5-Level Paging and 5-Level EPT

White Paper

Revision 1.0

December 2016

Document Number: 335252-001

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

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

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

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

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

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

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

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

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Date
335252-001	1.0	• Initial Release	December 2016

1 Introduction

This document describes planned extensions to the Intel 64 architecture to expand the size of addresses that can be translated through a processor's memory-translation hardware.

Modern operating systems use address-translation support called **paging**. Paging translates **linear addresses** (also known as virtual addresses), which are used by software, to **physical addresses**, which are used to access memory (or memory-mapped I/O). Section 1.1 describes the 64-bit paging hardware on Intel 64 processors. Existing processors limit linear addresses to 48 bits. Chapter 2 describes paging extensions that would relax that limit to 57 linear-address bits.

Virtual-machine monitors (VMMs) use the **virtual-machine extensions** (**VMX**) to support guest software operating in a virtual machine. **VMX transitions** are control-flow transfers between the VMM and guest software. VMX transitions involve the loading and storing of various processor registers. Some of these registers are defined to contain linear addresses. Because of this, the operation of VMX transitions depends in part on the linear-address width supported by the processor. Section 1.2 describes the existing treatment of linear-address registers by VMX transitions, while Chapter 3 describes the changes required to support larger linear addresses.

VMMs may also use additional address-translation support called **extended page tables** (**EPT**). When EPT is used, paging produces **guest-physical addresses**, which EPT translates to physical addresses. Section 1.3 describes the EPT hardware on existing Intel 64 processors, which limit guest-physical addresses to 48 bits. Chapter 4 describes EPT extensions to support 57 guest-physical-address bits.

1.1 Existing Paging in IA-32e Mode

On processors supporting Intel 64 architecture, software typically references memory using **linear addresses**. Most modern operating systems configure processors to use **paging**, which translates linear addresses to physical addresses. The processor uses the resulting physical addresses to access memory.

IA-32e mode is a mode of processor execution that extends the older 32-bit operation, known as **legacy mode**. Software can enter IA-32e mode with the following algorithm.

1. Use the MOV CR instruction to set CR4.PAE[bit 5]. (Physical-address extension must be enabled to enter IA-32e mode.)
2. Use the WRMSR instruction to set bit 8 (LME) of the IA32_EFER MSR (index C0000080H).
3. Use the MOV CR instruction to load CR3 with the address of a PML4 table (see below).
4. Use the MOV CR instruction to set CR0.PG[bit 31].

A logical processor is in IA-32e mode whenever CR0.PG = 1 and IA32_EFER.LME = 1. This fact is reported in IA32_EFER.LMA[bit 10]. Software cannot set this bit directly; it is always the logical-AND of CR0.PG and IA32_EFER.LME.

In IA-32e mode, linear addresses are 64 bits in size.[1] However, the corresponding paging mode (currently called **IA-32e paging**) does not use all 64 linear-address bits.

IA-32e paging does not use all 64 linear-address bits because processors limit the size of linear addresses. This limit is enumerated by the CPUID instruction. Specifically, CPUID.80000008H:EAX[bits 15:8] enumerates the number of linear-address bits (the maximum linear-address width) supported by the processor. Existing processors enumerate this value as 48.

Note: Processors also limit the size of physical addresses and enumerate the limit using CPUID. CPUID.80000008H:EAX[bits 7:0] enumerates the number of physical-address bits supported by the processor, the maximum physical-address width. Existing processors have enumerated values up to 46. Software can use more than 32 physical-address bits only if **physical-address extension** has been enabled by setting CR4.PAE, bit 5 of control register CR4.

The enumerated limitation on the linear-address width implies that paging translates only the low 48 bits of each 64-bit linear address. After a linear address is generated but before it is translated, the processor confirms that the address uses only the 48 bits that the processor supports.

The limitation to 48 linear-address bits results from the nature of IA-32e paging, which is illustrated in Figure 1-1.

