# Intel® Virtualization Technology Specification for the IA-32 Intel® Architecture

C97063-002
April 2005

# CONTENTS

PAGE

# CHAPTER 1
# INTRODUCTION AND VMX OVERVIEW

## 1.1 ABOUT THIS DOCUMENT

This documents describes Intel® Virtualization Technology for IA-32 processors, referred to as VT-x. VT-x constitutes a set of virtual-machine extensions (VMX) that support virtualization of processor hardware for multiple software environments by using virtual machines.

This document is organized as follows:

- Chapter 1 gives an overview of the virtual-machine extensions.
- Chapter 2 details the virtual-machine control structure (VMCS) and its usage.
- Chapter 3 details processor behavior in VMX non-root operation.
- Chapter 4 details the operation of VM entries.
- Chapter 5 details the operation of VM exits.
- Chapter 6 details VMX capability reporting.
- Chapter 7 provides a reference for the new VMX instructions.
- Chapter 8 details interactions between VMX operation and system-management mode (SMM).

This document assumes the reader is familiar with IA-32 processor features and makes references to IA-32 features published in the following documents:

- The *IA-32 Intel® Architecture Software Developer's Manual, Volume 1: Basic Architecture*.
- The *IA-32 Intel® Architecture Software Developer's Manual, Volumes 2A & 2B: Instruction Set Reference*.
- The *IA-32 Intel® Architecture Software Developer's Manual, Volume 3: System Programming Guide*.
- The *Intel® Extended Memory 64 Technology Software Developer's Guide, Volume 1 & 2*.

## 1.2 VIRTUAL MACHINE ARCHITECTURE

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

- **Virtual-machine monitor (VMM)**: A VMM acts as a host and has full control of the processor(s) and other platform hardware. VMM presents guest software (see below) 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 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.

## 1.3 INTRODUCTION TO VMX OPERATION

Processor support for virtualization is provided by a new 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 1.8).

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 these 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.

## 1.4 LIFE CYCLE OF VMM SOFTWARE

Figure 1-1 illustrates the life cycle of a VMM and its guest software by illustrating the interactions between them.

- Software enters VMX operation through execution of the VMXON instruction.
- The VMM can then enter its guests into virtual machines (one at a time) using VM entries. The VMM effects a VM entry using the VMX instructions VMLAUNCH and

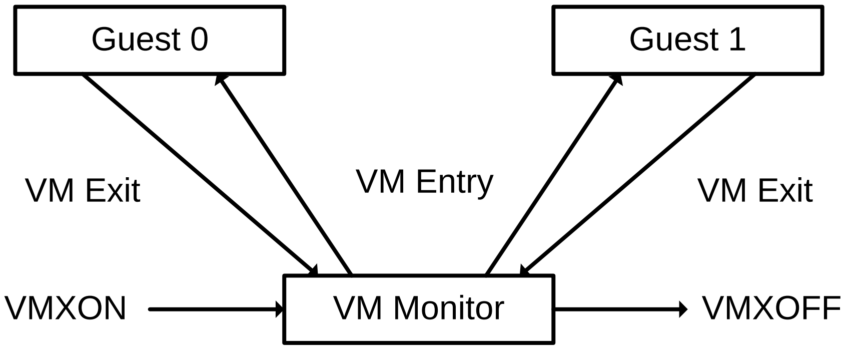

**Figure 1-1. Interaction of a Virtual-Machine Monitor and Guests**

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 via a VM entry.

- Eventually, the VMM may decide to shut itself down and leave VMX operation. It does so by executing the VMXOFF instruction.

## 1.5 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 can be read and written using the instructions VMPTRST and VMPTRLD. The VMM configures a VMCS using other instructions: VMREAD, VMWRITE, and VMCLEAR.

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.

Chapter 2 describes the structure of a VMCS. Chapter 3, Chapter 4, and Chapter 5 provide details on how the VMCS controls VMX non-root operation, VM entries, and VM exits. Chapter 7 provides detailed descriptions for each of the new VMX instructions.

## 1.6 DISCOVERING SUPPORT FOR VMX OPERATION

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 Figure 1-2.

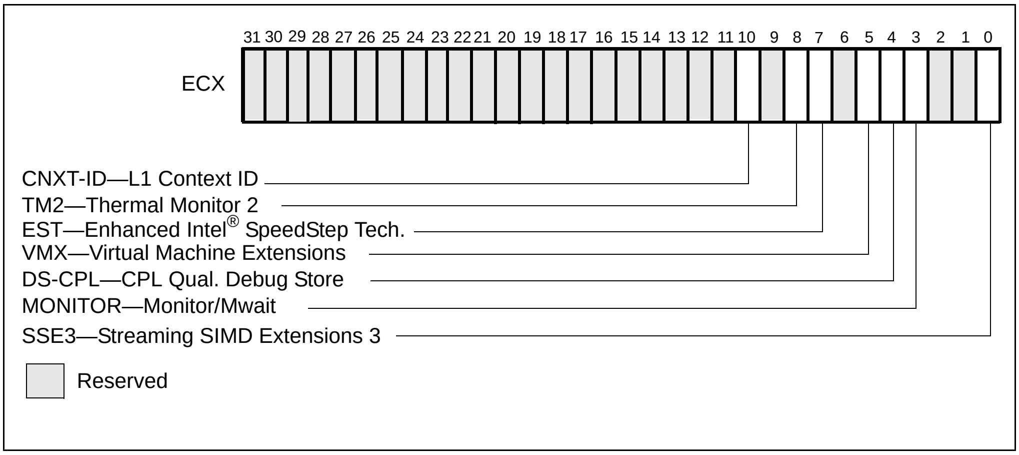

**Figure 1-2. CPUID Extended Feature Information ECX**

The VMX architecture is designed to be extensible so that future processors can support features not present initially and not described in this document. The availability of such features is reported to software using a set of capability MSRs (see Chapter 6).

## 1.7 ENABLING AND ENTERING VMX OPERATION

Before system software can enter VMX operation, it must enable it by setting CR4.VMXE[bit 13] = 1. VMX operation can then be 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 1.8). System software can leave 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 0000003AH). This MSR is cleared to zero when a logical processor is reset. The relevant bits of the MSR are described below:

- 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. Once the lock bit is set, the MSR cannot be modified until a power-up reset condition.
- Bit 2 enables VMXON. If this bit is clear, VMXON causes a general-protection exception.

Before executing VMXON, software should allocate a naturally aligned 4KB region of memory that a logical processor may use to support VMX operation.[1] This region is called the **VMXON region**. The physical address of the VMXON region (called the **VMXON pointer**) is provided in an operand to VMXON. Section 2.10.4 details how software should initialize and access the VMXON region.

## 1.8 RESTRICTIONS ON VMX OPERATION

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

- VMX operation restricts the values that may be loaded in registers CR0 and CR4. The following bits must be 1: CR0.PE, CR0.NE, CR0.PG, and CR4.VMXE. VMXON fails if any of these bits are clear (see "VMXON—Enter VMX Operation" on page 7-26). Any attempt to clear these bits during VMX operation (including VMX root operation) using the MOV CR instruction causes a general-protection exception. These bits cannot be cleared by VM entry or VM exit.

  CR0.PE and CR0.PG restrictions 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. If a VMM is to support guest software that expects to run in unpaged protected mode or in real-address mode, the VMM must support emulation of these modes. A VMM can use "identity" page tables to emulate unpaged protected mode and can use virtual-8086 mode as part of a strategy to emulate real-address mode.

  Future processors may differ with regard to bits in CR0 and CR4 that are fixed while in VMX operation. The requirements imposed by a particular processor is reported to software using VMX capability MSRs (see Section 6.6 and Section 6.7).

- VMXON fails if a logical processor is in A20M mode (see "VMXON—Enter VMX Operation" on page 7-26). Once the processor is in VMX operation, A20M 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 3.2).

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

# CHAPTER 2 VIRTUAL-MACHINE CONTROL STRUCTURE

## 2.1 OVERVIEW

The virtual-machine control data structure (VMCS) is defined for VMX operation. The VMCS manages transitions in and out of VMX non-root operation (VM entries and VM exits) as well as processor behavior in VMX non-root operation. A VMCS can be manipulated by the new instructions VMCLEAR, VMPTRLD, VMREAD, and VMWRITE.

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.

A logical processor associates with each VMCS a 4KB region in memory called the **VMCS region**.[1] Software references a VMCS by using the 64-bit physical address of this region; such an address is called a **VMCS pointer**. Every VMCS pointer must be 4KB-aligned (bits 11:0 must be zero). In addition, the pointer must not set bits beyond the processor's physical-address width.[2]

A logical processor may maintain any number of active VMCSs, at most one of which is the current VMCS:

- Software makes a VMCS **active** by executing VMPTLRD with the address of the VMCS. The processor may optimize VMX operation by maintaining the state of an active VMCS in memory, on the processor, or both. Software should not make a VMCS active on more than one logical processor (see Section 2.10.1 for how to migrate a VMCS from one logical processor to another). Software makes a VMCS inactive by executing VMCLEAR with the address of the VMCS. A logical processor will not use an inactive VMCS or maintain its state on the processor.

  If VMXOFF is executed while a VMCS is active, the VMCS data in the corresponding VMCS region are undefined after execution of VMXOFF. Software can avoid this problem by avoiding execution of VMXOFF while any VMCS is active.

- Software makes a VMCS **current** by executing VMPTLRD with the address of the VMCS; that address is loaded into the **current-VMCS pointer**. The VMX instructions VMLAUNCH, VMPTRST, VMREAD, VMRESUME, and VMWRITE operate on the current VMCS. In particular, the VMPTRST instruction stores the current-VMCS pointer into a specified memory location (it stores the value FFFFFFFF_FFFFFFFFH if there is no current VMCS). A VMCS remains current until either software executes VMPTRLD with

1. Future implementations may use VMCS regions of a different size. Software should consult the VMX capability MSR VMX_BASIC to determine the size of the VMCS region (see Section 6.1).
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.

the address of a different VMCS (which then becomes the current VMCS) or software executes VMCLEAR with the address of the current VMCS (after which there is no current VMCS).

This document frequently uses the term “the VMCS” to refer to the current VMCS.

## 2.2 FORMAT OF THE VMCS REGION

A VMCS region comprises 4KB contiguous bytes. The format of a VMCS region is given in Table 2-1.

**Table 2-1. Format of the VMCS Region**

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

The first 4 bytes of the VMCS region contain the **VMCS revision identifier**. Processors that maintain VMCS data in different formats (see below) use different VMCS revision identifiers. These identifiers enable software to avoid using a VMCS region formatted for one processor on a processor that uses a different format.

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 VMX_BASIC (see Section 6.1).

The next 4 bytes of the VMCS region are used for the **VMX-abort indicator**. The contents of these bytes do not control processor operation in any way. A logical processor writes a non-zero value into these bytes if a VMX abort occurs (see Section 5.7). 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 2.3 through Section 2.9.

To ensure proper behavior in VMX operation, software should maintain the VMCS region and related structures (enumerated in Section 2.10.3) in writeback cacheable memory. Future implementations may allow or require a different memory type. Software should consult the VMX capability MSR VMX_BASIC (see Section 6.1).

## 2.3 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.

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

## 2.4 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 4.3.2) and stored into these fields on every VM exit (see Section 5.3).

### 2.4.1 Guest Register State

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

- Control registers CR0, CR3, and CR4 (64 bits each).
- Debug register DR7 (64 bits).
- RSP, RIP, and RFLAGS (64 bits each).[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). The base-address fields for CS, SS, DS, and ES have only 32 architecturally-defined bits; nevertheless, the corresponding VMCS fields have 64 bits.
  - Segment limit (32 bits). The limit field is always a measure in bytes.

1. This document uses the notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because implementations of VMX also support Intel® EM64T. In a few places, notation such as EAX is used to refer to lower 32 bits of the indicated register.

— **Access rights** (32 bits). The format of this field is given in Table 2-2 and detailed as follows:

- The low 16 bits correspond to bits 23:8 of the upper 32 bits of a 64-bit segment descriptor. While bits 19:16 of code-segment and data-segment descriptors correspond to the upper 4 bits of the segment limit, the corresponding bits (bits 11:8) are reserved in this VMCS field.
- Bit 16 indicates an **unusable segment**. Attempts to use such a segment fault except in 64-bit mode. In general, a segment register is unusable if it has been loaded with a null selector.[1]
- Bits 31:17 are reserved.

**Table 2-2. 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)<br>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.

1. There are a few exceptions to this general 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 5.14 (Exception and Interrupt Reference) of *IA-32 Intel® Architecture Software Developer’s Manual, Volume 3*. In contrast, the TR register is usable after processor reset despite having a null selector; see Table 9-1 in Section 9.1 (Initialization Overview) of *IA-32 Intel® Architecture Software Developer’s Manual, Volume 3*.

Note that the value of the DPL field for SS is always equal to the logical processor's current privilege level (CPL).[1]

- The following fields for each of the registers GDTR and IDTR:
  - — Base address (64 bits).
  - — Limit (32 bits). The limit fields contain 32 bits even though these fields are specified as only 16 bits in the architecture.
- The MSRs IA32_DEBUGCTL (64 bits), IA32_SYSENTER_CS (32 bits), IA32_SYSENTER_ESP (64 bits), and IA32_SYSENTER_EIP (64 bits).

## 2.4.2 Guest Non-Register State

In addition to the register state described in Section 2.4.1, 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:[2]

  - — 0: **Active**. The logical processor is executing instructions normally.
  - — 1: **HLT**. The logical processor is inactive because it executed the HLT instruction.
  - — 2: **Shutdown**. The logical processor is inactive because it incurred a **triple fault**[3] 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 VMX_MISC (see Section 6.5) 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 2-3.

---

1. In protected mode, CPL is also associated with the RPL field in the CS selector. However, the RPL fields are not meaningful in real-address mode or in virtual-8086 mode.
2. Execution of the MWAIT instruction may put a logical processor into an inactive state. However, this VMCS field never reflects this state. See Section 5.1.
3. A *triple fault* occurs when a logical processor encounters an exception while attempting to deliver a double fault.

