit 7, bits 13:9, bit 15, bits 19:17, bits 24:21, bits 31:27, bits 39:33, and bits 63:44 are reserved and are read as 0.

The IA32_VMX_EPT_VPID_CAP MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and that support either the 1-setting of the "enable EPT" VM-execution control (only if bit 33 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1) or the 1-setting of the "enable VPID" VM-execution control (only if bit 37 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1).

A.11 VM FUNCTIONS

The IA32_VMX_VMFUNC MSR (index 491H) reports on the allowed settings of the VM-function controls (see Section 24.6.14). VM entry allows bit X of the VM-function controls to be 1 if bit X in the MSR is set to 1; if bit X in the MSR is cleared to 0, VM entry fails if bit X of the VM-function controls, the "activate secondary controls" primary processor-based VM-execution control, and the "enable VM functions" secondary processor-based VM-execution control are all 1.

The IA32_VMX_VMFUNC MSR exists only on processors that support the 1-setting of the "activate secondary controls" VM-execution control (only if bit 63 of the IA32_VMX_PROCBASED_CTLS MSR is 1) and the 1-setting of the "enable VM functions" secondary processor-based VM-execution control (only if bit 45 of the IA32_VMX_PROCBASED_CTLS2 MSR is 1).

APPENDIX B FIELD ENCODING IN VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 24.10.2 describes the structure of the encoding space (the meanings of the bits in each 32-bit encoding).

This appendix enumerates all fields in the VMCS and their encodings. Fields are grouped by width (16-bit, 32-bit, etc.) and type (guest-state, host-state, etc.)

B.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only guest-state areas and the host-state area contain 16-bit fields. As noted in Section 24.10.2, each 16-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.1.1 16-Bit Control Field

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. There is only one such 16-bit field as given in Table B-1.

Table B-1. Encoding for 16-Bit Control Fields (0000_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Virtual-processor identifier (VPID)[1]	000000000B	00000000H

NOTES:

1. This field exists only on processors that support the 1-setting of the “enable VPID” VM-execution control.

B.1.2 16-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-2 enumerates 16-bit guest-state fields.

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES selector	000000000B	00000800H

Table B-2. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CS selector	000000001B	00000802H
Guest SS selector	000000010B	00000804H
Guest DS selector	000000011B	00000806H
Guest FS selector	000000100B	00000808H
Guest GS selector	000000101B	0000080AH
Guest LDTR selector	000000110B	0000080CH
Guest TR selector	000000111B	0000080EH

B.1.3 16-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-3 enumerates the 16-bit host-state fields.

Table B-3. Encodings for 16-Bit Host-State Fields (0000_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host ES selector	000000000B	00000C00H
Host CS selector	000000001B	00000C02H
Host SS selector	000000010B	00000C04H
Host DS selector	000000011B	00000C06H
Host FS selector	000000100B	00000C08H
Host GS selector	000000101B	00000C0AH
Host TR selector	000000110B	00000C0CH

B.2 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a 64-bit field. There are 64-bit fields only for controls and for guest state. As noted in Section 24.10.2, every 64-bit field has two encodings, which differ on bit 0, the access type. Thus, each such field has an even encoding for full access and an odd encoding for high access.

B.2.1 64-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-4 enumerates the 64-bit control fields.

Table B-4. Encodings for 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)