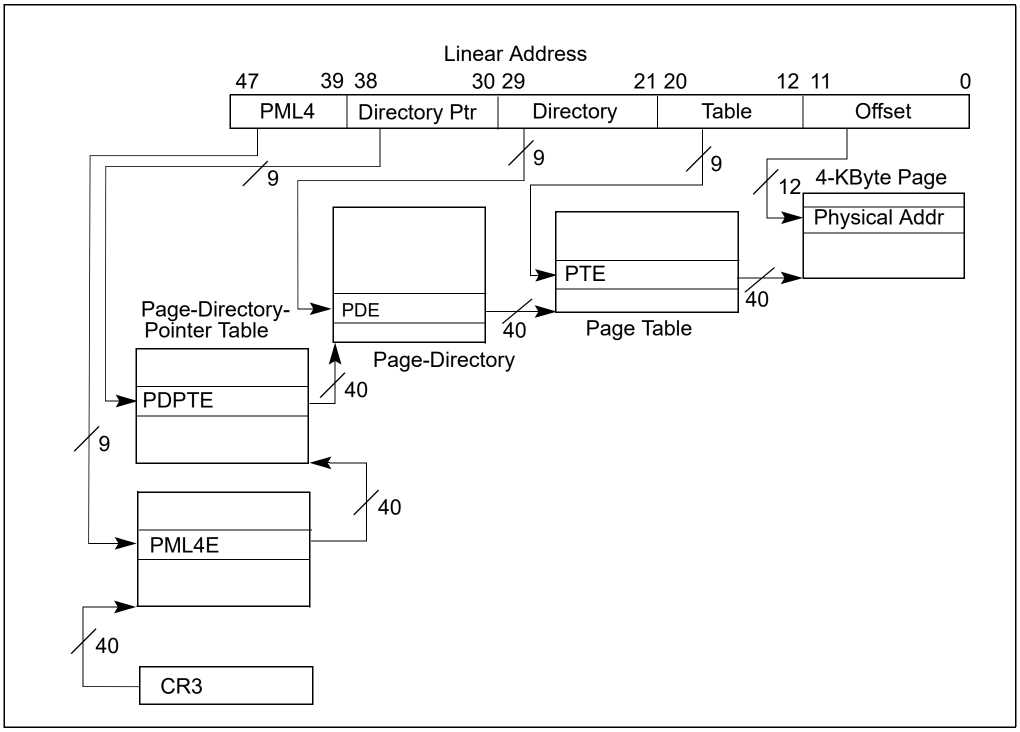

Figure 1-1. Linear-Address Translation Using IA-32e Paging

[1] IA-32e mode comprises two sub-modes: compatibility mode and 64-bit mode. In compatibility mode, software uses 32-bit addresses, which the processor zero-extends to 64-bit linear addresses. In 64-bit mode, software uses 64-bit addresses directly.

The processor performs IA-32e paging by traversing a 4-level hierarchy of **paging structures** whose root structure resides at the physical address in control register CR3. Each paging structure is 4-KBytes in size and comprises 512 8-byte entries. The processor uses the upper 36 bits of a linear address (bits 47:12), 9 bits at a time, to select paging-structure entries from the hierarchy.

Note: Figure 1-1 illustrates the translation of a linear address to a 4-KByte page. The paging process can be configured so that the translation of some linear addresses stops one or two levels earlier, translating instead to 2-MByte pages or 1-GByte pages.

In general, bits 51:12 of each paging-structure entry contain a 4-KByte aligned physical address. For each entry except the last, this address is that of the next paging structure; in the last entry, it is the physical address of a 4-KByte **page frame**. The final physical address is obtained by combining this page-frame address with the **page offset**, bits 11:0 of the original linear address.

Because only bits 47:0 of a linear address are used in address-translation, the processor reserves bits 63:48 for future expansion using a concept known as **canonicality**. A linear address is **canonical** if bits 63:47 of the address are identical. (Put differently, a linear address is canonical only if bits 63:48 are a sign-extension of bit 47, which is the uppermost bit used in linear-address translation.)

When a 64-bit linear address is generated to access memory, the processor first confirms that the address is canonical. If the address is not canonical, the memory access causes a fault, and the processor makes no attempt to translate the address.[1]

Intel 64 architecture includes numerous registers that are defined to hold linear addresses. These registers may be loaded using a variety of instructions. In most cases, these instructions cause a general-protection exception (#GP) if an attempt is made to load one of these registers with a value that is not canonical.

Physical-address bits in a paging-structure entry beyond the enumerated physical-address width are reserved. A page-fault exception (#PF) results if an attempt is made to access a linear address whose translation encounters a paging-structure entry that sets any of those bits.

1.2 Linear-Address Width and VMX Transitions

VM entries and VM exits manipulate numerous processor registers that contain linear addresses. The transitions respect the processor’s linear-address width in a manner based on canonicality.

Certain fields in the VMCS correspond to registers that contain linear addresses. VM entries confirm that most of those fields contain values that are canonical. Some registers, such as RIP and the LDTR base address, receive special treatment.

VM exits save into the VMCS the state of certain registers, some of which contain linear addresses. Because the processor generally ensures that the values in these registers are canonical (see Section 1.1), the values that VM exits save for these registers will generally be canonical.

[1] In general, an attempt to access memory using a linear address that is not canonical causes a general-protection exception (#GP). A stack-fault exception — #SS — occurs instead if the memory access was made using the SS segment.