**Table 2-3. 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 *IA-32 Intel® Architecture Software Developer's Manual, Volume 2B*.<br><br>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 Chapters 3 and 4 of the *IA-32 Intel® Architecture Software Developer's Manual, Volumes 2A & 2B* and Section 5.8.3 ("Masking Exceptions and Interrupts When Switching Stacks") in the *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.<br><br>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 | Reserved | VM entry will fail if this bit is not 0. See Section 4.3.1.5. |
| 3 | Blocking by NMI | See Section 5.7.1 ("Handling Multiple NMIs") in the *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.<br><br>Delivery of a non-maskable interrupt (NMI) blocks subsequent NMIs until the next execution of IRET (see Section 3.3 of this document regarding 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. |
| 31:4 | Reserved | VM entry will fail if these bits are not 0. See Section 4.3.1.5. |

- **Pending debug exceptions** (64 bits). IA-32 processors may recognize one or more debug exceptions without immediately delivering them.[1] This field contains information about such exceptions. Details and the format of this field is given in Table 2-4.
- **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 4.3.1.5).

1. For example, execution of a MOV to SS or a POP to SS may inhibit some debug exceptions for one instruction. See Section 5.8.3 ("Masking Exceptions and Interrupts When Switching Stacks") of *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.

   In addition, certain events incident to an instruction (for example, INIT) may take priority over debug traps generated by that instruction. See Table 5-2 ("Priority Among Simultaneous Exceptions and Interrupts") in the *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.

**Table 2-4. 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 its corresponding enabling bit in DR7 is not set. |
| 11:4 | Reserved | VM entry will fail if these bits are not 0. See Section 4.3.1.5. |
| 12 | Enabled breakpoint | When set, this bit indicates that at least one data or I/O breakpoint was met and was enabled in DR7. |
| 13 | Reserved | VM entry will fail if this bit is not 0. See Section 4.3.1.5. |
| 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 will fail if these bits are not 0. See Section 4.3.1.5. |

## 2.5 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 5.5).

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

- CR0, CR3, and CR4 (64 bits each).
- RSP and RIP (64 bits each).
- 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).
- The MSRs IA32_SYSENTER_CS (32 bits), IA32_SYSENTER_ESP (64 bits), and IA32_SYSENTER_EIP (64 bits).

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 5.5 for details of how state is loaded on VM exits.

## 2.6 VM-EXECUTION CONTROL FIELDS

The VM-execution control fields govern VMX non-root operation. These are described in Section 2.6.1 through Section 2.6.8.

## 2.6.1 Pin-Based VM-Execution Controls

The pin-based VM-execution controls constitute a 32-bit vector that governs the handling of asynchronous events (interrupts). There are two pin-based VM-execution controls currently defined:

- Bit 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.
- Bit 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 3.3).

All other bits in this field are reserved as follows: bits 31:5 are reserved to 0; bit 1, bit 2, and bit 4 are reserved to 1.[1] Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 4.2).

## 2.6.2 Processor-Based VM-Execution Controls

The processor-based VM-execution controls are a 32-bit vector that governs the handling of synchronous events, mainly those caused by the execution of specific instructions.[2] Table 2-5 lists the controls supported. See Chapter 3 for complete details of how these controls affect processor behavior in VMX non-root operation.

**Table 2-5. Definitions of 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 2.4.2). |
| 3 | Use TSC offsetting | This control determines whether executions of RDTSC return a value modified by the TSC offset field (see Section 3.3). |
| 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 cause VM exits. |

1. Software may consult the VMX capability MSR VMX_PINBASED_CTLS (see Section 6.2) to determine how it should set the reserved bits.
2. Some instructions cause VM exits regardless of the settings of the processor-based VM-execution controls (see Section 3.1.2), as do task switches (see Section 3.2).

Table 2-5. Definitions of Processor-Based VM-Execution Controls (Contd.)

| Bit Position(s) | Name | Description |
|---|---|---|
| 19 | CR8-load exiting | This control determines whether executions of MOV to CR8 cause VM exits. |
| 20 | CR8-store exiting | This control determines whether executions of MOV from CR8 cause VM exits. |
| 21 | Use TPR shadow | Setting this control to 1 activates the TPR shadow, which is maintained in a page of memory addressed by the virtual-APIC address. See Section 3.3. |
| 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.<br>This control is ignored if the “activate I/O bitmaps” control is 1. |
| 25 | Activate I/O bitmaps | This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 2.6.4 and Section 3.1.3).<br>For this control, “0” means “do not activate I/O bitmaps” and “1” means “activate I/O bitmaps.” If the I/O bitmaps are activated, the setting of the “unconditional I/O exiting” is ignored. |
| 29 | MONITOR exiting | This control determines whether executions of MONITOR cause VM exits. |
| 30 | PAUSE exiting | This control determines whether executions of PAUSE cause VM exits. |

Other bits in this field are reserved as follows: bit 0, bits 18:17, bit 22, bits 28:27, and bit 31 are reserved to 0; bit 1, bits 6:4, bit 8, bits 16:13, and bit 26 are reserved to 1.[1] Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 4.2).

## 2.6.3 Exception Bitmap

The **exception bitmap** is a 32-bit field that contains one bit for each IA-32 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 3.2 for details.

1. Software may consult the VMX capability MSR VMX_PROCBASED_CTLS (see Section 6.2) to determine how it should set the reserved bits.

### 2.6.4 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 4KB 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 “activate 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 3.1.3 for details. If the bitmaps are used, their addresses must be 4KB-aligned.

### 2.6.5 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 instruction. The signed value 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 3 for a detailed treatment of the behavior of RDTSC in VMX non-root operation.

### 2.6.6 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).

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 3 for details regarding how these fields affect VMX non-root operation.

### 2.6.7 CR3-Target Controls

The VM-execution control fields include a set of 4 64-bit **CR3 target values** and a 32-bit **CR3-target count**. Executions of MOV to CR3 in VMX non-root operation do not cause a VM exit if its source operand matches one of these values; if the CR3-target count is less than 4, then not all the CR3-target values are considered.

There are no limitations on the values that can be written for the CR3 target values. VM entry fails (see Section 4.2) 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 VMX_MISC (see Section 6.5) to determine the number of values supported.

### 2.6.8 Controls for CR8 Accesses

The CR8 register can be used to access the task-priority register (TPR) of the logical processor's local APIC. The VMCS contains two fields that control MOV CR8 instructions if the "use TPR shadow" VM-execution control is 1:

- **Virtual-APIC page address** (64 bits). This field is the physical address of the 4KB virtual-APIC page. The virtual-APIC page contains the TPR shadow, which is read and written by the MOV CR8 instructions. The TPR shadow comprises bits 7:4 in byte 128 of the virtual-APIC page. If the "use TPR shadow" VM-execution control is 1, the virtual-APIC page address must be 4KB-aligned.
- **TPR threshold** (32 bits). Bits 3:0 of this field determine the threshold below which the TPR shadow (see previous item) cannot fall. A VM exit occurs after an execution of MOV to CR8 that reduces the TPR shadow below this value.

Note that the TPR in the local APIC can also be accessed using memory-mapped I/O. These controls does not affect access made in that way. They affect only MOV CR8 instructions (see Section 3.1.3 and Section 3.3).

## 2.7 VM-EXIT CONTROL FIELDS

The VM-exit control fields govern the behavior of VM exits. They are discussed in Section 2.7.1 and Section 2.7.2.

### 2.7.1 VM-Exit Controls

The VM-exit controls constitute a 32-bit vector that governs the basic operation of VM exits. Two VM-exit controls are currently defined:

- Bit 9: **host address-space size**. 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]

1. Since Intel® EM64T 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.

- Bit 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.

All other bits in this field are reserved as follows: bit 12 and bits 31:18 are reserved to 0; bits 8:0, bits 11:10, bits 14:13, and bits 17:16 are reserved to 1.[1] Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 4.2).

## 2.7.2 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.[2] 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 2-6. If the VM-exit MSR-store count is not zero, the address must be 16-byte aligned.

**Table 2-6. Format of an MSR Entry**

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

See Section 5.4 for how this area is used on VM exits.

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

- **VM-exit MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM exit. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM exit.[3]

1. Software may consult the VMX capability MSR VMX_EXIT_CTLS (see Section 6.3) to determine how it should set the reserved bits.
2. Future implementations may allow more MSRs to be stored reliably. Software should consult the VMX capability MSR VMX_MISC to determine the number supported (see Section 6.5).
3. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR VMX_MISC to determine the number supported (see Section 6.5).

- **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 2-6). If the VM-exit MSR-load count is not zero, the address must be 16-byte aligned.

See Section 5.6 for how this area is used on VM exits.

## 2.8 VM-ENTRY CONTROL FIELDS

The VM-entry control fields govern the behavior of VM entries. They are discussed in Section 2.8.1 through Section 2.8.3.

### 2.8.1 VM-Entry Controls

The VM-entry controls constitute a 32-bit vector that governs the basic operation of VM entries.

There is one VM-entry control currently defined: bit 9, **IA-32e mode guest**. This bit determines whether the logical processor is in IA-32e mode after VM entry. Its value is loaded into IA32_EFER.LMA and IA32_EFER.LME as part of VM entry.[1]

All other bits in this field are reserved as follows: bits 11:10 and bits 31:13 are reserved to 0; bits 8:0 and bit 12 are reserved to 1.[2] Failure to set reserved bits properly causes subsequent VM entries to fail (see Section 4.2).

### 2.8.2 VM-Entry Controls for MSRs

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

- **VM-entry MSR-load count** (32 bits). This field contains the number of MSRs to be loaded on VM entry. It is recommended that this count not exceed 512. Otherwise, unpredictable processor behavior (including a machine check) may result during VM entry.[3]
- **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 2-6. If the VM-entry MSR-load count is not zero, the address must be 16-byte aligned.

See Section 4.4 for details of how this area is used on VM entries.

1. Since Intel® EM64T 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.
2. Software may consult the VMX capability MSR VMX_ENTRY_CTLS (see Section 6.4) to determine how it should set the reserved bits.
3. Future implementations may allow more MSRs to be loaded reliably. Software should consult the VMX capability MSR VMX_MISC to determine the number supported (see Section 6.5).

## 2.8.3 VM-Entry Controls for Event Injection

VM entry can be configured to conclude by delivering an event through the guest 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 of the event to be injected. Table 2-7 gives the format of this field.

**Table 2-7. Format of the VM-Entry Interruption-Information Field**

| Bit Position(s) | Content |
|---|---|
| 7:0 | Vector of interrupt or exception |
| 10:8 | Interruption type:<br>0: External interrupt<br>1: Reserved<br>2: Non-maskable interrupt (NMI)<br>3: Hardware exception<br>4: Software interrupt<br>5: Privileged software exception<br>6: Software exception<br>7: Reserved |
| 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.

  — 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.

  — 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.

- **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 4.5 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.

## 2.9 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 Chapter 7).

### 2.9.1 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 2-8.

**Table 2-8. Format of Exit Reason**

| Bit Position(s) | Contents |
|---|---|
| 15:0 | Basic exit reason |
| 30:16 | Reserved (cleared to 0) |
| 31 | VM-entry failure (0 = true VM exit; 1 = VM-entry failure) |

  Because some VM-entry failures load processor state from the host-state area (see Section 4.7), software must be able to distinguish such cases from true VM exits. Bit 31 is used for that purpose. 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 A lists basic exit reasons.

- **Exit qualification** (64 bits). 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; INVLPG; 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 5.2.1 for details.

### 2.9.2 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. The format of this field is given in Table 2-9.

**Table 2-9. Format of the VM-Exit Interruption-Information Field**

| Bit Position(s) | Content |
|---|---|
| 7:0 | Vector of interrupt or exception |
| 10:8 | Interruption type:<br>0: External interrupt<br>1: Not used<br>2: Non-maskable interrupt (NMI)<br>3: Hardware exception<br>4 – 5: Not used<br>6: Software exception<br>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 5.2.2 explains the interruption-type field and provides details of how these fields are saved on VM exits.

### 2.9.3 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. This information is provided in the following fields:

- **IDT-vectoring information** (32 bits): see Table 2-10. The individual fields are defined as they were for the VM-exit interruption-information field (see Section 2.9.2). However, in this case, they refer not to the cause of the VM exit but to the event that was being delivered in VMX non-root operation when the VM exit occurred. The type field may receive value 4 (software interrupt) if the VM exit occurred during the delivery of a software interrupt. In this case, the vector field receives the interrupt number.

**Table 2-10. Format of the IDT-Vectoring Information Field**

| Bit Position(s) | Content |
|---|---|
| 7:0 | Vector of interrupt or exception |
| 10:8 | Interruption type:<br>0: External interrupt<br>1: Not used<br>2: Non-maskable interrupt (NMI)<br>3: Hardware exception<br>4: Software interrupt<br>5: Not used<br>6: Software exception<br>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). On VM exits that set bits 31 and 11 in the IDT-vectoring information field, this field receives the error code that would have been delivered onto the stack by the event that was being delivered through the IDT (see above) at the time of the VM exit.

See Section 5.2.3 explains the interruption-type field and provides details of how these fields are saved on VM exits.

## 2.9.4 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. See Section 5.2.4 for details of when and how this field is used.
- **Guest linear address** (64 bits). This field is used in the following cases:
  - — VM exits due to attempts to execute LMSW with a memory operand. This field receives the linear address of that operand.
  - — VM exits due to attempts to execute INS or OUTS. The field receives the value of the linear address generated by ES:(E)DI (for INS) or segment:(E)SI (for OUTS) at the time the instruction started.

  See Section 5.2.4 for details of when and how this field is used.

- **VMX-instruction information** (32 bits). For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON, this field receives details about the instruction that caused the VM exit. The format of the field is given in Table 2-11.