Field Name	Index	Encoding
Address of I/O bitmap A (full)	000000000B	00002000H
Address of I/O bitmap A (high)	000000000B	00002001H
Address of I/O bitmap B (full)	000000001B	00002002H
Address of I/O bitmap B (high)	000000001B	00002003H
Address of MSR bitmaps (full)[1]	000000010B	00002004H
Address of MSR bitmaps (high)[1]	000000010B	00002005H
VM-exit MSR-store address (full)	000000011B	00002006H
VM-exit MSR-store address (high)	000000011B	00002007H
VM-exit MSR-load address (full)	000000100B	00002008H
VM-exit MSR-load address (high)	000000100B	00002009H
VM-entry MSR-load address (full)	000000101B	0000200AH
VM-entry MSR-load address (high)	000000101B	0000200BH
Executive-VMCS pointer (full)	000000110B	0000200CH
Executive-VMCS pointer (high)	000000110B	0000200DH
TSC offset (full)	000001000B	00002010H
TSC offset (high)	000001000B	00002011H
Virtual-APIC address (full)[2]	000001001B	00002012H
Virtual-APIC address (high)[2]	000001001B	00002013H
APIC-access address (full)[3]	000001010B	00002014H
APIC-access address (high)[3]	000001010B	00002015H
VM-function controls (full)[4]	000001100B	00002018H
VM-function controls (high)[4]	000001100B	00002019H
EPT pointer (EPTP; full)[5]	000001101B	0000201AH
EPT pointer (EPTP; high)[5]	000001101B	0000201BH
EPTP-list address (full)[6]	000010010B	00002024H
EPTP-list address (high)[6]	000010010B	00002025H

NOTES:

1. This field exists only on processors that support the 1-setting of the "use MSR bitmaps" VM-execution control.
2. This field exists only on processors that support either the 1-setting of the "use TPR shadow" VM-execution control.
3. This field exists only on processors that support the 1-setting of the "virtualize APIC accesses" VM-execution control.
4. This field exists only on processors that support the 1-setting of the "enable VM functions" VM-execution control.
5. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.
6. This field exists only on processors that support the 1-setting of the "EPTP switching" VM-function control.

B.2.2 64-Bit Read-Only Data Field

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. There is only one such 64-bit field as given in Table B-5.(As with other 64-bit fields, this one has two encodings.)

Table B-5. Encodings for 64-Bit Read-Only Data Field (0010_01xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest-physical address (full)[1]	000000000B	00002400H
Guest-physical address (high)[1]	000000000B	00002401H

NOTES:

1. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

B.2.3 64-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-6 enumerates the 64-bit guest-state fields.

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
VMCS link pointer (full)	000000000B	00002800H
VMCS link pointer (high)	000000000B	00002801H
Guest IA32_DEBUGCTL (full)	000000001B	00002802H

Table B-6. Encodings for 64-Bit Guest-State Fields (0010_10xx_xxxx_xxxAb)

Field Name	Index	Encoding
Guest IA32_DEBUGCTL (high)	000000001B	00002803H
Guest IA32_PAT (full)[1]	000000010B	00002804H
Guest IA32_PAT (high)[1]	000000010B	00002805H
Guest IA32_EFER (full)[2]	000000011B	00002806H
Guest IA32_EFER (high)[2]	000000011B	00002807H
Guest IA32_PERF_GLOBAL_CTRL (full)[3]	000000100B	00002808H
Guest IA32_PERF_GLOBAL_CTRL (high)[3]	000000100B	00002809H
Guest PDPTE0 (full)[4]	000000101B	0000280AH
Guest PDPTE0 (high)[4]	000000101B	0000280BH
Guest PDPTE1 (full)[4]	000000110B	0000280CH
Guest PDPTE1 (high)[4]	000000110B	0000280DH
Guest PDPTE2 (full)[4]	000000111B	0000280EH
Guest PDPTE2 (high)[4]	000000111B	0000280FH
Guest PDPTE3 (full)[4]	000001000B	00002810H
Guest PDPTE3 (high)[4]	000001000B	00002811H

NOTES:
1. This field exists only on processors that support either the 1-setting of the "load IA32_PAT" VM-entry control or that of the "save IA32_PAT" VM-exit control.
2. This field exists only on processors that support either the 1-setting of the "load IA32_EFER" VM-entry control or that of the "save IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-entry control.
4. This field exists only on processors that support the 1-setting of the "enable EPT" VM-execution control.

B.2.4 64-Bit Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-7 enumerates the 64-bit control fields.