VM exits also load from the VMCS certain registers, some of which contain linear addresses. Each VM exit ensures that the value of each of these registers is canonical. Specifically, bits 47:0 of the register are loaded from the field in the host-state area; the value of bit 47 is then sign-extended into bits 63:48 of the register.

1.3 Existing Extended Page Tables (EPT)

Most Intel 64 processors supporting VMX also support an additional layer of address translation called **extended page tables** (**EPT**).

VM entry can be configured to activate EPT for guest software. When EPT is active, the addresses used and produced by paging (Section 1.1) are not used as physical addresses to reference in memory. Instead, the processor interprets them as **guest-physical addresses**, and translates them to physical addresses in a manner determined by the VMM. (This translation from guest-physical to physical applies not only to the output of paging but also to the addresses that the processor uses to reference the guest paging structures.)

If the EPT translation process cannot translate a guest-physical address, it causes an **EPT violation**. (EPT violations may also occur when an access to a guest-physical address violates the permissions established by EPT for that guest-physical address.) An EPT violation is a VMX-specific exception, usually causing a VM exit.

As noted in Section 1.1, existing processors limit physical addresses to 46 bits. That limit applies also to guest-physical addresses. As a result, guest-physical addresses that set bits beyond this limit are not translated by EPT. (For example, a page fault results if linear-address translation encounters a paging-structure entry with such an address.) Because of this, existing EPT has been limited to translating only 48 guest-physical-address bits.

The existing EPT translation process is analogous to the paging process that was illustrated earlier in Figure 1-1. Like 4-level paging, the processor implements EPT by traversing a 4-level hierarchy of 4-KByte **EPT paging structures**. The last EPT paging-structure entry contains the upper bits of the final physical address, while the lowest bits come from the original guest-physical address.

2 Expanding Linear Addresses: 5-Level Paging

2.1 5-Level Paging: Introduction

5-level paging is a new paging mode that will be available in IA-32e mode. As its name suggests, it will translate linear addresses by traversing a 5-level hierarchy of paging structures. Because the process is otherwise unmodified, 5-level paging extends the processor's linear-address width to 57 bits. (The additional 9 bits are used to select an entry from the fifth level of the hierarchy.) For clarity, the paging mode formerly called IA-32e paging will now be called **4-level paging**.

The remainder of this chapter specifies the architectural changes that define and are entailed by 5-level paging. Section 2.2 specifies how the CPU enumerates the new feature and how it is enabled by software. Section 2.3 describes changes to the process of linear-address generation, as well as a revision to the concept of canonicality. Section 2.4 details how 5-level paging translates linear addresses. Section 2.5 clarifies how the processor treats loads of registers containing linear addresses, while Section 2.6 to Section 2.8 consider interactions with various other features. (Interactions with the virtual-machine extensions are specified in Chapter 3.)

2.2 Enumeration and Enabling

This section describes how processors enumerate to software support for 5-level paging and related features and also how software enables the processor to use that support.

2.2.1 Enumeration by CPUID

Processors supporting the Intel 64 architecture typically use the CPUID instruction to enumerate to software specific processor functionality. Those processors that support 5-level paging enumerate that fact through a new feature flag as well as through changes in how related features are reported:

- CPUID.(EAX=07H, ECX=0):ECX[bit 16] is a new feature flag that will enumerate basic support for 5-level paging. All older processors clear this bit. A processor will set this bit if and only if it supports 5-level paging.
- As noted in Section 1.1, CPUID.80000008H:EAX[bits 15:8] enumerates the maximum linear-address width supported by the processor. All older processors that support Intel 64 architecture enumerated this value as 48. Processors that support 5-level paging will instead enumerate this value as 57.
- As noted in Section 1.1, CPUID.80000008H:EAX[bits 7:0] enumerates the maximum physical-address width supported by the processor. Processors that support Intel 64 architecture have enumerated at most 46 for this value. Processors that support 5-level paging are expected to enumerate higher values, up to 52.
- CPUID.(EAX=07H,ECX=0H):ECX.MAWAU[bits 21:17] is an existing field that enumerates the user MPX address-width adjust (MAWAU). This value specifies the number of linear-address bits above 48 on which the BNDLDX and BNDSTX instructions operate in 64-bit mode when CPL = 3.

Older processors that support Intel® MPX enumerated 0 for this value. Processors that support 5-level paging may enumerate either 0 or 9, depending on configuration by system software. See Section 2.7 for more details on how BNDLDX and BNDSTX use MAWAU and how system software determines its value.

- CPUID.(EAX=12H,ECX=0H):EDX[bits 15:8] is an existing field that enumerates information that specifies the maximum supported size of a 64-bit enclave. If the value enumerated is n, the maximum size is 2^n. Older processors that support Intel® SGX enumerated at most 47 for this value. Processors that support 5-level paging are expected to enumerate this value as 56.