**Table 2-11. Format of the VMX-Instruction Information Field**

| Bit Position(s) | Content |
|---|---|
| 1:0 | Scaling:<br>0: no scaling<br>1: scale by 2<br>2: scale by 4<br>3: scale by 8<br>Undefined for register instructions (bit 10 is set) or for memory instructions with no index register (bit 10 is clear and bit 22 is set) |
| 2 | Reserved (cleared to 0) |
| 6:3 | Reg1:<br>0 = RAX<br>1 = RCX<br>2 = RDX<br>3 = RBX<br>4 = RSP<br>5 = RBP<br>6 = RSI<br>7 = RDI<br>8–15 represent R8–R15, respectively<br>Undefined for memory instructions (bit 10 is clear) |
| 9:7 | Address size:<br>0: 16-bit<br>1: 32-bit<br>2: 64-bit<br>Other values not used<br>Undefined for register instructions (bit 10 is set) |
| 10 | Mem/Reg (0 = memory; 1 = register)<br>Note that VMCLEAR, VMPTRLD, VMPTRST, and VMXON are always memory instructions and thus clear this bit. |
| 14:11 | Reserved (cleared to 0) |
| 17:15 | Segment register:<br>0: ES<br>1: CS<br>2: SS<br>3: DS<br>4: FS<br>5: GS<br>Other values unused<br>Undefined for register instructions (bit 10 is set) |
| 21:18 | IndexReg (encoded as Reg1 above)<br>Undefined if bit 22 is set or undefined |
| 22 | IndexReg invalid (0 = valid; 1 = invalid)<br>Undefined for register instructions (bit 10 is set) |

**Table 2-11. Format of the VMX-Instruction Information Field (Contd.)**

| Bit Position(s) | Content |
|---|---|
| 26:23 | BaseReg (encoded as Reg1 above)<br>Undefined if bit 27 is set or undefined |
| 27 | BaseReg invalid (0 = valid; 1 = invalid)<br>Undefined for register instructions (bit 10 is set) |
| 31:28 | Reg2 (same encoding as Reg1 above)<br>Undefined on VM exits due to VMCLEAR, VMPTRLD, VMPTRST, and VMXON |

### 2.9.5 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 any error encountered by a non-faulting execution of one of the VMX instructions. See Section 7.2 for details of its use and Appendix B for a listing of error numbers.

## 2.10 SOFTWARE ACCESS TO THE VMCS AND RELATED STRUCTURES

This section details guidelines that software should observe when accessing a VMCS and related structures as well as the potential consequences for failing to follow such guidelines.

### 2.10.1 Software Access to the Virtual-Machine Control Structure

To ensure proper processor behavior, software should observe certain guidelines when accessing 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).

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 performing such accesses:

- 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 lead to unpredictable behavior. Any or all of the

following may occur: (1) VM entries may fail for unexplained reasons or may load undesired processor state; (2) the processor may not correctly support VMX non-root operation as documented in Chapter 3 and may generate unexpected VM exits; and (3) VM exits may load undesired processor state, save incorrect state into the VMCS, or cause the logical processor to transition to a shutdown state.

Software can avoid such problems 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.

Software should use the VMREAD and VMWRITE instructions to access the different fields in the current VMCS (see Section 2.10.2 for details).

Software should initialize all 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.

### 2.10.2 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 7 for details of the operation 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. The structure is summarized in Table 2-12.

**Table 2-12. Structure of VMCS Component Encoding**

| Bit Position(s) | Contents |
|---|---|
| 31:15 | Reserved (must be 0) |
| 14:13 | Width:<br>0: 16-bit<br>1: full 64-bit<br>2: 32-bit<br>3: natural 64-bit |
| 12 | Reserved (must be 0) |
| 11:10 | Type:<br>0: control<br>1: read-only data<br>2: guest state<br>3: host state |
| 9:1 | Index |
| 0 | Access type (0 = full; 1 = high); must be full for 16-bit, 32-bit, and natural 64-bit fields |

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

- Field width. Bits 14:13 encode the width of the field (which may be 16 bits, 32 bits, full 64 bits, or natural 64 bits). A field is **full 64 bits** if a 32-bit VM monitor would need to access the entire field; it is **natural 64 bits** if a 32-bit VM monitor would need to access only the low 32 bits. For example, the field for RIP in the host-state area is natural 64 bits because a 32-bit VM monitor does not need to access the high 32 bits of the field (they would always be zero); in contrast, the VM-exit MSR-store address requires full 64 bits because it is a physical address that may be 64 bits in any processor mode.

  Full 64-bit fields 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 those whose field width is full 64-bit (see above). A VMREAD or VMWRITE using an encoding with this bit cleared to 0 accesses the entire field. For a full 64-bit field, a VMREAD or VMWRITE using an encoding with this bit set to 1 accesses only the high 32 bits of the field.

Appendix C 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 (full and natural) 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 (full and natural) using the full access type in 64-bit mode.
  - 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.
- Full 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 modify a full 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).

### 2.10.3 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). Note that, 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 2.10.1).

### 2.10.4 The VMXON Region

Before executing VMXON, software allocates a region of memory (called the VMXON region) that the logical processor may use to support VMX operation. The physical address of this region (the VMXON pointer) is provided in an operand to VMXON. Like VMCS pointers, the VMXON pointer must be 4KB-aligned (bits 11:0 must be zero); in addition, the pointer must not set any bits beyond the processor’s physical-address width.

Before executing VMXON, software should write the VMCS revision identifier (see Section 2.2) 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 2.10.1).

## 2.11 USING VMCLEAR TO INITIALIZE A VMCS REGION

A processor may use the VMCS data portion of a VMCS region to maintain implementation-specific information about the VMCS. When software first allocates a region of memory for use as a VMCS region, the data in that region may be interpreted in an implementation-specific manner. 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.

A logical processor uses the VMCS region to maintain the **launch state** of the corresponding VMCS. The launch state may be **clear** or **launched**. The VMCLEAR instruction puts the VMCS referenced by its operand into the clear state. The VMLAUNCH instruction requires a VMCS whose launch state is clear and changes its launch state to launched. The VMRESUME instruction requires a VMCS whose launch state is launched. 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 read it (it cannot be read using VMREAD). Improper software usage (for example, software writing to the VMCS data of an active VMCS) may leave the launch state undefined.

The following software usage is consistent with these limitations:

- VMCLEAR should be executed for a VMCS before it is used for VM entry.
- 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 2.10.1), 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.

# CHAPTER 3
# VMX NON-ROOT OPERATION

This chapter describes the differences between VMX non-root operation and ordinary processor operation with special attention to causes of VM exits:

- Section 3.1 identifies the instructions that can cause VM exits.
- Section 3.2 identifies other sources of VM exits.
- Section 3.3 describes how some instructions, when they do not cause VM exits, operate differently in VMX non-root operation.
- Section 3.4 describes other changes to processor execution in VMX non-root operation.

## 3.1 INSTRUCTIONS THAT CAUSE VM EXITS

Certain instructions may cause VM exits if executed in VMX non-root operation. Unless otherwise specified, these 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 (for example, the value of RIP saved in the guest-state area of the VMCS references the instruction causing the VM exit). Section 5.1 provides details of how architectural state is updated before a VM exit.

Section 3.1.1 defines the prioritization between IA-32 faults and VM exits for instructions subject to both. Section 3.1.2 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 3.1.3 identifies instructions that cause VM exits depending on the settings of certain VM-execution control fields.

### 3.1.1 Relative Priority of IA-32 Faults and VM Exits

The following general principles describe the ordering between existing IA-32 faults and VM exits:

- Certain exceptions have priority over VM exits. These include invalid-opcode exceptions, faults based on privilege level, and general-protection exceptions 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.[1]
- Faults incurred while fetching instruction operands have priority over VM exits that are conditioned based on the contents of those operands (see the case of LMSW in Section 3.1.3).

1. MOV DR is an exception to this rule; see Section 3.1.3.

- Fault-like VM exits have priority over general-protection 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 3.1.2 or Section 3.1.3 below identifies an instruction execution that may lead to a VM exit, it is assumed that the instruction does not incur a fault that would take priority over a VM exit.

### 3.1.2 Instructions That Cause VM Exits Unconditionally

The following instructions cause VM exits whenever they are executed in VMX non-root operation: CPUID, INVD, MOV from CR3, RDMSR, WRMSR, and the new instructions VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, and VMXON.

### 3.1.3 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 "activate 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 "activate I/O bitmaps" VM-execution control is 0, the instruction causes a VM exit.
  - If the "activate 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 2.6.4). 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 "activate I/O bitmaps" VM-execution control is 1.)
- INLVPG. The INLVPG instruction causes a VM exit if the "INLVPG exiting" VM-execution control is 1.
- LMSW. In general, the LMSW instruction causes a VM exit if it would write, for any bit set in the low 4 bits of the CR0 guest/host mask, a value different than the corresponding

bit in the CR0 read shadow. Note that 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 CR8. The MOV from CR8 instruction causes a VM exit if the "CR8-store exiting" VM-execution control is 1.
- MOV to CR0. The MOV to CR0 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR0 guest/host mask, the corresponding bit in the CR0 read shadow. (If every bit is clear in the CR0 guest/host mask, MOV to CR0 cannot cause a VM exit.)
- MOV to CR3. The MOV to CR3 instruction causes a VM exit unless the value of its source operand is equal to one of the CR3-target values specified in the VMCS. Note that, 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.
- MOV to CR4. The MOV to CR4 instruction causes a VM exit unless the value of its source operand matches, for the position of each bit set in the CR4 guest/host mask, the corresponding bit in the CR4 read shadow.
- MOV to CR8. The MOV to CR8 instruction causes a VM exit if the "CR8-load exiting" VM-execution control is 1. Note that, 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 3.3), 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 3.1.1; they take priority over all faults that may occur in the execution of MOV DR.
- MWAIT. The MWAIT instruction causes a VM exit if the "MWAIT exiting" VM-execution control is 1.
- PAUSE. The PAUSE instruction causes a VM exit if the "PAUSE exiting" VM-execution control is 1.
- RDPMC. The RDPMC instruction causes a VM exit if the "RDPMC exiting" VM-execution control is 1.
- RDTSC. The RDTSC instruction causes a VM exit if the "RDTSC exiting" VM-execution control is 1.

The MOV to CR8 instruction may cause a "trap-like" VM exit. This means that the instruction completes before the VM exit occurs and that processor state is updated by the instruction (for example, the value of RIP saved in the guest-state area of the VMCS references the next instruction). Specifically, a VM exit occurs after execution of MOV to CR8 if the following are true:

- The “CR8-load exiting” VM-execution control is 0.
- The “use TPR shadow” VM-execution control is 1.
- The execution of MOV to CR8 reduces the value of the TPR shadow below that of the TPR threshold (see Section 2.6.8 and Section 3.3).

## 3.2 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 2.6.3). 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 wants VM exits on all page faults, it could set bit 14 in the exception bitmap to 1 and set the page-fault error-code mask and match fields each to 00000000H. If it does not require VM exits on page faults, it could set bit 14 in the exception bitmap to 1, set the page-fault error-code mask field to 00000000H, and set the page-fault error-code match field to FFFFFFFFH.
- 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.)
- INITs. Assertions of the INIT signal cause VM exits. A logical processor performs none of the normal operations associated with these events: they 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, INITs 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 3.4.2.

In addition, there is one control that causes VM exits based on the readiness of guest software to receive an external interrupt:

- 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 2-3). Such a VM exit occurs immediately after VM entry if these conditions hold at that time (see Section 4.6.4).

  A pending non-maskable interrupt (NMI) takes priority over a VM exit caused by the "interrupt-window exiting" VM-execution control. VM exits caused by this control take priority over any pending external interrupts.

  Such 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. They do not occur if the logical processor is in the shutdown state or the wait-for-SIPI state.

## 3.3 CHANGES TO INSTRUCTION BEHAVIOR IN VMX NON-ROOT OPERATION

The behavior of some instructions is changed in VMX non-root operation; some of these changes are determined by the settings of certain VM-execution control fields. The following items detail these 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).
  - 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 (see Section 3.1.3).

- IRET. Behavior of IRET with regard to the blocking by NMI (see Table 2-3) is determined by the setting of the “NMI exiting” VM-execution control:
  - If the control is 0, IRET operates normally and unblocks NMIs.
  - If the control is 1, IRET does not affect blocking by NMI.
- LMSW. An execution of LMSW that does not cause a VM exit (see Section 3.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask.
- 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.
- 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.
- MOV from CR8. Behavior of the MOV from CR8 instruction 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 128 of the page referenced by the virtual-APIC page address (see Section 2.6.8).
  - If the “CR8-store exiting” VM-execution control is 1, MOV from CR8 causes a VM exit (see Section 3.1.3); 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 3.1.3) leaves unmodified any bit in CR0 corresponding to a bit set in the CR0 guest/host mask. It causes a general-protection exception if it attempts to clear PE, NE, or PG (if a bit corresponding to one of these is clear in the CR0 guest/host mask); see Section 1.8.
- MOV to CR4. An execution of MOV to CR4 that does not cause a VM exit (see Section 3.1.3) leaves unmodified any bit in CR4 corresponding to a bit set in the CR4 guest/host mask. It causes a general-protection exception if it attempts to clear VMXE (if the corresponding bit is clear in the CR4 guest/host mask); see Section 1.8.

- MOV to CR8. Behavior of the MOV to CR8 instruction 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.
  - 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 bytes 128 of the page referenced by the virtual-APIC page address (see Section 2.6.8). Such a store may cause a VM exit to occur after it completes (see Section 3.1.3).
  - If the “CR8-load exiting” VM-execution control is 1, MOV to CR8 causes a VM exit (see Section 3.1.3); the “use TPR shadow” VM-execution control is ignored in this case.
- RDTSC. Behavior of the RDTSC instruction is determined by the settings of the “RDTSC exiting” and “use TSC offsetting” VM-execution controls:
  - If both controls are 0, RDTSC operates normally.
  - If the “RDTSC exiting” VM-execution control is 0 and the “use TSC offsetting” VM-execution control is 1, 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 (see Section 3.1.3).

## 3.4 OTHER CHANGES IN VMX NON-ROOT OPERATION

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

### 3.4.1 Event Blocking

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

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

### 3.4.2 Treatment of Task Switches

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

1. If a task gate is being used, appropriate checks are made on its P bit and on the proper values of the relevant privilege fields. The following cases detail the privilege checks performed:
   - If CALL, INT *n*, INT3, INTO, or JMP accesses a task gate in IA-32e mode, a general-protection exception occurs.
   - 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.
   - If CALL or JMP accesses a TSS descriptor directly in IA-32e mode, a general-protection exception occurs.
   - If CALL or JMP accesses a TSS descriptor directly outside IA-32e mode, privilege levels are checked on the TSS descriptor.
   - If a non-maskable interrupt (NMI), an exception other than breakpoint exceptions (#BP; generated by INT3) and overflow exceptions (#OF; generated by INTO), or an external interrupt accesses a task gate in the IDT in IA-32e mode, a general-protection exception occurs.
   - 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 in legacy mode, no privilege checks are performed.
   - If IRET is executed with RFLAGS.NT = 1 in IA-32e mode, a general-protection exception occurs.
   - 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 5.2. 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 5.2.

# CHAPTER 4
# 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 4.1).
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 4.2).
3. The following may be performed in parallel and in any order (Section 4.3):
   - 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 (as extended by Intel EM64T).
   - Processor state is loaded from the guest-state area and based on the VM-entry controls.
   - Address-range monitoring is cleared.
4. MSRs are loaded from the VM-entry MSR-load area (Section 4.4).
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 4.5).