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (full)[1]	000000000B	00002C00H

Table B-7. Encodings for 64-Bit Host-State Fields (0010_11xx_xxxx_xxxAb)

Field Name	Index	Encoding
Host IA32_PAT (high)[1]	000000000B	00002C01H
Host IA32_EFER (full)[2]	000000001B	00002C02H
Host IA32_EFER (high)[2]	000000001B	00002C03H
Host IA32_PERF_GLOBAL_CTRL (full)[3]	000000010B	00002C04H
Host IA32_PERF_GLOBAL_CTRL (high)[3]	000000010B	00002C05H

NOTES:

1. This field exists only on processors that support the 1-setting of the "load IA32_PAT" VM-exit control.
2. This field exists only on processors that support the 1-setting of the "load IA32_EFER" VM-exit control.
3. This field exists only on processors that support the 1-setting of the "load IA32_PERF_GLOBAL_CTRL" VM-exit control.

B.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 24.10.2, each 32-bit field allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.3.1 32-Bit Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-8 enumerates the 32-bit control fields.

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
Pin-based VM-execution controls	000000000B	00004000H
Primary processor-based VM-execution controls	000000001B	00004002H
Exception bitmap	000000010B	00004004H
Page-fault error-code mask	000000011B	00004006H
Page-fault error-code match	000000100B	00004008H
CR3-target count	000000101B	0000400AH
VM-exit controls	000000110B	0000400CH
VM-exit MSR-store count	000000111B	0000400EH

Table B-8. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
VM-exit MSR-load count	000001000B	00004010H
VM-entry controls	000001001B	00004012H
VM-entry MSR-load count	000001010B	00004014H
VM-entry interruption-information field	000001011B	00004016H
VM-entry exception error code	000001100B	00004018H
VM-entry instruction length	000001101B	0000401AH
TPR threshold[1]	000001110B	0000401CH
Secondary processor-based VM-execution controls[2]	000001111b	0000401EH
PLE_Gap[3]	000010000b	00004020H
PLE_Window[3]	000010001b	00004022H

NOTES:

1. This field exists only on processors that support the 1-setting of the “use TPR shadow” VM-execution control.
2. This field exists only on processors that support the 1-setting of the “activate secondary controls” VM-execution control.
3. This field exists only on processors that support the 1-setting of the “PAUSE-loop exiting” VM-execution control.

B.3.2 32-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-9 enumerates the 32-bit read-only data fields.

Table B-9. Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
VM-instruction error	000000000B	00004400H
Exit reason	000000001B	00004402H
VM-exit interruption information	000000010B	00004404H
VM-exit interruption error code	000000011B	00004406H
IDT-vectoring information field	000000100B	00004408H
IDT-vectoring error code	000000101B	0000440AH
VM-exit instruction length	000000110B	0000440CH
VM-exit instruction information	000000111B	0000440EH

B.3.3 32-Bit Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-10 enumerates the 32-bit guest-state fields.

Table B-10. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest ES limit	000000000B	00004800H
Guest CS limit	000000001B	00004802H
Guest SS limit	000000010B	00004804H
Guest DS limit	000000011B	00004806H
Guest FS limit	000000100B	00004808H
Guest GS limit	000000101B	0000480AH
Guest LDTR limit	000000110B	0000480CH
Guest TR limit	000000111B	0000480EH
Guest GDTR limit	000001000B	00004810H
Guest IDTR limit	000001001B	00004812H
Guest ES access rights	000001010B	00004814H
Guest CS access rights	000001011B	00004816H
Guest SS access rights	000001100B	00004818H
Guest DS access rights	000001101B	0000481AH
Guest FS access rights	000001110B	0000481CH
Guest GS access rights	000001111B	0000481EH
Guest LDTR access rights	000010000B	00004820H
Guest TR access rights	000010001B	00004822H
Guest interruptibility state	000010010B	00004824H
Guest activity state	000010011B	00004826H
Guest SMBASE	000010100B	00004828H
Guest IA32_SYSENTER_CS	000010101B	0000482AH
VMX-preemption timer value[1]	000010111B	0000482EH

NOTES:

1. This field exists only on processors that support the 1-setting of the "activate VMX-preemption timer" VM-execution control.