2.2.2 Enabling by Software

Section 1.1 identified an algorithm by which software can enter IA-32e mode. On processors that do not support 5-level paging, this algorithm enables 4-level paging. On processors that support 5-level paging, it can be adapted to enable 5-level paging instead.

Processors that support 5-level paging allow software to set a new enabling bit, CR4.LA57[bit 12].[1] A logical processor in IA-32e mode (IA32_EFER.LMA = 1) uses 5-level paging if CR4.LA57 = 1. Outside of IA-32e mode (IA32_EFER.LMA = 0), the value of CR4.LA57 does not affect paging operation.

The following items detail how a logical processor determines the current paging mode.

- If CR0.PG = 0, paging is disabled.
- If IA32_EFER.LMA = 0, one of the legacy 32-bit paging modes is used (depending on the value of legacy paging-mode bits in CR4).[2]
- If IA32_EFER.LMA = 1 and CR4.LA57 = 0, 4-level paging is used.
- If IA32_EFER.LMA = 1 and CR4.LA57 = 1, 5-level paging is used.

Software can thus use the following algorithm to enter IA-32e mode with 5-level paging.

1. Use the MOV CR instruction to set CR4.PAE and CR4.LA57.
2. Use the WRMSR instruction to set IA32_EFER.LME.
3. Use the MOV CR instruction to load CR3 with the address of a PML5 table (see Section 2.4).
4. Use the MOV CR instruction to set CR0.PG.

The processor allows software to modify CR4.LA57 only outside of IA-32e mode. In IA-32e mode, an attempt to modify CR4.LA57 using the MOV CR instruction causes a general-protection exception (#GP).

2.3 Linear-Address Generation and Canonicality

As noted in Section 1.1, processors with a linear-address width of 48 bits reserve linear-address bits 63:48 for future expansion. Linear addresses that use only bits 47:0 (because bits 63:48 are a sign-extension of bit 47) are called **canonical**.

[1] Software can set CR4.LA57 only if CPUID.(EAX=07H, ECX=0):ECX[bit 16] is enumerated as 1.
[2] Recall that IA32_EFER.LMA is the logical-AND of CR0.PG and IA32_EFER.LME.

When a 64-bit linear address is generated to access memory, the processor first confirms that the address is canonical. If the address is not canonical, the memory access causes a fault, and the address is not translated.

Processors that support 5-level paging can translate 57-bit linear addresses when 5-level paging is enabled. But if software has enabled only 4-level paging, such a processor can translate only 48-bit linear addresses. This fact motivates the definition of two levels of canonicality.

A linear address is **48-bit canonical** if bits 63:47 of the address are identical. Similarly, an address is **57-bit canonical** if bits 63:56 of the address are identical. Any linear address is that 48-bit canonical is also 57-bit canonical.

When a 64-bit linear address is generated to access memory, a processor that supports 5-level paging checks for canonicality based on the current paging mode: if 4-level paging is enabled, the address must be 48-bit canonical; if 5-level paging is enabled, the address need only be 57-bit canonical. If the appropriate canonicality is not observed, the memory access causes a fault.

2.4 5-Level Paging: Linear-Address Translation

As noted in Section 2.2.2, a logical processor uses 5-level paging if IA32_EFER.LMA = 1 and CR4.LA57 = 1.

Like 4-level paging, 5-level paging translates linear addresses using a hierarchy of in-memory paging structures. Because 5-level paging increases the linear-address width to 57 bits (from the 48 bits supported by 4-level paging), 5-level paging allows up to 128 PBytes of linear-address space to be accessed at any given time.

Also like 4-level paging, 5-level paging uses CR3 to locate the first paging-structure in the hierarchy. (CR3 has the same mode-specific format with 5-level paging as it does with 4-level paging.) The following items describe in more detail the changes that 5-level paging makes to the translation process.

- Translation begins by identifying a 4-KByte naturally aligned PML5 table. It is located at the physical address specified in bits 51:12 of CR3. A PML5 table comprises 512 64-bit entries (PML5Es). A PML5E is selected using the physical address defined as follows.
  - — Bits 51:12 are from CR3.
  - — Bits 11:3 are bits 56:48 of the linear address.
  - — Bits 2:0 are all 0.

  Because a PML5E is identified using bits 56:48 of the linear address, it controls access to a 256-TByte region of the linear-address space. The format of a PML5E is given in Table 2-1.

Table 2-1. Format of a PML5 Entry (PML5E) that References a PML4 Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a PML4 table.
1 (R/W)	Read/write; if 0, writes may not be allowed to the 256-TByte region controlled by this entry.
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 256-TByte region controlled by this entry.
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the PML4 table referenced by this entry.