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 4.1 may generate ordinary IA-32 faults (for example, an invalid-opcode exception). Such faults are delivered normally.
- Some of the checks in Section 4.1 and all the checks in Section 4.2 cause control to pass to the instruction following the VM-entry instruction. The failure is indicated by setting RFLAGS.ZF (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 Appendix B for the error numbers.
- The checks in Section 4.3 and Section 4.4 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 4.7 for more details.

## 4.1 BASIC VM-ENTRY CHECKS

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

1. If a 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 one of which causes VM entry to fail:
    a. if there is MOV-SS blocking (see Table 2-3);
    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 Appendix B for an enumeration of error numbers.

## 4.2 CHECKS ON VMX CONTROLS AND HOST-STATE AREA

If the checks in Section 4.1 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 IA-32 as extended by Intel EM64T.

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 VMX controls or host-state area (see Appendix B).

These checks may be performed in any order. Thus, an indication by error number of one cause (for example, VMX controls) does not imply that there are not also other errors.

The checks on VMX controls and the host-state area are presented in Section 4.2.1 through Section 4.2.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

### 4.2.1 Basic Checks on VMX Controls

This section enumerates checks on VMX control fields:

- Reserved bits in the pin-based VM-execution controls must be set properly. The reserved settings are indicated in Section 2.6.1. In addition, software may consult the VMX capability MSR VMX_PINBASED_CTLS to determine the proper settings (see Section 6.2).

- Reserved bits in the processor-based VM-execution controls must be set properly. The reserved settings are indicated in Section 2.6.2. In addition, software may consult the VMX capability MSR VMX_PROCBASED_CTLS to determine the proper settings (see Section 6.2).
- Reserved bits in the VM-exit controls must be set properly. The reserved settings are indicated in Section 2.7.1. In addition, software may consult the VMX capability MSR VMX_EXIT_CTLS to determine the proper settings (see Section 6.3).
- Reserved bits in the VM-entry controls must be set properly. The reserved settings are indicated in Section 2.8.1. In addition, software may consult the VMX capability MSR VMX_ENTRY_CTLS to determine the proper settings (see Section 6.4).
- 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 VMX_MISC to determine the number of values supported (see Section 6.5).
- Fields relevant to VM-entry event injection must be set properly. These fields are the VM-entry interruption-information field (see Table 2-7), 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 (1 or 7).
  - — 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.
  - — The field's deliver-error-code bit (bit 11) is 1 if and only if the interruption type is hardware exception and 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.

## 4.2.2 Checks on Physical Addresses and Referenced Data

The VMX controls include a number of fields that contain physical addresses. These must be properly aligned and must respect the processor's physical-address width:[1]

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.

- If the "activate I/O bitmaps" VM-execution control is 1, bits 11:0 of each I/O-bitmap address must be 0. In addition, neither address should set any bits beyond the processor's physical-address width.
- If the "use TPR shadow" VM-execution control is 1, bits 11:0 of each virtual-APIC page address must be 0. In addition, the address should not set any bits beyond the processor's physical-address width.
- The following check is performed for each MSR area (VM exit MSR-store, VM exit MSR-load, VM entry MSR-load) if the corresponding MSR count field is non-zero:
  - — The lower 4 bits of the MSR storage-area address must be 0. In addition, the address should not set any bits beyond the processor's physical-address width.
  - — The address of the last byte in the MSR-storage area should not set any bits beyond the processor's physical-address width. The address of this last byte is MSR storage-area address + (MSR count * 16) – 1. (The arithmetic used for the computation uses more bits than the processor's physical-address width.)

In addition to these checks on the physical addresses, the following check is performed if the "use TPR shadow" VM-execution control is 1: the value of bits 3:0 of the TPR threshold should not be greater than the value of bits 7:4 in byte 128 on the page referenced by the virtual-APIC page address.

### 4.2.3 Checks on Host Control Registers and MSRs

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

- The CR0 field must set bit 0 (corresponding to CR0.PE), bit 5 (CR0.NE), and bit 31 (CR0.PG). Future implementations may restrict CR0 differently. Software should read the VMX capability MSRs VMX_CR0_FIXED0 and VMX_CR0_FIXED1 to determine the number of values supported (see Section 6.6).[1]
- The CR4 field must set bit 13 (corresponding to CR4.VMXE). Future implementations may restrict CR4 differently. Software should read the VMX capability MSRs VMX_CR4_FIXED0 and VMX_CR4_FIXED1 to determine the number of values supported (see Section 6.7).
- 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]
- The IA32_SYSENTER_ESP field and the IA32_SYSENTER_RIP field must each contain a canonical address.

1. The bits corresponding to NW (bit 29) and CD (bit 30) are never checked because the values of these bits are not changed by VM exit; see Section 5.5.1.
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.

### 4.2.4 Checks on Segment and Descriptor-Table Registers

The following checks are performed on fields in the host-state area corresponding 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.
- The base-address fields for FS, GS, GDTR, IDTR, and TR must contain canonical addresses.

### 4.2.5 Checks Related to Address-Space Size

The following checks related to address-space size are performed on VMX controls and fields in the host-state area:

- If a logical processor is outside IA-32e mode (if IA32_EFER.LMA = 0) at the time of VM entry, the "IA-32e mode guest" VM-entry control must be 0.
- 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.
  - 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.

## 4.3 CHECKING AND LOADING GUEST STATE

If all checks on the VMX controls and the host-state area pass (see Section 4.2), 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 as extended by Intel EM64T; (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, a logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 4.7.

### 4.3.1 Checks on the Guest State Area

This section describes checks performed on fields in the guest-state area of the VMCS. These checks may be performed in any order. The following subsections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the guest-state area.

#### 4.3.1.1 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 set bit 0 (corresponding to CR0.PE), bit 5 (CR0.NE), and bit 31 (CR0.PG). Future implementations may restrict CR0 differently. Software should read the VMX capability MSRs VMX_CR0_FIXED0 and VMX_CR0_FIXED1 to determine the number of values supported (see Section 6.6).[1]
- The CR4 field must set bit 13 (corresponding to CR0.VMXE). Future implementations may restrict CR4 differently. Software should read the VMX capability MSRs VMX_CR4_FIXED0 and VMX_CR4_FIXED1 to determine the number of values supported (see Section 6.7).
- If the “IA-32e mode guest” VM-entry control is 1, bit 5 in the CR4 field (corresponding to CR4.PAE) must be 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.[2]
- Bits 63:32 in the DR7 field must be 0.
- Bits reserved in the IA32_DEBUGCTL MSR must be 0 in the field for that register.
- The IA32_SYSENTER_ESP field and the IA32_SYSENTER_RIP field must each contain a canonical address.

#### 4.3.1.2 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.

1. The bits corresponding to NW (bit 29) and CD (bit 30) are never checked because the values of these bits are not changed by VM entry; see Section 4.3.2.1.
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.

- Any one of these registers is said to be **usable** if the unusable bit (bit 16) is 0 in the access-rights field for that register.

The following are the checks on these fields:

- Selector fields.
  - — TR. The TI flag (bit 2) must be 0.
  - — LDTR. If LDTR is usable, the TI flag (bit 2) must be 0.
  - — SS. If the guest will not be virtual-8086, the RPL (bits 1:0) must equal the RPL of the selector field for CS.
- Base-address fields.
  - — TR. 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.
    - If the guest will be virtual-8086, the address must be the selector field shifted right 4 bits.
  - — SS, DS, ES.
    - If the register is usable, bits 63:32 of the address must be zero.
    - If the guest will be virtual-8086, the address must be the selector field shifted right 4 bits.
  - — FS, GS.
    - The address must be canonical.
    - If the guest will be virtual-8086, the address must be the selector field shifted right 4 bits.
- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields. The different sub-fields are considered separately:
  - — Bits 3:0 (Type).
    - TR.
      - — 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).
    - LDTR. If LDTR is usable, the Type must be 2 (LDT).

    - CS.
        - Bit 0 of the Type must be 1 (accessed).
        - If the guest will not be virtual-8086, bit 3 of the Type must be 1 (code segment).
        - If the guest will be virtual-8086, the Type must be 3 (read/write, accessed, expand-up data segment).
    - SS.
        - If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
        - If the guest will be virtual-8086, the Type must be 3 (read/write, accessed, expand-up data segment).
    - DS, ES, FS, GS.
        - If the register is usable, bit 0 of the Type must be 1 (accessed).
        - If the register is usable and bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
        - If the guest will be virtual-8086, the Type must be 3 (read/write, accessed, expand-up data segment).
- Bit 4 (S).
    - TR. S must be 0.
    - LDTR. If LDTR is usable, S must be 0.
    - CS. S must be 1.
    - SS, DS, ES, FS, GS. If the register is usable, S must be 1.
- Bits 6:5 (DPL).
    - CS.
        - If the Type is in the range 8–11 (non-conforming code segment), the DPL must equal the RPL (bits 1:0) from the selector field.
        - If the Type is in the range 13–15 (conforming code segment), the DPL cannot be greater than the RPL from the selector field.
        - If the guest will be virtual-8086, the DPL must be 3.
    - SS.
        - If the guest will not be virtual-8086, the DPL must equal the RPL from the selector field
        - If the guest will be virtual-8086, the DPL must be 3.

    - DS, ES, FS, GS.
      - If the guest will not be virtual-8086, the register is usable, and the register's Type is in the range 0 – 11 (data segment or non-conforming code segment), then the DPL cannot be less than the RPL from the selector field
      - If the guest will be virtual-8086, the DPL must be 3.
- Bit 7 (P).
  - CS, TR. P must be 1.
  - SS, DS, ES, FS, GS, LDTR. If the register is usable, P must be 1.
- Bits 11:8 (reserved).
  - CS, TR. These bits must all be 0.
  - SS, DS, ES, FS, GS, LDTR. If the register is usable, these bits must all be 0.
- Bit 14 (D/B).
  - CS. D/B must be 0 if the guest will be virtual-8086, or if the guest will be IA-32e mode and the L bit (bit 13) in the access-rights field is 1.
  - SS, DS, ES, FS, GS. D/B must be 0 if the guest will be virtual-8086.
- Bit 15 (G).
  - CS, TR.
    - If any bit in the limit field in the range 11:0 is 0, or if the guest will be virtual-8086, G must be 0.
    - If any bit in the limit field in the range 31:20 is 1, G must be 1.
  - SS, DS, ES, FS, GS, LDTR. The following checks apply if the register is usable:
    - If any bit in the limit field in the range 11:0 is 0 or if the guest will be virtual-8086, 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).
  - TR. The unusable bit must be 0.
  - CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the unusable bit must be 0.
- Bits 31:17 (reserved).
  - CS, TR. These bits must all be 0.
  - SS, DS, ES, FS, GS, LDTR. If the register is usable, these bits must all be 0.

#### 4.3.1.3 Checks on Guest Descriptor-Table Registers

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

- The base-address fields must contain canonical addresses.
- Bits 31:16 of each limit field must be 0.

#### 4.3.1.4 Checks on Guest RIP and RFLAGS

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

- RIP.
  - — 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, 15, 5 and 3 must be 0 in the field, and reserved bit 1 must be 1. In addition, the VM flag (bit 17) must be 0 if the "IA-32e mode guest" VM-entry control is 1.
  - — The RF flag (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.

#### 4.3.1.5 Checks on Guest Non-Register State

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

- Activity state.
  - — The activity-state field must contain a value in the range 0 – 3, indicating an activity state supported by the implementation (see Section 2.4.2). Future processors may include support for other activity states. Software should read the VMX capability MSR VMX_MISC (see Section 6.5) 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.[2]

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. As noted in Section 2.4.1, SS.DPL corresponds to the logical processor's current privilege level (CPL).

- — 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 whose injection is allowed for the different activity states:
  - Active. Any interruption is allowed.
  - HLT. The only events allowed are those with interruption type external interrupt or non-maskable interrupt (NMI) and those with interruption type hardware exception and vector 1 (debug exception) or vector 18 (machine-check exception).
  - Shutdown. Only NMIs and machine-check exceptions are allowed.
  - Wait-for-SIPI. No interruptions are allowed.

- Interruptibility state.
  - — The reserved bits (bits 31:4 and 2) must be 0.
  - — The field cannot indicate blocking by both STI and MOV SS (bits 0 and 1 cannot both be 1).
  - — Bit 0 (blocking by STI) must be 0 if the IF flag (bit 9) is 0 in the RFLAGS field.
  - — Bit 0 (blocking by STI) and bit 1 (blocking by MOV-SS) must both be 0 if the valid bit (bit 31) in the VM-entry interruption-information field is 1 and the interruption type (bits 10:8) in that field has value 0, indicating external interrupt.
  - — 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).
  - — 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.
  - — Note that there is *no requirement* that bit 3 (blocking by NMI) 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.
- Pending debug exceptions.
  - — Bits 11:4, bit 13, and bits 63:15 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 interrupt-

ibility-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]
  - — 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 2.2).
  - — The value of the field must differ from that of the current VMCS pointer.

#### 4.3.1.6 Checks on Guest Page-Directory Pointers

If the "IA-32e mode guest" VM-entry control is 0 and bit 5 in the CR4 field (corresponding to CR4.PAE) is 1, the logical processor will use the **physical-address extension** (PAE) if the VM entry completes successfully. See Section 3.8 ("36-Bit Physical Addressing Using the PAE Paging Mechanism") of *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.[2]

When PAE is in use, the physical address in CR3 references a table of **page-directory pointers** (PDPTRs). A MOV to CR3 when PAE is in use checks the validity of these pointers. A VM entry to a guest that uses PAE checks the validity of the PDPTRs referenced by the CR3 field, using the same checks that are used when CR3 is loaded with MOV to CR3. If MOV to CR3 would cause a general-protection exception due to the PDPTRs that would be loaded (e.g., because a reserved bit is set), the VM entry fails.

## 4.3.2 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.

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. The physical-address extension now supports 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.

This loading may be performed in parallel with the checking of VMCS contents (see Section 4.3.1).

The loading of guest state is detailed in Section 4.3.2.1 to Section 4.3.2.4. 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 4.4). This loading occurs only after the state loading described in this section and the checking of VMCS contents described in Section 4.3.1.