The limit fields for GDTR and IDTR are defined to be 32 bits in width even though these fields are only 16-bits wide in the Intel 64 and IA-32 architectures. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

B.3.4 32-Bit Host-State Field

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. There is only one such 32-bit field as given in Table B-11.

Table B-11. Encoding for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host IA32_SYSENTER_CS	000000000B	00004C00H

B.4 NATURAL-WIDTH FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural-width field. As noted in Section 24.10.2, each of these fields allows only full access, meaning that bit 0 of its encoding is 0. Each such encoding is thus an even number.

B.4.1 Natural-Width Control Fields

A value of 0 in bits 11:10 of an encoding indicates a control field. These fields are distinguished by their index value in bits 9:1. Table B-12 enumerates the natural-width control fields.

Table B-12. Encodings for Natural-Width Control Fields (0110_00xx_xxxx_xxx0B)

Field Name	Index	Encoding
CR0 guest/host mask	000000000B	00006000H
CR4 guest/host mask	000000001B	00006002H
CR0 read shadow	000000010B	00006004H
CR4 read shadow	000000011B	00006006H
CR3-target value 0	000000100B	00006008H
CR3-target value 1	000000101B	0000600AH
CR3-target value 2	000000110B	0000600CH
CR3-target value 3[1]	000000111B	0000600EH

NOTES:

1. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

B.4.2 Natural-Width Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table B-13 enumerates the natural-width read-only data fields.

Table B-13. Encodings for Natural-Width Read-Only Data Fields (0110_01xx_xxxx_xxx0B)

Field Name	Index	Encoding
Exit qualification	000000000B	00006400H
I/O RCX	000000001B	00006402H
I/O RSI	000000010B	00006404H
I/O RDI	000000011B	00006406H
I/O RIP	000000100B	00006408H
Guest-linear address	000000101B	0000640AH

B.4.3 Natural-Width Guest-State Fields

A value of 2 in bits 11:10 of an encoding indicates a field in the guest-state area. These fields are distinguished by their index value in bits 9:1. Table B-14 enumerates the natural-width guest-state fields.

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B)

Field Name	Index	Encoding
Guest CR0	000000000B	00006800H
Guest CR3	000000001B	00006802H
Guest CR4	000000010B	00006804H
Guest ES base	000000011B	00006806H
Guest CS base	000000100B	00006808H
Guest SS base	000000101B	0000680AH
Guest DS base	000000110B	0000680CH

Table B-14. Encodings for Natural-Width Guest-State Fields (0110_10xx_xxxx_xxx0B) (Contd.)

Field Name	Index	Encoding
Guest FS base	000000111B	0000680EH
Guest GS base	000001000B	00006810H
Guest LDTR base	000001001B	00006812H
Guest TR base	000001010B	00006814H
Guest GDTR base	000001011B	00006816H
Guest IDTR base	000001100B	00006818H
Guest DR7	000001101B	0000681AH
Guest RSP	000001110B	0000681CH
Guest RIP	000001111B	0000681EH
Guest RFLAGS	000010000B	00006820H
Guest pending debug exceptions	000010001B	00006822H
Guest IA32_SYSENTER_ESP	000010010B	00006824H
Guest IA32_SYSENTER_EIP	000010011B	00006826H

The base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural-width (with 64 bits on processors supporting Intel 64 architecture) even though these fields are only 32-bits wide in the Intel 64 architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

B.4.4 Natural-Width Host-State Fields

A value of 3 in bits 11:10 of an encoding indicates a field in the host-state area. These fields are distinguished by their index value in bits 9:1. Table B-15 enumerates the natural-width host-state fields.

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B)

Field Name	Index	Encoding
Host CR0	000000000B	00006C00H
Host CR3	000000001B	00006C02H
Host CR4	000000010B	00006C04H
Host FS base	000000011B	00006C06H
Host GS base	000000100B	00006C08H
Host TR base	000000101B	00006C0AH

Table B-15. Encodings for Natural-Width Host-State Fields (0110_11xx_xxxx_xxx0B) (Contd.)