Table 2-1. Format of a PML5 Entry (PML5E) that References a PML4 Table (Continued)

Bit Position(s)	Contents
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the PML4 table referenced by this entry.
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation.
6	Ignored.
7 (PS)	Reserved (must be 0).
11:8	Ignored.
M–1:12	Physical address of 4-KByte aligned PML4 table referenced by this entry.
51:M	Reserved (must be 0).
62:52	Ignored.
63	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 256-TByte region controlled by this entry); otherwise, reserved (must be 0).

- The next step of the translation process identifies a 4-KByte naturally aligned PML4 table. It is located at the physical address specified in bits 51:12 of the PML5E (see Table 2-1). A PML4 table comprises 512 64-bit entries (PML4Es). A PML4E is selected using the physical address defined as follows.
  - — Bits 51:12 are from the PML5E.
  - — Bits 11:3 are bits 47:39 of the linear address.
  - — Bits 2:0 are all 0.

  As is normally the case when accessing a paging-structure entry, the memory type used to access the PML4E is based in part on the PCD and PWT bits in the PML5E.

  Because a PML4E is identified using bits 56:39 of the linear address, it controls access to a 512-GByte region of the linear-address space.

Once the PML4E is identified, bits 38:0 of the linear address determine the remainder of the translation process exactly as is done for 4-level paging. As suggested in Table 2-1, the values of bit 1, bit 2, and bit 63 of the PML5E are used normally (in combination with the corresponding bits in other paging-structure entries) to determine access rights. The accessed flag (bit 5) in the PML5E is updated as is done for other paging-structure entries.

The operation of 5-level paging is illustrated in Figure 2-1.

2.5 Linear-Address Registers and Canonicality

Intel 64 architecture includes numerous registers that are defined to hold linear addresses. These registers may be loaded using a variety of instructions. As noted in Section 1.1, each of these instructions typically causes a general-protection exception (#GP) if an attempt is made to load a linear-address register with a value that is not canonical.

As noted in Section 2.3, processors that support 5-level paging use two definitions of canonicality: 48-bit canonicality and 57-bit canonicality. This section describes how such a processor checks the canonicality of the values being loaded into the linear-address registers. One approach is used for operations that load RIP (the instruction pointer; see Section 2.5.1) and another is used for those that load other registers (see Section 2.5.2).

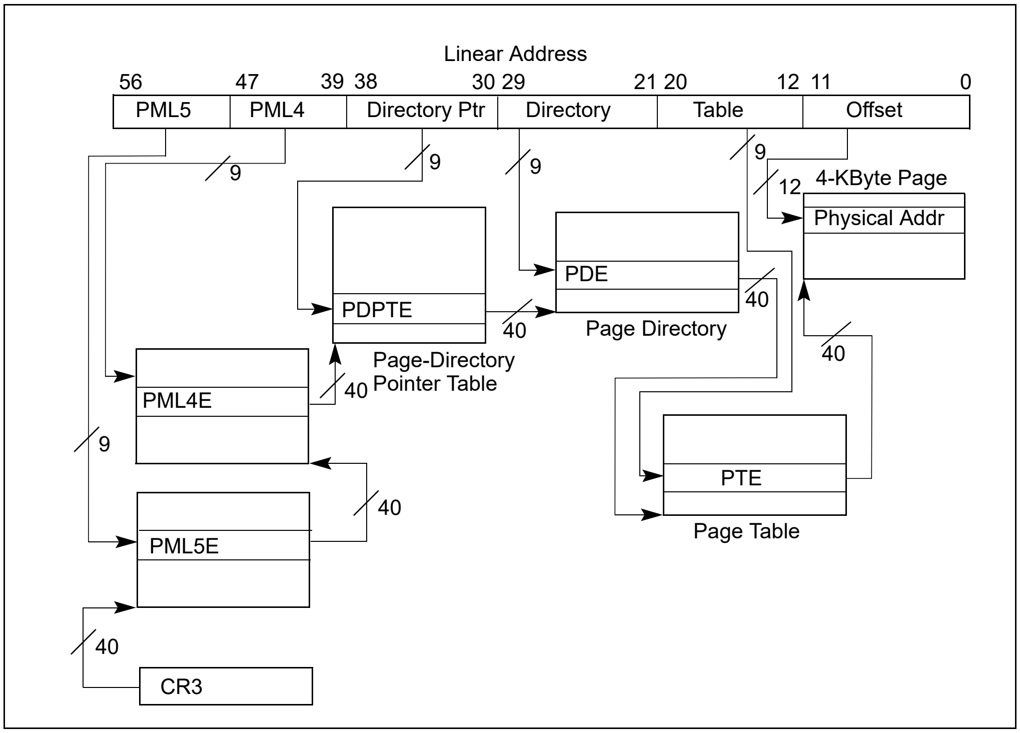

Figure 2-1. Linear-Address Translation Using 5-Level Paging

2.5.1 Canonicality Checking on RIP Loads

The RIP register contains the offset of the current instruction pointer within the CS segment. Because the processor treats the CS base address as zero in 64-bit mode, the value of the RIP register in that mode is the linear address of the instruction pointer.