#### 4.3.2.1 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.
- 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 following describes how some MSRs are loaded using fields in the guest-state area:
  - IA32_DEBUGCTL MSR is loaded from the IA32_DEBUGCTL field.
  - The MSRs FS.base and GS.base are loaded from the base-address fields for FS and GS, respectively (see Section 4.3.2.2).
  - IA32_EFER.LMA and IA32_EFER.LME are each loaded with the setting of the “IA-32e mode guest” VM-entry control.
  - The IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are loaded from the IA32_SYSENTER_CS field, IA32_SYSENTER_ESP field, and the IA32_SYSENTER_EIP field, respectively. Bits 63:32 of IA32_SYSENTER_CS (whose field is only 32 bits) are cleared to 0.

  Each of these MSRs is subsequently overwritten if it appears in the VM-entry MSR-load area. See Section 4.4.

If any of CR3[63:5], CR4.PAE, CR4.PSE, or IA32_EFER.LMA is changing, the TLBs are updated so that, after VM entry, a logical processor will not use any 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).

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.

### 4.3.2.2 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 4.3.1.2). 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 fields are loaded.
    - If the unusable bit is 1, the remainder of CS access rights are undefined after VM entry.
- SS, DS, ES, FS, and 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. The only exceptions are the following:
      - SS.DPL: always loaded from the SS access-rights field. This will be the current privilege level (CPL) after the VM entry completes.
      - The base addresses for FS and GS: always loaded.
      - The base address for LDTR: set to an undefined but canonical value.
      - Bits 63:32 of the base addresses for SS, DS, and ES: cleared to 0.

    Note that the values loaded for base addresses for FS and GS are also manifest in the FS.base and GS.base MSRs.

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

#### 4.3.2.3 Loading Guest RIP, RSP, and RFLAGS

RSP, RIP, and RFLAGS are loaded from the RSP field, the RIP field, and the RFLAGS field, respectively.

#### 4.3.2.4 Loading Page-Directory Pointers

If the "IA-32e mode guest" VM-entry control is 0 and bit 5 in the CR4 field (corresponding to CR4.PAE) is 1, the logical processor will use the physical-address extension (PAE) after the VM entry. See Section 3.8 ("36-Bit Physical Addressing Using the PAE Paging Mechanism") of *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.[1]

When PAE is in use, the physical address in CR3 references a table of page-directory pointers (PDPTRs). A MOV to CR3 when PAE is in use loads the PDPTRs into the processor (into internal, non-architectural registers). A VM entry to a guest that uses PAE loads the PDPTRs into the processor as would MOV to CR3.

### 4.3.3 Clearing Address-Range Monitoring

IA-32 processors allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 7.7.3 ("MONITOR/MWAIT Instruction") of *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*. VM entries clear any address-range monitoring that may be in effect.

## 4.4 LOADING MSRS

VM entries may load MSRs from the VM-entry MSR-load area (see Section 2.8.2). 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.

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

- Bits 63:32 of the entry are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 would cause a general-protection exception if executed via WRMSR with CPL = 0.[2]

The VM entry fails if processing fails for any entry. A logical processor responds to such failures by loading state from the host-state area, as it would for a VM exit. See Section 4.7.

1. The physical-address extension now supports 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. Note that, 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-entry MSR-load area for the purpose of modifying the LME bit.

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, a logical processor will not use any translations that were cached before the transition.

## 4.5 EVENT INJECTION

If the valid bit in the VM-entry interruption-information field is 1, a logical processor delivers an event after all components of guest state have been loaded (including MSRs). 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 4.5.1 provides details of event injection. In general, the event is delivered exactly as it would had it 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 4.5.2 details cases in which event injection causes a VM exit.

### 4.5.1 Details of Event Injection

The event-injection process is controlled by the contents of the VM-entry interruption information field (format given in Table 2-7), 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

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

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.
- 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 IA-32, a software interrupt in this case 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, ordinary INT *n*/INT3/INTO-style privilege checking is performed on the IDT descriptor being accessed (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.

- 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).

### 4.5.2 VM Exits During Event Injection

An event being injected never directly causes a VM exit 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 3.4.2). Similarly, if event delivery encounters a nested exception, a VM exit may occur depending on the contents of the exception bitmap.

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 (see Section 5.1). 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 5.2.3).

## 4.6 SPECIAL FEATURES OF VM ENTRY

This section details a variety of features of VM entry. It uses the following terminology: a VM entry is **injecting** if the valid bit (bit 31) of the VM-entry interruption information field is set.

### 4.6.1 Interruptibility State

The interruptibility-state field in the guest-state area (see Table 2-3) 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 injecting, there is no blocking by STI or by MOV SS following the VM entry, regardless of the contents of the interruptibility-state field.
- If the VM entry is not injecting, the following apply:
  - Events are blocked by STI if and only if the bit 0 in the interruptibility-state field is 1. Such blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry; see Section 4.6.3).
  - Events are blocked by MOV SS if and only if the bit 1 in the interruptibility-state field is 1. This may affect the treatment of pending debug exceptions; see Section 4.6.3. Such blocking is cleared after the guest executes one instruction or incurs an exception (including a debug exception made pending by VM entry).
  - Non-maskable interrupts (NMIs) are blocked if the bit 3 in the interruptibility-state field is 1. If the “NMI exiting” VM-execution control is 0, such blocking remains in effect until IRET is executed (even if the instruction generates a fault). If the “NMI exiting” control is 1, such blocking remains in effect as long as the logical processor is in VMX non-root operation.

### 4.6.2 Activity State

The activity-state field in the guest-state area of the VMCS controls whether, after VM entry, a logical processor is active or in one of the inactive states identified in Section 2.4.2. The use of this field is determined as follows:

- If the VM entry is injecting, the logical processor is in the active state after VM entry. While the consistency checks described in Section 4.3.1.5 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 injecting, a 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.

- 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), INITs, and system-management interrupts (SMIs). External interrupts, NMIs, and INITs that 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.

## 4.6.3 Delivery of Pending Debug Exceptions after VM Entry

The pending debug exceptions field in the guest-state area of the VMCS indicates whether there are debug exceptions that have not yet been delivered (see Section 2.4.2). This section describes how these are treated on VM entry.

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

- The VM entry is injecting 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 injecting with either of the following interruption type: software interrupt or software exception.
- The VM entry is not injecting 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 injecting, the pending debug exceptions are treated as they would had they been encountered normally in guest execution:
  - If a 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 a 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 injecting (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 5.9, “Priority Among Simultaneous Exceptions and Interrupts” in the *IA-32 Intel® Architecture Software Developer’s Manual, Volume 3*). Thus, an INIT or a system-management interrupt (SMI) takes priority of such an exception. The exception takes priority over any pending non-maskable interrupt (NMI) or external interrupt.

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.

### 4.6.4 Interrupt-Window Exiting

The “interrupt-window exiting” VM-execution control may cause a VM exit to occur immediately after VM entry (see Section 3.2 for details).

A pending non-maskable interrupt (NMI) takes priority over VM exits caused by this control. VM exits caused by this control take priority over any pending external interrupts.

VM exits cause by this control wake the logical processor if the logical processor just entered the HLT state because of a VM entry (see Section 4.6.2). Such VM exits do not occur if the logical processor just entered the shutdown state or the wait-for-SIPI state.

### 4.6.5 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.

## 4.7 VM-ENTRY FAILURES DUE TO GUEST STATE

VM-entry failures due to the checks identified in Section 4.3.1 and failures during MSR loading identified in Section 4.4 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.
     - Bit 31 is set to 1 to indicate a VM-entry failure.
     - The basic exit reason part of the exit-reason field 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 4.3.1.
       34. VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs (see Section 4.4).
       41. VM-entry failure due to machine check. A machine check occurred during VM entry (see Section 4.8).
     - The remainder of the exit-reason field 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 PDPTRs (see Section 4.3.1.6).
       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 4.3.1.5, page 4-11).
       4. Failure was due to an invalid VMCS link pointer (see Section 4.3.1.5, page 4-12).
     - 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 5.5). If this results in [CR4.PAE & CR0.PG & IA32_EFER.LMA] = 1, page-directory pointers (PDPTRS) may be checked and loaded (see Section 5.5.4).
3. MSRs may be loaded from the VM-exit MSR-load area (see Section 5.6).

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.

## 4.8 MACHINE CHECKS DURING VM ENTRY

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

- The machine check is handled normally. If CR4.MCE = 1, a machine-check exception (#MC) is delivered through the IDT. If CR4.MCE = 0, the processor goes to the shutdown state.
- A VM-entry failure occurs as described in Section 4.7. The basic exit reason is 41, for "VM-entry failure due to machine check."

The first option is not used if the machine check occurs after any guest state has been loaded.

# CHAPTER 5 VM EXITS

VM exits occur in response to certain instructions and events in VMX non-root operation. Section 3.1 and Section 3.2 detail the causes of VM exits. VM exits perform the following operations:

1. Information about the cause of the VM exit is recorded in the VM-exit information fields and the valid bit (bit 31) is cleared in the VM-entry interruption-information field (Section 5.2).
2. Processor state is saved into the guest-state area (Section 5.3).
3. MSRs may be saved into the VM-exit MSR-store area (Section 5.4).
4. The following may be performed in parallel and in any order (Section 5.5):
   - Processor state is loaded based in part on the host-state area and some VM-exit controls.
   - Address-range monitoring is cleared.
5. MSRs may be loaded from the VM-exit MSR-load area (Section 5.6).

These steps are detailed in Section 5.2 through Section 5.6. Section 5.1 clarifies the nature of the architectural state before a VM exit begins.

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

## 5.1 ARCHITECTURAL STATE BEFORE A VM EXIT

This section clarifies the architectural state that exists before a VM exit, especially for those 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. INITs that arrive while the processor is not in the wait-for-SIPI activity state cause VM exits directly.
- An exception, NMI, or external interrupt causes a VM exit **indirectly** if it does not do so directly but delivery of the event causes a nested exception, double fault, or task switch that causes a VM exit.
- An event results in a VM exit if it causes a VM exit, directly or indirectly.

The following items provide details of when architectural state is and is not updated in response to VM exits resulting from certain events:

- If an event causes a VM exit directly, it does not update the architectural state as it would 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.)
  - 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 which 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 a task switch: old task-state segment (TSS); new TSS; old TSS descriptor; new TSS descriptor; RFLAGS.NT; or the TR register.
  - No last-exception record is made if the event that would do so causes a VM exit directly.
  - The fact that a machine-check exception causes a VM exit directly does not prevent the machine-check MSRs from being updated. These are considered to be updated by the machine check itself and not the resulting machine-check exception.
  - If a logical processor had been in an inactive state (see Section 2.4.2) and not executing instructions, some events are blocked but others may return the logical processor to the active state. If an unblocked event causes a VM exit directly, the return to the active state occurs only after the VM exit completes.[1] The VM exit generates any special bus cycle that is normally generated when the active state is entered from that activity state.
- If an event causes a VM exit indirectly, the exception 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.

1. 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.

— An external interrupt acknowledges the interrupt controller and the interrupt is no longer pending.

— If a logical processor had been in an inactive state, it enters the active state and, before the VM exit commences, generates any special bus cycle that is normally generated when the active state is entered from that activity state.

— There is no blocking by STI or by MOV SS when the VM exit commences.

— Processor state that is normally updated as part of delivery through the IDT (CS, RIP, SS, RSP, RFLAGS) is not modified. However, the incomplete delivery of the event may write to the stack.

— The treatment of last-exception records is implementation dependent:

  - Some processors make a last-exception record when beginning the delivery of an event through the IDT (before it can encounter a nested exception). Such processors perform this update even if the event encounters a nested exception that causes a VM exit (including the case where nested exceptions lead to a triple fault).

  - Other processors delay making a last-exception record until event delivery has reached some event handler successfully (perhaps after one or more nested exceptions). Such processors do not update the last-exception record if a VM exit or triple fault occurs before an event handler is reached.

- If a VM exit results from a fault 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 state of previous blocking by NMI may be recorded in the VM-exit interruption-information field; see Section 5.2.2.

- 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, an external interrupt, an NMI, 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 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 5.3.4).

- 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.

  For these VM exits, the instruction’s modifications to architectural state complete before the VM exit occurs. Such modifications include those to a logical processor’s interruptibility state (see Table 2-3). If there had been blocking by STI before the instruction executed, such blocking is no longer in effect (the same is true for blocking by MOV SS).

## 5.2 RECORDING VM-EXIT INFORMATION AND UPDATING CONTROLS

VM exits begin by recording information about the nature of and reason for the VM exit in the VM-exit information fields. Section 5.2.1 to Section 5.2.4 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.

### 5.2.1 Basic VM-Exit Information

Section 2.9.1 defines the basic VM-exit information fields. The following items detail their use.

- The basic exit reason part of the exit-reason field is loaded with a number indicating the general cause of the VM exit. Appendix A enumerates the numbers used and their meaning. The remainder of the exit-reason field is cleared.
- Exit qualification. This field is saved for VM exits due to the following causes: page-fault exceptions; start-up IPIs (SIPIs); task switches; INVLPG; VMCLEAR; VMPTRLD; VMPTRST; VMREAD; VMWRITE; VMXON; control-register accesses; MOV DR; I/O instructions; and MWAIT. For all other VM exits, this field is cleared. The following items provide details:
  - — For debug exceptions, the exit qualification has the format given in Table 5-1.

**Table 5-1. 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) |

— For page-fault exceptions, the exit qualification contains the linear address that caused the page fault. Bits 63:32 are cleared if a logical processor was not in 64-bit mode before the VM exit.

— Start-up IPI (SIPI). The SIPI vector information is stored in bits 7:0 of the exit qualification. Bits 63:8 are cleared to 0.

— Task switch. Details about the reason for the VM exit are encoded as given in Table 5-2.

**Table 5-2. 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:<br>0: CALL instruction<br>1: IRET instruction<br>2: JMP instruction<br>3: Task gate in IDT |
| 63:32 | Reserved (cleared to 0) |

— For INVLPG, the exit qualification contains the linear-address operand of the instruction. Bits 63:32 are cleared if a 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. Note that this address is not architecturally defined and may be implementation-specific.

— VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXON. The 64-bit instruction displacement. The exit qualification receives the value of the instruction's displacement field, which is sign-extended to 64 bits if necessary. If the instruction has no displacement (for example, has a register operand), zero is stored into the exit qualification.

  An exception is made for RIP-relative addressing (which can be used only in 64-bit mode), which 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, the bits of this field beyond the instruction's address size are undefined. Specifically, suppose that the address-size field in the instruction-information field (see below) reports an N-bit address size. Then bits 63:N of the reported instruction displacement are undefined.