Field Name	**Index**	**Encoding**
Host GDTR base	000000110B	00006C0CH
Host IDTR base	000000111B	00006C0EH
Host IA32_SYSENTER_ESP	000001000B	00006C10H
Host IA32_SYSENTER_EIP	000001001B	00006C12H
Host RSP	000001010B	00006C14H
Host RIP	000001011B	00006C16H

APPENDIX C
VMX BASIC EXIT REASONS

Every VM exit writes a 32-bit exit reason to the VMCS (see Section 24.9.1). Certain VM-entry failures also do this (see Section 26.7). The low 16 bits of the exit-reason field form the basic exit reason which provides basic information about the cause of the VM exit or VM-entry failure.

Table C-1 lists values for basic exit reasons and explains their meaning. Entries apply to VM exits, unless otherwise noted.

Table C-1. Basic Exit Reasons

Basic Exit Reason	Description
0	**Exception or non-maskable interrupt (NMI).** Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1. This case includes executions of BOUND that cause #BR, executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD2 (they cause #UD).
1	**External interrupt.** An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	**Triple fault.** The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	**INIT signal.** An INIT signal arrived
4	**Start-up IPI (SIPI).** A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	**I/O system-management interrupt (SMI).** An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 29.15.2).
6	**Other SMI.** An SMI arrived and caused an SMM VM exit (see Section 29.15.2) but not immediately after retirement of an I/O instruction.
7	**Interrupt window.** At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	**NMI window.** At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	**Task switch.** Guest software attempted a task switch.
10	**CPUID.** Guest software attempted to execute CPUID.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
11	**GETSEC.** Guest software attempted to execute GETSEC.
12	**HLT.** Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.
13	**INVD.** Guest software attempted to execute INVD.
14	**INVLPG.** Guest software attempted to execute INVLPG and the "INVLPG exiting" VM-execution control was 1.
15	**RDPMC.** Guest software attempted to execute RDPMC and the "RDPMC exiting" VM-execution control was 1.
16	**RDTSC.** Guest software attempted to execute RDTSC and the "RDTSC exiting" VM-execution control was 1.
17	**RSM.** Guest software attempted to execute RSM in SMM.
18	**VMCALL.** VMCALL was executed either by guest software (causing an ordinary VM exit) or by the executive monitor (causing an SMM VM exit; see Section 29.15.2).
19	**VMCLEAR.** Guest software attempted to execute VMCLEAR.
20	**VMLAUNCH.** Guest software attempted to execute VMLAUNCH.
21	**VMPTRLD.** Guest software attempted to execute VMPTRLD.
22	**VMPTRST.** Guest software attempted to execute VMPTRST.
23	**VMREAD.** Guest software attempted to execute VMREAD.
24	**VMRESUME.** Guest software attempted to execute VMRESUME.
25	**VMWRITE.** Guest software attempted to execute VMWRITE.
26	**VMXOFF.** Guest software attempted to execute VMXOFF.
27	**VMXON.** Guest software attempted to execute VMXON.
28	**Control-register accesses.** Guest software attempted to access CR0, CR3, CR4, or CR8 using CLTS, LMSW, or MOV CR and the VM-execution control fields indicate that a VM exit should occur (see Section 25.1 for details). This basic exit reason is not used for trap-like VM exits following executions of the MOV to CR8 instruction when the "use TPR shadow" VM-execution control is 1.
29	**MOV DR.** Guest software attempted a MOV to or from a debug register and the "MOV-DR exiting" VM-execution control was 1.
30	**I/O instruction.** Guest software attempted to execute an I/O instruction and either: 1: The "use I/O bitmaps" VM-execution control was 0 and the "unconditional I/O exiting" VM-execution control was 1. 2: The "use I/O bitmaps" VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1.