Operations that load RIP (including both instructions such as JMP as well as control transfers through the IDT) check first whether the value to be loaded is canonical relative to the current paging mode. If the processor determines that the address is not canonical, the RIP load is not performed and a general-protection exception (#GP) occurs.

Note: An instruction that would load RIP with a non-canonical address faults, meaning that the return instruction pointer of the fault handler is the address of the faulting instruction and not the non-canonical address whose load was attempted.

The canonicality checking performed by these operations uses 48-bit canonicality when 4-level paging is active. When 5-level paging is active, the checking is relaxed to require only 57-bit canonicality.

The SYSCALL and SYSENTER instructions load RIP from the IA32_LSTAR and IA32_SYSENTER_EIP MSRs, respectively. On processors that support only 4-level paging, these instructions do not check that the values being loaded are canonical because the WRMSR instruction ensures that each of these MSRs contains a value that is 48-bit canonical. On processors that support 5-level paging, the checking by WRMSR is relaxed to 57-bit canonicality (see Section 2.5.2). On such processors, an execution of SYSCALL or SYSENTER with 4-level paging checks that the value being loaded into RIP is 48-bit canonical.[1]

The normal advancing of the instruction pointer to the next instruction boundary may result in the RIP register holding a non-canonical address. The fetch of the next instruction from that non-canonical address will result in a general-protection exception as indicated in Section 2.3. In this case, the return instruction pointer of the fault handler will be that non-canonical address.

2.5.2 Canonicality Checking on Other Loads

In addition to RIP, the CPU maintains numerous other registers that hold linear addresses:

- GDTR and IDTR (in their base-address portions).
- LDTR, TR, FS, and GS (in the base-address portions of their hidden descriptor caches).
- The debug-address registers (DR0 through DR3), which hold the linear addresses of breakpoints.
- The following MSRs: IA32_BNDCFGS, IA32_DS_AREA, IA32_KERNEL_GS_BASE, IA32_LSTAR, IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B, IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B, IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B, IA32_RTIT_ADDR3_A, IA32_RTIT_ADDR3_B, IA32_SYSENTER_EIP, and IA32_SYSENTER_ESP.
- The x87 FPU instruction pointer (FIP).
- The user-mode configuration register BNDCFGU, used by Intel® MPX.

With a few exceptions, the processor ensures that the addresses in these registers are always canonical in the following ways.

- Some instructions fault on attempts to load a linear-address register with a non-canonical address:
  - An execution of the LGDT or LIDT instruction causes a general-protection exception (#GP) if the base address specified in the instruction’s memory operand is not canonical.
  - An execution of the LLDT or LTR instruction causes a #GP if the base address to be loaded from the GDT is not canonical.
  - An execution of WRMSR, WRFSBASE, or WRGSBASE causes a #GP if it would load the base address of either FS or GS with a non-canonical address.
  - An execution of WRMSR causes a #GP if it would load any of the following MSRs with a non-canonical address: IA32_BNDCFGS, IA32_DS_AREA, IA32_FS_BASE, IA32_GS_BASE, IA32_KERNEL_GS_BASE, IA32_LSTAR, IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B, IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B, IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B, IA32_RTIT_ADDR3_A, IA32_RTIT_ADDR3_B, IA32_SYSENTER_EIP, or IA32_SYSENTER_ESP.[2]

[1] The SYSRET and SYSEXIT instructions, which complement SYSCALL and SYSENTER, load RIP from RCX and RDX, respectively. Even before 5-level paging, these instructions checked the canonicality of the value to be loaded into RIP. As with other instructions that load RIP, this checking will be based on the current paging mode.

[2] Such canonicality checking may apply also when the WRMSR instruction is used to load some non-architectural MSRs (not listed here) that hold a linear address.

- An execution of XRSTORS causes a #GP if it would load any of the following MSRs with a non-canonical address: IA32_RTIT_ADDR0_A, IA32_RTIT_ADDR0_B, IA32_RTIT_ADDR1_A, IA32_RTIT_ADDR1_B, IA32_RTIT_ADDR2_A, IA32_RTIT_ADDR2_B, IA32_RTIT_ADDR3_A, and IA32_RTIT_ADDR3_B.