— For control-register accesses, the exit qualification has the format given in Table 5-3.

**Table 5-3. Exit Qualification for Control-Register Accesses**

| Bit Positions | Contents |
|---|---|
| 3:0 | Number of control register (0 for CLTS and LMSW) |
| 5:4 | Access type:<br>0 = MOV to CR<br>1 = MOV from CR<br>2 = CLTS<br>3 = LMSW |
| 6 | LMSW operand type:<br>0 = register<br>1 = memory<br>For CLTS and MOV CR, cleared to 0 |
| 7 | Reserved (cleared to 0) |
| 11:8 | For MOV CR, the general-purpose register:<br><br>0 = RAX<br>1 = RCX<br>2 = RDX<br>3 = RBX<br>4 = RSP<br>5 = RBP<br>6 = RSI<br>7 = RDI<br>8–15 represent R8–R15, respectively<br><br>For CLTS and LMSW, cleared to 0 |
| 15:12 | Reserved (cleared to 0) |

**Table 5-3. Exit Qualification for Control-Register Accesses (Contd.)**

| Bit Positions | Contents |
|---|---|
| 31:16 | For LMSW, the LMSW source data<br>For CLTS and MOV CR, cleared to 0 |
| 63:32 | Reserved (cleared to 0) |

— For MOV DR, the exit qualification has the format given in Table 5-4.

**Table 5-4. 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:<br>0 = RAX<br>1 = RCX<br>2 = RDX<br>3 = RBX<br>4 = RSP<br>5 = RBP<br>6 = RSI<br>7 = RDI<br>8–15 represent R8–R15, respectively |
| 63:12 | Reserved (cleared to 0) |

— For I/O instructions, the exit qualification has the format given in Table 5-5.

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

| Bit Position(s) | Contents |
|---|---|
| 2:0 | Size of access:<br>0 = 1-byte<br>1 = 2-byte<br>3 = 4-byte<br>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) |

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

| Bit Position(s) | Contents |
|---|---|
| 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) |

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

### 5.2.2 Information for VM Exits Due to Vectored Events

Section 2.9.2 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). (These VM exits include those that occur on an attempt at a task switch that causes an exception (before generating a VM exit due to the task switch) that causes a VM exit.)

The following items detail the use of these fields.

- VM-exit interruption information (format given in Table 2-9). 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. Note that BOUND range exceeded exceptions (#BR; generated by BOUND) and those 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. If bit 11 is set to 1, that error code is placed in the VM-exit exception error-code field (see below).
  - — Bit 12 is undefined in any of the following cases:
    - If the VM exit occurs with the "NMI exiting" VM-execution control set to 1.
    - If the VM exit sets the valid bit in the IDT-vectoring information field (see Section 5.2.3).

    - 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 VM exit is due to a fault on the IRET instruction and blocking by NMI (see Table 2-3) 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.

  — 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 for IA-32 exceptions. 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.

### 5.2.3 Information for VM Exits During Event Delivery

Section 2.9.3 defined fields containing information for VM exits that occur while delivering an event through the IDT[1] and as a result of either of the following two cases:

- 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).[2]
- A task switch is invoked via a task gate in the IDT. Note that the VM exit occurs due to the task switch only after the initial checks of the task switch pass (see Section 3.4.2).

1. This includes cases in which the event delivery was caused by event injection as part of VM entry; see Section 4.5.2.
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 2-10).
  - — This field is marked valid (by setting bit 31) and the relevant data are saved for VM exits that occur in the course of delivering an event through the IDT and as a result of either of the following two cases: (1) a fault occurs during event delivery and causes a VM exit; or (2) a task switch is invoked via a task gate in the IDT.
  - — 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 exactly when it would be set for IA-32 exceptions.
  - — For other VM exits, the value of this field is undefined.

### 5.2.4 Information for VM Exits Due to Instruction Execution

Section 2.9.4 defined fields containing information for VM exits that occur due to instruction execution. The following items detail their use.

- VM-exit instruction length. This field is used in the following cases:
  - — For VM exits due to attempts to execute one of the following instructions that cause VM exits unconditionally (see Section 3.1.2) or based on the settings of VM-execution controls (see Section 3.1.3): CLTS, CPUID, HLT, IN, INS INVD, INVLPG, LMSW, MONITOR, MOV CR, MOV DR, MWAIT, OUT, OUTS, PAUSE, RDMSR, RDPMC, RDTSC, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMREAD, VMRESUME, VMWRITE, VMXOFF, VMXON, and WRMSR.
  - — For VM exits due to software exceptions (those generated by executions of INT3 or INTO).

- For VM exits due to exceptions that occur during delivery of a software interrupt (generated by INT *n*) or a software exception (generated by an execution of INT3 or INTO).
- 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 a IDT-vectoring information field indicating a software interrupt or software exception.

In all these cases, this fields receives the length in bytes (1–15) of the instruction (including any instruction prefixes) whose execution led to the VM exit. All other VM exits leave this field undefined.

- Guest linear address. For VM exits due to some instructions, this field receives the linear address of one of the instruction operands. Bits 63:32 are cleared if a logical processor was not in 64-bit mode before the VM exit.
  - For VM exits due to attempts to execute LMSW with a memory operand, this field receives the linear address of that operand.
  - For VM exits due to attempts to execute INS or OUTS, this 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 for OUTS by a segment override prefix) at the time the instruction started.
  - For all other VM exits, the field is undefined.
- VMX-instruction information (format given in Table 2-11).
  - For VM exits due to attempts to execute VMCLEAR, VMPTRLD, VMPTRST, VMREAD, VMWRITE, or VMXON, this field receives information about the instruction that caused the VM exit.
  - For all other VM exits, the field is undefined.

## 5.3 SAVING GUEST STATE

Each field in the guest-state area of the VMCS (see Section 2.4) is written with the corresponding component of processor state. The full values of each 64-bit field (for example, the base address for GDTR) is saved regardless of the mode of a logical processor before and after the VM exit.

In general, the state saved is that which was in a logical processor at the time the VM exit commences. See Section 5.1 for a discussion of which architectural updates occur at that time.

Section 5.3.1 through Section 5.3.4 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.

### 5.3.1 Saving Control Registers, Debug Registers, and MSRs

The contents of CR0, CR3, CR4, DR7, and the IA32_DEBUGCTL, IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP MSRs are saved into the corresponding fields.

### 5.3.2 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 2.4.1) 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.
    - Bits 63:32 of the values saved for the base addresses are always zero.
    - DS and ES. 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.

## 5.3.3 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 or a start-up IPI (SIPI), the value saved is that which was in RIP before the event occurred.
  - If the VM exit occurs due to the 1-setting of the "interrupt-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 5.2.2), 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 exits is due to a triple fault, the value saved is exactly that which would have been provided to the guest event handler (see above) 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.
  - If the VM exit is due to a MOV to CR8 that reduced the value of the TPR shadow below that of the TPR threshold, the value saved references the instruction following the MOV to CR8.
- The contents of the RSP register are saved into the RSP field.

1. The reference here is to the full 64-bit value of RIP before any truncation that would occur had the stack width been only 32 bits or 16 bits.

- With the exception of the RF (bit 16), the contents of the RFLAGS register is saved into the RFLAGS field. The 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[1] 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.[2]
  - For all other VM exits, the value saved in is the value RFLAGS.RF had before the VM exit occurred.

## 5.3.4 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. See Section 5.1 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 5.1 for details of how events leading to a VM exit may affect this state.
- The pending debug exceptions field is saved as clear for all VM exits except the following:
  - VM exits caused by an INIT, a machine-check exception, an execution of MOV to CR8 that reduces the value of the TPR shadow below that of the TPR threshold.

1. The reference here is to the full 64-bit value of RFLAGS before any truncation that would occur had the stack width been only 32 bits or 16 bits.
2. 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.

— VM exits that not caused by debug exceptions and that occur while there is MOV-SS blocking of debug exceptions.

For VM exits that do not clear the field, the value saved is determined as follows:

— Each of bits 3:0 may be set if it corresponds to a matched breakpoint. This may be true even if the corresponding breakpoint is not enabled in DR7.

— Suppose that a VM exit is due to an INIT, a machine-check exception, or MOV to CR8 that reduces the value of the TPR shadow below that of the TPR threshold. 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 an INIT or machine check occurs immediately after VM entry, the value saved may match that which was loaded on VM entry (see Section 4.6.3). 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 4.6.3) 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 4.6.3). 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 4.6.3) 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.

## 5.4 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 2.7.2). 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:

- 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 5.7.

## 5.5 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.

The full values of each 64-bit field loaded (for example, the base address for GDTR) is loaded regardless of the mode of a logical processor before and after the VM exit.

The loading of host state is detailed in Section 5.5.1 to Section 5.5.5. These sections reference VMCS fields that correspond to processor state. Unless otherwise stated, these references are to fields in the host-state area.

In addition to loading host state, VM exits clear address-range monitoring (Section 5.5.6).

After the state loading described in this section, VM exits may load MSRs from the VM-exit MSR-load area (see Section 5.6). This loading occurs only after the state loading described in this section.

### 5.5.1 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. However, the following bits are not modified:
  - For CR0, PE, ET, NE, CD, NW, PG, and bits 63:32, 28:19, 17, and 15:6.[1]

1. Note that 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.

— For CR3, bits 63:52 and bits in the range 51:32 beyond the processor's physical-address width (they are cleared to 0).[1]

— CR4.VMXE.

- DR7 is set to 00000000_00000400H.
- The following MSRs are established as follows:

— The IA32_DEBUGCTL MSR is cleared to 00000000_00000000H.

— IA32_SYSENTER_CS, IA32_SYSENTER_ESP, and IA32_SYSENTER_EIP are loaded from the IA32_SYSENTER_CS field, the IA32_SYSENTER_ESP field, and the IA32_SYSENTER_EIP field, respectively. Since the IA32_SYSENTER_CS field contains only 32 bits, bits 63:32 of that register are cleared to 0.

— IA32_EFER.LMA and IA32_EFER.LME are each loaded with the setting of the "host address-space size" VM-exit control.

Each of these MSRs is subsequently overwritten if it appears in the VM-exit MSR-load area. See Section 5.6.

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, a 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).

## 5.5.2 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. Note that the checks specified Section 4.3.1.2 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 if the VM exit is to 64-bit mode, when it is possible to use 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.

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.

- FS and GS. Undefined (but canonical) if the segment is unusable and the VM exit is not to 64-bit mode; otherwise, loaded from the base-address field. The MSRs FS.base and GS.base are always identical to the base addresses of FS and GS, respectively.
- TR. Loaded from the host-state area.

- 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.
  - 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.
- CS.L is loaded with the setting of the “host address-space size” VM-exit control. Because this control is also loaded into IA32_EFER.LMA (see Section 5.5.1), 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, 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 set to zero, 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. The GDTR and IDTR limits are each set to FFFFH.

### 5.5.3 Loading Host RIP, RSP, and RFLAGS

RIP and RSP are loaded from the RIP field and the RSP field, respectively.

RFLAGS is set to 00000000_00000002H.

### 5.5.4 Checking and Loading Host Page-Directory Pointers

If the "host address-space size" VM-exit control is 0 and bit 5 in the CR4 field (corresponding to CR4.PAE) is 1, the logical processor will use the **physical-address extension** (PAE) after the VM exit. See Section 3.8 ("36-Bit Physical Addressing Using the PAE Paging Mechanism") of *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*.[1] When PAE is in use, the physical address in CR3 references a table of **page-directory pointers** (PDPTRs). A MOV to CR3 when PAE is in use checks the validity of the PDPTRs and loads them into the processor (into internal, non-architectural registers).

A VM exit to a VMM that uses PAE checks the validity of the PDPTRs referenced by the CR3 field, using the same checks that are used when CR3 is loaded with MOV to CR3. If MOV to CR3 would cause a general-protection exception due to the PDPTRs that would be loaded (e.g., because a reserved bit is set), a VMX abort occurs. If a VM exit to a VMM that uses PAE does not cause a VMX abort, the PDPTRs are loaded into the processor as would MOV to CR3.

### 5.5.5 Updating Non-Register State

A logical processor is always in the active state after a VM exit.

There is no blocking by STI or by MOV SS after a VM exit. VM exits caused directly by a non-maskable interrupt (NMI) block subsequent NMIs; other VM exits do not affect blocking by NMI (but see Section 5.1 for the case in which an NMI causes a VM exit indirectly).

There are no pending debug exceptions after a VM exit.

1. The physical-address extension now supports 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.

### 5.5.6 Clearing Address-Range Monitoring

IA-32 processors allow software to monitor a specified address range using the MONITOR and MWAIT instructions. See Section 7.7.3 ("MONITOR/MWAIT Instruction") of *IA-32 Intel® Architecture Software Developer's Manual, Volume 3*. VM exits clear any address-range monitoring that may be in effect.

## 5.6 LOADING MSRS

VM exits may load MSRs from the VM-exit MSR-load area (see Section 2.7.2). 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 either of the following cases:

- Bits 63:32 of the entry are not all 0.
- An attempt to write bits 127:64 to the MSR indexed by bits 31:0 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 5.7.

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, a logical processor does not use any translations that were cached before the transition.

## 5.7 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 2.2). The following values are used:

1. There was a failure in saving guest MSRs (see Section 5.4).
2. Host checking of the page-directory pointers (PDPTRs) failed (see Section 5.5.4).
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.

1. Note that, 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.

4. There was a failure on loading host MSRs (see Section 5.6).
5. There was a machine check during VM exit (see Section 5.8).

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, a logical processor experiencing a VMX abort 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 checks; INITs; external interrupts; non-maskable interrupts (NMIs); start-up IPIs (SIPIs); and system-management interrupts (SMIs).

## 5.8 MACHINE CHECK DURING VM EXIT

If a machine check occurs during VM exit, one of the following occurs:

- The machine check is handled normally. If CR4.MCE = 1, a machine-check exception (#MC) delivered through the guest IDT. If CR4.MCE = 0, the processor goes to the shutdown state.
- A VMX abort is generated (see Section 5.7). The logical processor blocks events as done normally in VMX abort. The VMX abort indicator is 5, for “machine check during VM exit.”

The first option is not used if the machine check occurs after any host state has been loaded.

# CHAPTER 6 VMX CAPABILITY REPORTING

As noted in Section 1.6, 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 the features described in this document.

Support for specific features detailed in this document is reported through values that can be read from a set of capability MSRs. These MSRs are indexed starting at MSR address 1152. The MSRs are read-only; an attempt to write them (with WRMSR) produces a general-protection exception (#GP(0)). These MSRs 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)).