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
31	**RDMSR.** Guest software attempted to execute RDMSR and either: 1: The "use MSR bitmaps" VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the *n*th bit in read bitmap for low MSRs is 1, where *n* was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the *n*th bit in read bitmap for high MSRs is 1, where *n* is the value of RCX & 00001FFFH.
32	**WRMSR.** Guest software attempted to execute WRMSR and either: 1: The "use MSR bitmaps" VM-execution control was 0. 2: The value of RCX is neither in the range 00000000H – 00001FFFH nor in the range C0000000H – C0001FFFH. 3: The value of RCX was in the range 00000000H – 00001FFFH and the *n*th bit in write bitmap for low MSRs is 1, where *n* was the value of RCX. 4: The value of RCX is in the range C0000000H – C0001FFFH and the *n*th bit in write bitmap for high MSRs is 1, where *n* is the value of RCX & 00001FFFH.
33	**VM-entry failure due to invalid guest state.** A VM entry failed one of the checks identified in Section 26.3.1.
34	**VM-entry failure due to MSR loading.** A VM entry failed in an attempt to load MSRs. See Section 26.4.
36	**MWAIT.** Guest software attempted to execute MWAIT and the "MWAIT exiting" VM-execution control was 1.
37	**Monitor trap flag.** A VM entry occurred due to the 1-setting of the "monitor trap flag" VM-execution control and injection of an MTF VM exit as part of VM entry. See Section 25.7.2.
39	**MONITOR.** Guest software attempted to execute MONITOR and the "MONITOR exiting" VM-execution control was 1.
40	**PAUSE.** Either guest software attempted to execute PAUSE and the "PAUSE exiting" VM-execution control was 1 or the "PAUSE-loop exiting" VM-execution control was 1 and guest software executed a PAUSE loop with execution time exceeding PLE_Window (see Section 25.1.3).
41	**VM-entry failure due to machine-check event.** A machine-check event occurred during VM entry (see Section 26.8).
43	**TPR below threshold.** The logical processor determined that the value of the TPR shadow was below that of the TPR threshold VM-execution control field while the "use TPR shadow" VM-execution control was 1 in one of the following cases: ▪ After guest software executed MOV to CR8 (see Section 25.1.3). ▪ As part of a TPR-shadow update (see Section 25.5.3.3). ▪ After VM entry with the 1-setting of the "virtualize APIC accesses" VM-execution control (see Section 26.6.7).

Table C-1. Basic Exit Reasons (Contd.)

Basic Exit Reason	Description
44	**APIC access.** Guest software attempted to access memory at a physical address on the APIC-access page and the "virtualize APIC accesses" VM-execution control was 1 (see Section 25.2).
46	**Access to GDTR or IDTR.** Guest software attempted to execute LGDT, LIDT, SGDT, or SIDT and the "descriptor-table exiting" VM-execution control was 1.
47	**Access to LDTR or TR.** Guest software attempted to execute LLDT, LTR, SLDT, or STR and the "descriptor-table exiting" VM-execution control was 1.
48	**EPT violation.** An attempt to access memory with a guest-physical address was disallowed by the configuration of the EPT paging structures.
49	**EPT misconfiguration.** An attempt to access memory with a guest-physical address encountered a misconfigured EPT paging-structure entry.
50	**INVEPT.** Guest software attempted to execute INVEPT.
51	**RDTSCP.** Guest software attempted to execute RDTSCP and the "enable RDTSCP" and "RDTSC exiting" VM-execution controls were both 1.
52	**VMX-preemption timer expired.** The preemption timer counted down to zero.
53	**INVVPID.** Guest software attempted to execute INVVPID.
54	**WBINVD.** Guest software attempted to execute WBINVD and the "WBINVD exiting" VM-execution control was 1.
55	**XSETBV.** Guest software attempted to execute XSETBV.
57	**RDRAND.** Guest software attempted to execute RDRAND and the "RDRAND exiting" VM-execution control was 1.
58	**INVPCID.** Guest software attempted to execute INVPCID and the "enable INVPCID" and "INVLPG exiting" VM-execution controls were both 1.
59	**VMFUNC.** Guest software invoked a VM function with the VMFUNC instruction and encountered a function-specific condition causing a VM exit.

INDEX FOR VOLUMES 3A, 3B & 3C