This enforcement always uses the enumerated maximum linear-address width and is independent of the current paging mode. Thus, a processor that supports 5-level paging will allow the instructions mentioned above to load these registers with addresses that are 57-bit canonical but not 48-bit canonical — **even if 4-level paging is active**. (As a result, instructions that store these values — SGDT, SIDT, SLDT, STR, RDFSBASE, RDGSBASE, RDMSR, XSAVE, XSAVEC, XSAVEOPT, and XSAVES — may save addresses that are 57-bit canonical but not 48-bit canonical, even if 4-level paging is active.)

- The FXRSTOR, XRSTOR, and XRSTORS instructions ignore attempts to load some of these registers with non-canonical addresses:
  - Loads of FIP ignore any bits in the memory image beyond the enumerated maximum linear-address width. The processor sign-extends to most significant bit (e.g., bit 56 on processors that support 5-level paging) to ensure that FIP is always canonical.
  - Loads of BNDCFGU (by XRSTOR or XRSTORS) ignore any bits in the memory image beyond the enumerated maximum linear-address width. The processor sign-extends to most significant bit (e.g., bit 56 on processors that support 5-level paging) to ensure that BNDCFGU is always canonical.
- Every non-control x87 instruction loads FIP. The value loaded is always canonical relative to the current paging mode: 48-bit canonical if 4-level paging is active, and 57-bit canonical if 5-level paging is active.

DR0 through DR3 can be loaded with the MOV to DR instruction. The instruction allows those registers to be loaded with non-canonical addresses. The MOV from DR instruction will return the value last loaded with the MOV to DR instruction, even if the address is not canonical. Breakpoint address matching is supported only for canonical linear addresses.

2.6 Interactions with TLB-Invalidation Instructions

Intel 64 architecture includes three instructions that may invalidate TLB entries for the linear address of an instruction operand: INVLPG, INVPCID, and INVVPID. The following items describe how they are affected by linear-address width.

- The INVLPG instruction takes a memory operand. It invalidates any TLB entries that the logical processor is caching for the linear address of that operand for the current linear address space. The instruction does not fault if that address is not canonical relative to the current paging mode (e.g., is not 48-bit canonical when 4-level paging is active). However, no invalidation is performed because the processor does not cache TLB entries for addresses that are not canonical relative to the current paging mode.
- The INVPCID instruction takes a register operand (INVPCID type) and a memory operand (INVPCID descriptor). If the INVPCID type is 0, the instruction invalidates any TLB entries that the logical processor is caching for the linear address and PCID specified in the INVPCID descriptor. If the linear address is not canonical relative the linear-address width supported by the processor, the instruction causes a general-protection exception (#GP). If the processor supports 5-level paging, the instruction will not cause such a #GP for an address that is 57-bit canonical, regardless of paging mode, even if 4-level paging is active and the address is not 48-bit canonical.

- The INVVPID instruction takes a register operand (INVVPID type) and a memory operand (INVVPID descriptor). If the INVPCID type is 0, the instruction invalidates any TLB entries that the logical processor is caching for the linear address and VPID specified in the INVVPID descriptor. If the linear address is not canonical relative the linear-address width supported by the processor, the instruction fails.[1] If the processor supports 5-level paging, the instruction will not fail for an address that is 57-bit canonical, regardless of paging mode, even if 4-level paging is active and the address is not 48-bit canonical.

2.7 Interactions with Intel® MPX

The Intel® Memory Protection Extensions (Intel® MPX) define a set of 4 bound registers, each of which software can associate with a specific pointer in memory. Intel MPX includes two instructions — BNDLDX and BNDSTX — that allow software to load from or store into memory the bounds associated with a particular pointer in memory.

The BNDLDX and BNDSTX instructions each take a bound register and a memory operand (the associated pointer). Each of these parses the linear address of the memory operand to traverse a hierarchical data structure in memory. In 64-bit mode, these instructions do not necessarily use all the bits in the supplied 64-bit addresses. The number of bits used is 48 plus a value called the **MPX address-width adjust** (**MAWA**).

The value of MAWA depends on CPL; the current paging mode (4-level paging or 5-level paging); and, if 5-level paging is active, the value of a new MSR. Processors that support both Intel MPX and 5-level paging support the IA32_MPX_LAX MSR (MSR index 1000H). Only bit 0 of the MSR is defined.

If CPL < 3, the supervisor MAWA (**MAWAS**) is used. The value of MAWAS is determined by the setting of CR4.LA57. If CR4.LA57 = 0 (4-level paging is active; recall that MAWA is relevant only in 64-bit mode), the value of MAWAS is 0. If CR4.LA57 = 1 (5-level paging is active), the value of MAWAS is 9. The value of MAWAS is not enumerated by the CPUID instruction.