## 6.1 BASIC INFORMATION

The VMX_BASIC MSR (index 1152) consists of the following fields:

- Bits 31:0 contain the 32-bit VMCS revision identifier used by the processor.
- 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 all clear).
- Bits 53:50 report the memory type that the processor uses to access the VMCS for VMREAD and VMWRITE and to access the VMCS and data structures referenced by pointers in the VMCS (for example, I/O bitmaps, TPR shadow, etc.) during VM entries, VM exits, and in VMX non-root operation. The first processors to support VMX operation use the write-back type. The values used are given in Table 6-1.

**Table 6-1. Memory Types Used For VMCS Access**

| Value(s) | Field |
|---|---|
| 0 | Strong Uncacheable (UC) |
| 1–5 | Not used |
| 6 | Write Back (WB) |
| 7–15 | Not used |

Software should map all VMCS regions and referenced data structures with the indicated memory type.

- The values of bits 49:45 and bits 63:54 are reserved and are read as 0.

## 6.2 VM-EXECUTION CONTROLS

The VMX_PINBASED_CTLS MSR (index 1153) reports on the allowed settings of the pin-based VM-execution controls (see Section 2.6.1):

- Bits 31:0 indicate the **allowed 0-settings** of these controls. VM entry fails if bit X in the pin-based VM-execution controls is 0 and bit X is 1 in this MSR.
- Bits 63:32 indicate the **allowed 1-settings** of these controls. VM entry fails if bit X in the pin-based VM-execution controls is 1 and bit 32+X is 0 in this MSR.

The VMX_PROCBASED_CTLS MSR (index 1154) reports on the allowed settings of the processor-based VM-execution controls (see Section 2.6.2):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X in the processor-based VM-execution controls is 0 and bit X is 1 in this MSR.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X in the processor-based VM-execution controls is 1 and bit 32+X is 0 in this MSR.

## 6.3 VM-EXIT CONTROLS

The VMX_EXIT_CTLS MSR (index 1155) reports on the allowed settings of the VM-exit controls (see Section 2.7.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X in the VM-exit controls is 0 and bit X is 1 in this MSR.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X in the VM-exit controls is 1 and bit 32+X is 0 in this MSR.

## 6.4 VM-ENTRY CONTROLS

The VMX_ENTRY_CTLS MSR (index 1156) reports on the allowed settings of the VM-entry controls (see Section 2.8.1):

- Bits 31:0 indicate the allowed 0-settings of these controls. VM entry fails if bit X in the VM-entry controls is 0 and bit X is 1 in this MSR.
- Bits 63:32 indicate the allowed 1-settings of these controls. VM entry fails if bit X in the VM-entry controls is 1 and bit 32+X is 0 in this MSR. VM entry also fails if either bit 10 or bit 11 in the VM-entry controls is 1 regardless of the values of bit 42 and bit 43 in this MSR.

## 6.5 MISCELLANEOUS DATA

The VMX_MISC MSR (index 1157) consists of the following fields:

- 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. Note that 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 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).
- Bits 5:0, bits 15:9, and bits 63:28 are reserved and are read as 0.

## 6.6 VMX-FIXED BITS IN CR0

The first processors to support VMX operation require that CR0.PE, CR0.NE, and CR0.PG all be 1 when the processor is in VMX operation (see Section 1.8). Future processors may differ with regard to bits in CR0 that are fixed while in VMX operation. The VMX_CR0_FIXED0 MSR (index 1158) and VMX_CR0_FIXED1 MSR (index 1159) indicate how bits in CR0 may be set in VMX operation.

The VMX_CR0_FIXED0 MSR and the VMX_CR0_FIXED1 MSR report on bits in CR0 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X of VMX_CR0_FIXED0 is 1, then that bit of CR0 is fixed to 1 in VMX operation. Similarly, if bit X of VMX_CR0_FIXED1 is 0, then that bit of CR0 is fixed to 0 in VMX operation.

In addition to indicating that CR0.PE, CR0.NE, and CR0.PG must be 1 in VMX operation, these MSRs indicates that bits 63:32 must be 0.

## 6.7 VMX-FIXED BITS IN CR4

The first processors to support VMX operation require that CR4.VMXE be set whenever the processor is in VMX operation (see Section 1.8). Future processors may differ with regard to bits in CR4 that are fixed while in VMX operation. The VMX_CR4_FIXED0 MSR (index

1160) and VMX_CR4_FIXED1 MSR (index 1161) indicate how bits in CR4 may be set in VMX operation.

The VMX_CR4_FIXED0 MSR and the VMX_CR4_FIXED1 MSR report on bits in CR4 that are allowed to be 0 and 1, respectively, in VMX operation. If bit X of VMX_CR4_FIXED0 is 1, then that bit of CR4 is fixed to 1 in VMX operation. Similarly, if bit X of VMX_CR4_FIXED1 is 0, then that bit of CR4 is fixed to 0 in VMX operation.

In addition to indicating that CR0.PE, CR0.NE, and CR0.PG must be 1 in VMX operation, these MSRs indicates that all reserved bits in CR4 must be 0.

## 6.8 VMCS ENUMERATION

The VMX_VMCS_ENUM MSR (index 1162) provides information to assist software in enumerating fields in the VMCS.

As noted in Section 2.10.2, 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.

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.
- The values of bit 0 and bits 63:10 are reserved and are read as 0.

# CHAPTER 7
# VMX INSTRUCTION SET REFERENCE

The virtual-machine extensions (VMX) includes five instructions that manage the virtual-machine control structure (VMCS) and five instruction that manage VMX operation.

Section 7.1 provides an overview of the instructions. Section 7.2 describes conventions used in documenting the instructions. Section 7.3 contains the a detailed description of each instruction.

## 7.1 OVERVIEW

The behavior of the VMCS-maintenance instructions are 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 behaviors of the rest of the VMX instructions are summarized below:

- VMCALL. This instruction allows a guest in VMX non-root operation to call the VMM for service. A VM exit occurs, transferring control to the VMM.
- 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 leaves 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.

Only VMCALL can be executed in compatibility mode (it causes a VM exit). The other instructions generate invalid-opcode exceptions if executed in compatibility mode.

## 7.2 CONVENTIONS

The operation sections for the VMX instructions in Section 7.3 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 Appendix B.

## 7.3 VMX INSTRUCTION REFERENCE

This section provides detailed descriptions of the VMX instructions.

This section does not completely detail the debug exceptions (#DB) that may be generated by the VMX instructions. Debug exceptions are controlled by basic IA-32 functionality and, in general, this is not changed in VMX operation. Debug exceptions are mentioned below only when their treatment differs from normal IA-32 functionality. Failure to mention debug exceptions in conjunction with a particular instruction should not be construed as implying that that instruction cannot cause a debug exception. Accesses by a logical processor to a VMCS region or to data structures referenced by addresses in the VMCS cannot trigger data breakpoints.

# VMCALL—Call to VM Monitor

| Opcode | Instruction | Description |
|---|---|---|
| 0F 01 C1 | VMCALL | Call to VM monitor by causing VM exit |

## Description

This instruction is designed so that guest software can make a call for service into an underlying VM monitor. The details of the programming interface for such calls are monitor-specific; this instruction does nothing more than cause a VM exit, registering the appropriate exit reason.

## 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);
    ELSE VMfail(VMCALL executed in VMX root operation);
FI;
```

If retired instructions are being counted, an execution of VMCALL is counted if the instruction does not fault. A non-faulting execution is counted regardless of whether it causes a VM exit or fails due to being executed in VMX root operation.

## Flags Affected

See the operation section and Section 7.2.

## Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REP* | Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Ignored |
| Operand size | Causes #UD |
| Address size | Ignored |
| REX | Ignored |

## 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. |
| #DB | This instruction does not cause a single-step trap due to RFLAGS.TF=1 if it causes a VM exit. |

### Real-Address Mode Exceptions

| | |
|---|---|
| #UD | A logical processor cannot be in real-address mode while in VMX operation and the VMCALL instruction is not recognized outside VMX operation. |

### Virtual-8086 Mode Exceptions

| | |
|---|---|
| #UD | If executed outside VMX non-root operation. |

### Compatibility Mode Exceptions

| | |
|---|---|
| #UD | If executed outside VMX non-root operation. |

### 64-Bit Mode Exceptions

| | |
|---|---|
| #UD | If executed outside VMX operation. |
| #DB | This instruction does not cause a single-step trap due to RFLAGS.TF=1 if it causes a VM exit. |

## 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 Section 2.11.

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 (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
            THEN VMfail(VMCLEAR with invalid physical address);
            ELSIF addr = VMXON pointer
                THEN VMfail(VMCLEAR with VMXON pointer);
                ELSE
                    ensure that data for VMCS referenced by the operand is in memory;
                    initialize implementation-specific data in VMCS region;
                    launch state of VMCS referenced by the operand ← “clear”
                    IF operand addr = current-VMCS pointer
                        THEN current-VMCS pointer ← FFFFFFFF_FFFFFFFFH;
                    FI;
                    VMsucceed;
        FI;
FI;
```

### Flags Affected

See the operation section and Section 7.2.

### Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD. |
| REP* | Reserved and may cause unpredictable behavior (applies to both REPNE/REPNZ and REP/REPE/REPZ). |
| Segment overrides | Treated normally |
| Operand size | Ignored |
| Address size | Treated normally |
| REX | Register extensions treated normally; operand-size overrides ignored |

### 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. |

## 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, a logical processor performs a series of consistency checks as detailed in Section 4.2 (VMX controls and host-state area) and Section 4.3.1 (guest-state area). 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. Neither VMLAUNCH nor VMRESUME should be used immediately after either MOV to SS or POP to SS.

### 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 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)) 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 4.7);
                    ELSE
                        Attempt to load MSRs from VM-entry MSR-load area;
                        IF failure
                            THEN VM entry fails (see Section 4.7);
                            ELSE
                                IF VMLAUNCH
                                    THEN launch state of VMCS ← "launched";
                                FI;
                                VM entry succeeds;
                        FI;
                FI;
        FI;
FI;
```

Further details of the operation of the VM-entry instructions appear in Chapter 4.

If retired instructions are being counted, an execution of one of the VM-entry instructions are counted if it does not fault. A non-faulting execution are counted regardless of whether it leads to a successful VM entry.

## Flags Affected

See the operation section.

## Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REP* | Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Ignored |
| Operand size | Causes #UD |
| Address size | Ignored |
| REX | Ignored |

## Protected Mode Exceptions

| | |
|---|---|
| #GP(0) | If the current privilege level is not 0. |
| #UD | If executed outside VMX operation. |
| #DB | These instructions does not cause a single-step trap due to RFLAGS.TF=1 if they cause a successful VM entry. |

### 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. |
| #DB | These instructions does not cause a single-step trap due to RFLAGS.TF=1 if they cause a successful VM entry. |

## 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.[1]

The operand of this instruction is always 64 bits and is always in memory.

### Operation

```
IF (register operand) or (not in VMX operation) or (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 processor’s physical-address width
           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;
               FI;
       FI;
FI;
```

1. Software should consult the VMX capability MSR VMX_BASIC to discover the VMCS revision identifier supported by this processor (see Section 6.1).

### Flags Affected

See the operation section and Section 7.2.

### Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REPNE/REPNZ | Causes #UD |
| REP/REPE/REPZ | Changes encoding to that of VMXON; see “VMXON—Enter VMX Operation” for operation and interactions with other prefixes. |
| Segment overrides | Treated normally |
| Operand size | Changes encoding to that of VMCLEAR; see “VMCLEAR—Clear Virtual-Machine Control Structure” for operation and interactions with other prefixes. |
| Address size | Treated normally |
| REX | Register extensions treated normally; operand-size overrides ignored |

### 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 (RFLAGS.VM = 1) or
(IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit(reason=”VMPTRST”);
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 7.2.

### Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REP* | Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Treated normally |
| Operand size | Causes #UD |
| Address size | Treated normally |
| REX | Register extensions treated normally; operand-size overrides ignored |

### 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. This source operand is always 32 bits, regardless of the value of CS.D. The destination operand, which may be a register or in memory, is always treated as a 32-bit quantity outside IA-32e mode (the setting of CS.D is ignored) and as a 64-bit quantity in 64-bit mode. The value of CS.D and any address-size overrides continue to determine the resolved memory address size normally. If the specified VMCS field is shorter than this, the high bits of the destination are cleared to 0. If the field is longer than the destination, 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(reason=”VMREAD”);
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 7.2.

### Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |

| | |
|---|---|
| REP* | Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Treated normally |
| Operand size | Causes #UD |
| Address size | Treated normally |
| REX | Register extensions treated normally; operand-size overrides ignored |

### 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 entry for 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 using the contents of a specified primary source operand (register or memory). The VMCS field is identified by the VMCS-field encoding contained in the register secondary source operand. This secondary source operand is always 32 bits, regardless of the value of CS.D. The primary source operand, which may be a register or in memory, is always treated as a 32-bit quantity outside IA-32e mode (the setting of CS.D is ignored) and as a 64-bit quantity in 64-bit mode. The value of CS.D and any address-size overrides continue to determine the resolved memory address size normally. If the specified VMCS field is shorter than this, the high bits of the primary source operand are ignored. If the field is longer than the primary source operand, 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 (RFLAGS.VM = 1) or (IA32_EFER.LMA = 1 and CS.L = 0)
    THEN #UD;
ELSIF in VMX non-root operation
    THEN VMexit(reason=”VMWRITE”);
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 7.2.

## Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REP* | Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Treated normally |
| Operand size | Causes #UD |
| Address size | Treated normally |
| REX | Register extensions treated normally; operand-size overrides ignored |

## 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 a logical processor out of VMX operation, unblocks INIT, re-enables A20M, and clears any address-range monitoring. See Section 7.7.3 (“MONITOR/MWAIT Instruction”) of *IA-32 Intel® Architecture Software Developer’s Manual, Volume 3*.

### 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);
    ELSE
        leave VMX operation;
        unblock INIT;
        unblock and enable A20M;
        clear address-range monitoring;
        VMsucceed;
FI;
```

### Flags Affected

See the operation section and Section 7.2.

### Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REP* | Cause #UD (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Ignored |
| Operand size | Causes #UD |
| Address size | Ignored |
| REX | Ignored |

### 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 a logical processor in VMX operation with no current VMCS, blocks INIT, disables A20M, and clears any address-range monitoring established by the MONITOR instruction. See Section 7.7.3 (“MONITOR/MWAIT Instruction”) of *IA-32 Intel® Architecture Software Developer’s Manual*, *Volume 3*.

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.

The operand of this instruction is always 64 bits and is always in memory.

### Operation

```
IF (register operand) or (CR4.VMXE = 0) or (CR0.PE = 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
       (CR0.NE = 0) or (CR0.PG = 0) (* see Section 1.8 *) or
       (bit 0 (lock bit) of IA32_FEATURE_CONTROL MSR is clear) or
       (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 VMX physical-address width
                   THEN VMfailInvalid;
                   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;
                               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 7.2.

## Use of Prefixes

| | |
|---|---|
| LOCK | Causes #UD |
| REP* | Ignored (includes REPNE/REPNZ and REP/REPE/REPZ) |
| Segment overrides | Treated normally |
| Operand size | Ignored |
| Address size | Treated normally |
| REX | Register extensions treated normally; operand-size overrides ignored |

## 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. |

# CHAPTER 8
# INTERACTIONS WITH SYSTEM-MANAGEMENT MODE

The interactions of the virtual-machine extensions (VMX) with system-management interrupts (SMIs) and system-management mode (SMM) are few. This section details those interactions.

## 8.1 TREATMENT OF SMI DELIVERY

Ordinary SMI delivery saves processor state into SMRAM and then loads state based on certain architectural definitions. Processors that support VMX operation perform SMI delivery as follows:

```
Assert SMMEM on subsequent bus transactions
IF processor generally issues SMIACK on entry to SMM
     THEN issue SMIACK special bus transaction before subsequent bus transactions
FI
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
        leave VMX operation;
        save VMX-critical state defined below;
        preserve current VMCS pointer as noted below;
FI;
CR4.VMXE ← 0;
perform ordinary SMI delivery:
    save processor state in SMRAM;
    set processor state to standard SMM values;[1]
```

The pseudocode above makes reference to the saving of **VMX-critical state**. This state comprises the following: (1) SS.DPL (the current privilege level); (2) RFLAGS.VM; and (3) the state of blocking by STI and by MOV SS (see Table 2-3). These data may be saved internal to the processor or in the VMCS region of the current VMCS. Note that a processor that does not support SMI recognition while blocking by STI or by MOV SS need not save the state of such blocking.

SMI delivery in VMX operation ensures that the current VMCS pointer is preserved. It is not saved in SMRAM.

1. This causes the logical processor to block assertions of INIT, NMI, and SMI.

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.

If a logical processor is in the wait-for-SIPI state, SMIs are blocked. The behavior described in this section is not invoked in this case.

## 8.2 TREATMENT OF RSM

Ordinary execution of RSM restores processor state from SMRAM. 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;
        CR4.VMXE ← value stored internally;
        IF internal storage indicates that the logical processor
        had been in VMX operation (root or non-root)
            THEN
                enter VMX operation (root or non-root);
                restore VMX-critical state as defined in Appendix 8.1;
                set CR0.PE, CR0.NE, and CR0.PG to 1;
                IF RFLAGS.VM = 0
                    THEN
                        CS.RPL ← SS.DPL;
                        SS.RPL ← SS.DPL;
                FI;
                If necessary, restore current VMCS pointer;
        FI;
        Leave SMM
        Deassert SMMEM on subsequent bus transactions
        IF processor generally issues SMIACK on leaving SMM
            THEN issue SMIACK special bus transaction
                    before subsequent bus transactions
        FI
        IF logical processor will be in VMX operation after RSM
            THEN block A20M and leave A20M mode;
        FI;
FI;
```

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 (see Section 3.2).

RSM unblocks SMIs and restores the state of blocking by NMI (see Table 2-3), as it does normally. Assertions of INIT are blocked after RSM if and only if a logical processor will be in VMX root operation.

## 8.3 PROTECTION OF CR4.VMXE IN SMM

While a logical processor is in SMM, CR4.VMXE is treated as a reserved bit. Any attempt by software running in SMM to set this bit causes a general-protection exception. Software cannot use VMX instructions or enter VMX operation while in SMM.

# APPENDIX A
# BASIC EXIT REASONS

As noted in Section 2.9.1, every VM exit writes a 32-bit exit reason to the VMCS. As noted in Section 4.7, certain VM-entry failures also do this. 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 A-1 provides values that may be written for the basic exit reason and explains their meaning. Each entry corresponds to a VM exit unless otherwise noted.

**Table A-1. Basic Exit Reasons**

| Error Number | 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; or (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. An INIT arrived |
| 4 | Start-up IPI (SIPI). A SIPI arrived while the logical processor was in the "wait-for-SIPI" state. |
| 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. |
| 9 | Task switch. Guest software attempted a task switch. |
| 10 | CPUID. Guest software attempted to execute CPUID. |
| 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. |
| 18 | VMCALL. Guest software executed VMCALL. |

Table A-1.  Basic Exit Reasons (Contd.)

| Error Number | Description |
|---|---|
| 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 3.1 for details). |
| 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 "activate I/O bitmaps" VM-execution control was 0 and the "unconditional I/O exiting" VM-execution control was 1 or (2) the "activate I/O bitmaps" VM-execution control was 1 and a bit in the I/O bitmap associated with one of the ports accessed by the I/O instruction was 1. |
| 31 | RDMSR. Guest software attempted to execute RDMSR. |
| 32 | WRMSR. Guest software attempted to execute WRMSR. |
| 33 | VM-entry failure due to invalid guest state. A VM entry failed one of the checks identified in Section 4.3.1. |
| 34 | VM-entry failure due to MSR loading. A VM entry failed in an attempt to load MSRs. See Section 4.4. |
| 36 | MWAIT. Guest software attempted to execute MWAIT and the "MWAIT exiting" VM-execution control was 1. |
| 39 | MONITOR. Guest software attempted to execute MONITOR and the "MONITOR exiting" VM-execution control was 1. |
| 40 | PAUSE. Guest software attempted to execute PAUSE and the "PAUSE exiting" VM-execution control was 1. |
| 41 | VM-entry failure due to machine check. A machine check occurred during VM entry (see Section 4.8). |
| 43 | TPR below threshold. Guest software executed MOV to CR8, the "use TPR shadow" VM-execution control was 1, and the instruction reduces the value of the TPR shadow below that of the TPR threshold. |

# APPENDIX B
# VM-INSTRUCTION ERROR NUMBERS

The operation sections for the VMX instructions in Section 7.3 indicate that, for certain error conditions, the VM-instruction error field is loaded with an error number to indicate the source of the error. Table B-1 lists the error numbers that are used:

**Table B-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 with a corrupted VMCS. Indicates corruption of the current VMCS (see Section 2.10.1). |
| 7 | VM entry with invalid VMX-control field(s). |
| 8 | VM entry with invalid host-state field(s). |
| 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. |
| 26 | VM entry with events blocked by MOV SS. |

# APPENDIX C
# ENCODINGS OF FIELDS IN THE VMCS

Every component of the VMCS is encoded by a 32-bit field that can be used by VMREAD and VMWRITE. Section 2.10.2 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.)

## C.1 16-BIT FIELDS

A value of 0 in bits 14:13 of an encoding indicates a 16-bit field. Only the guest-state area and the host-state area contain 16-bit fields. As noted in Section 2.10.2, 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.

### C.1.1 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 C-1 enumerates the 16-bit guest-state fields.

**Table C-1. Encodings for 16-Bit Guest-State Fields (0000_10xx_xxxx_xxx0B)**

| Field Name | Index | Encoding |
|---|---|---|
| Guest ES selector | 000000000B | 00000800H |
| Guest CS selector | 000000001B | 00000802H |
| Guest SS selector | 000000010B | 00000804H |
| Guest DS selector | 000000011B | 00000806H |
| Guest FS selector | 000000100B | 00000808H |
| Guest GS selector | 000000101B | 0000080AH |
| Guest LDTR selector | 000000110B | 0000080CH |
| Guest TR selector | 000000111B | 0000080EH |

### C.1.2 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 C-2 enumerates the 16-bit host-state fields.

**Table C-2. 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 |

## C.2 FULL 64-BIT FIELDS

A value of 1 in bits 14:13 of an encoding indicates a full 64-bit field. There are full 64-bit fields only for controls and for guest state. As noted in Section 2.10.2, every full 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.

### C.2.1 Full 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 C-3 enumerates the full 64-bit control fields.

**Table C-3. Encodings for Full 64-Bit Control Fields (0010_00xx_xxxx_xxxAb)**

| Field Name | Index | Encoding |
|---|---|---|
| I/O bitmap A (full) | 000000000B | 00002000H |
| I/O bitmap A (high) | 000000000B | 00002001H |
| I/O bitmap B (full) | 000000001B | 00002002H |
| I/O bitmap B (high) | 000000001B | 00002003H |
| VM-exit MSR-store address (full) | 000000011B | 00002006H |
| VM-exit MSR-store address (high) | 000000011B | 00002007H |

**Table C-3. Encodings for Full 64-Bit Control Fields (0010_00xx_xxxx_xxxAb) (Contd.)**

| Field Name | Index | Encoding |
|---|---|---|
| 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 |
| TSC offset (full) | 000001000B | 00002010H |
| TSC offset (high) | 000001000B | 00002011H |
| Virtual-APIC page address (full) | 000001001B | 00002012H |
| Virtual-APIC page address (high) | 000001001B | 00002013H |

## C.2.2 Full 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 C-4 enumerates the full 64-bit guest-state fields.

**Table C-4. Encodings for Full 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 |
| Guest IA32_DEBUGCTL (high) | 000000001B | 00002803H |

## C.3 32-BIT FIELDS

A value of 2 in bits 14:13 of an encoding indicates a 32-bit field. As noted in Section 2.10.2, 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.

## C.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 C-5 enumerates the 32-bit control fields.

**Table C-5. Encodings for 32-Bit Control Fields (0100_00xx_xxxx_xxx0B)**

| Field Name | Index | Encoding |
|---|---|---|
| Pin-based VM-execution controls | 000000000B | 00004000H |
| 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 |
| 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 | 000001110B | 0000401CH |

## C.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 C-6 enumerates the 32-bit read-only data fields.

**Table C-6. 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 |

**Table C-6.  Encodings for 32-Bit Read-Only Data Fields (0100_01xx_xxxx_xxx0B) (Contd.)**

| Field Name | Index | Encoding |
|---|---|---|
| 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 |
| VMX-instruction information | 000000111B | 0000440EH |

## C.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 C-7 enumerates the 32-bit guest-state fields.

**Table C-7.  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 |

**Table C-7. Encodings for 32-Bit Guest-State Fields (0100_10xx_xxxx_xxx0B) (Contd.)**

| Field Name | Index | Encoding |
|---|---|---|
| Guest GS access rights | 000001111B | 0000481EH |
| Guest LDTR access rights | 000010000B | 00004820H |
| Guest TR access rights | 000010001B | 00004822H |
| Guest interruptibility information | 000010010B | 00004824H |
| Guest activity state | 000010011B | 00004826H |
| Guest IA32_SYSENTER_CS | 000010101B | 0000482AH |

Note that 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 IA-32 architecture. VM entry ensures that the high 16 bits of both these fields are cleared to 0.

### C.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 C-8.

**Table C-8. Encodings for 32-Bit Host-State Field (0100_11xx_xxxx_xxx0B)**

| Field Name | Index | Encoding |
|---|---|---|
| Host IA32_SYSENTER_CS | 000000000B | 00004C00H |

## C.4 NATURAL 64-BIT FIELDS

A value of 3 in bits 14:13 of an encoding indicates a natural 64-bit field. As noted in Section 2.10.2, 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.

## C.4.1 Natural 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 C-9 enumerates the natural 64-bit control fields.

**Table C-9. Encodings for Natural 64-Bit 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[a] | 000000111B | 0000600EH |

**NOTES**

a. If a future implementation supports more than 4 CR3-target values, they will be encoded consecutively following the 4 encodings given here.

## C.4.2 Natural 64-Bit Read-Only Data Fields

A value of 1 in bits 11:10 of an encoding indicates a read-only data field. These fields are distinguished by their index value in bits 9:1. Table C-10 enumerates the natural 64-bit read-only data fields.

**Table C-10. Encodings for Natural 64-Bit Read-Only Data Fields (0110_01xx_xxxx_xxx0B)**

| **Field Name** | **Index** | **Encoding** |
|---|---|---|
| Exit qualification | 000000000B | 00006400H |
| Guest linear address | 000000101B | 0000640AH |

## C.4.3 Natural 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 C-11 enumerates the natural 64-bit guest-state fields.

**Table C-11. Encodings for Natural 64-Bit Guest-State Fields (0110_10xx_xxxx_xxx0B)**

| Field Name | Index | Encoding |
|---|---|---|
| Guest CR0 | 000000000B | 00006800H |
| Guest CR3 | 000000001B | 00006802H |
| Guest CR4 | 000000010B | 00006804H |
| Guest ES base | 000000011B | 00006806H |
| Guest CS base | 000000100B | 00006808H |
| Guest SS base | 000000101B | 0000680AH |
| Guest DS base | 000000110B | 0000680CH |
| Guest FS base | 000000111B | 0000680EH |
| Guest GS base | 000001000B | 00006810H |
| Guest LDTR base | 000001001B | 00006812H |
| 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 |

Note that the base-address fields for ES, CS, SS, and DS in the guest-state area are defined to be natural 64-bit even though these fields are only 32-bits wide in the EM64T architecture. VM entry ensures that the high 32 bits of these fields are cleared to 0.

## C.4.4 Natural 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 C-12 enumerates the natural 64-bit host-state fields.

**Table C-12. Encodings for Natural 64-Bit Host-State Fields (0110_11xx_xxxx_xxx0B)**

| Field Name | Index | Encoding |
|---|---|---|
| Host CR0 | 000000000B | 00006C00H |
| Host CR3 | 000000001B | 00006C02H |
| Host CR4 | 000000010B | 00006C04H |
| Host FS base | 000000011B | 00006C06H |
| Host GS base | 000000100B | 00006C08H |
| Host TR base | 000000101B | 00006C0AH |
| Host GDTR base | 000000110B | 00006C0CH |
| Host IDTR base | 000000111B | 00006C0EH |
| Host IA32_SYSENTER_ESP | 000001000B | 00006C10H |
| Host IA32_SYSENTER_EIP | 000001001B | 00006C12H |
| Host RSP | 000001010B | 00006C14H |
| Host RIP | 000001011B | 00006C16H |

# INDEX