If CPL = 3, the user MAWA (**MAWAU**) is used. The value of MAWAU is determined as follows. If CR4.LA57 = 0 or IA32_MPX_LAX[bit 0] = 0, the value of MAWAU is 0. If CR4.LA57 = 1 and IA32_MPX_LAX[bit 0] = 1, the value of MAWAU is 9. The current value of MAWAU is enumerated in CPUID.(EAX=07H,ECX=0H):ECX.MAWAU[bits 21:17].

The following items specify how an execution of the BNDLDX and BNDSTX instructions in 64-bit mode parses a linear address to traverse a hierarchical data structure.

- A bound directory is located at the 4-KByte aligned linear address specified in bits 63:12 of BNDCFGx.[2] A BDE is selected using the LAp (linear address of pointer to a buffer) to construct a 64-bit offset as follows:
  - — bits 63:31+MAWA are 0;
  - — bits 30+MAWA:3 are LAp[bits 47+MAWA:20]; and
  - — bits 2:0 are 0.

[1] INVVPID is a VMX instruction. In response to certain conditions, execution of a VMX may **fail**, meaning that it does not complete its normal operation. When a VMX instruction fails, control passes to the next instruction (rather than to a fault handler) and a flag is set to report the failure.

[2] If CPL < 3, BNDCFGS is used; if CPL = 3, BNDCFGU is used.

The address of the BDE is the sum of the bound-directory base address (from BNDCFGx) plus this 64-bit offset.

If either BNDLDX or BNDSTX is executed inside an enclave, the instruction operates as if MAWAU = 0 (regardless of the values of CR4.LA57 and IA32_MPX_LAX[bit 0]).

- The processor uses bits 63:3 of the BDE as the 8-byte aligned address of a bound table (BT). A BTE is selected using the LAp (linear address of pointer to a buffer) to construct a 64-bit offset as follows:

  — bits 63:22 are 0;

  — bits 21:5 are LAp[bits 19:3]; and

  — bits 4:0 are 0.

  The address of the BTE is the sum of the bound-table base address (from the BDE) plus this 64-bit offset.

A bound directory comprises $2^{28+\mathrm{MAWA}}$ 64-bit entries (BDEs);[1] thus, the size of a bound directory in 64-bit mode is $2^{1+\mathrm{MAWA}}$ GBytes. A bound table comprises 2^{17} 32-byte entries (BTEs); thus, the size of a bound table in 64-bit mode is 4 MBytes (independent of MAWA).

2.8 Interactions with Intel® SGX

Intel® Software Guard Extensions (Intel® SGX) define new processor functionality that is implemented as SGX leaf functions within the ENCLS (supervisor) and ENCLU (user) instructions.

The SGX leaf functions include memory accesses using linear addresses normally. When executed in 64-bit mode, the linear address are 64 bits in width and are subject to the normal treatment of accesses to memory with 64-bit linear addresses (see Section 2.3). In addition, some of the leaf functions apply specific architectural checks related to linear-address width. The following items detail these checks and how they are defined for processors that support 5-level paging.

- The ECREATE leaf function of ENCLS creates a new enclave by creating a new SGX enclave control structure (SECS). For a 64-bit enclave, the processor checks whether the enclave base linear address (specified in the SECS) is canonical, generating a general-protection exception (#GP) if it is not. On processors that support 5-level paging, this check is for 57-bit canonicality, regardless of the current paging mode.

  In addition to checking the canonicality of the enclave base linear address, ECREATE confirms that the enclave size (specified in the SECS) is not greater than the maximum size supported by the processor (if the enclave size is too large, ECREATE generates a #GP). As noted in Section 2.2.1, older processors supported 64-bit enclaves with sizes up to 2^{47} bytes; processors that support 5-level paging are expected to support enclaves with sizes up to 2^{56} bytes.

  If bits 4:3 of the enclave's XSAVE feature request mask (XFRM) are set (indicating that Intel MPX will be enabled during execution of the enclave), ECREATE generates a #GP if the enclave's size is greater than 2^{48} bytes, even if the processor enumerates support for larger enclaves.

- The EENTER and ERESUME leaf functions of ENCLU transfer control flow to an entry point within a specified enclave. For entry to a 64-bit enclave, the processor checks

[1] A bound directory used in a 64-bit enclave always comprises 2^{28} 64-bit BDEs and thus has a size of 2 GBytes.

whether certain linear addresses are canonical, generating a general-protection exception (#GP) if any one is not. The following items detail these checks.

  - — The linear address of the specified entry point must be canonical. If 4-level paging is active, it must be 48-bit canonical; if 5-level paging is active, it must be 57-bit canonical.
  - — The linear address of the asynchronous exit point (AEP — the address to which the processor transfers control on an asynchronous enclave exit) must be canonical. If 4-level paging is active, it must be 48-bit canonical; if 5-level paging is active, it must be 57-bit canonical.
  - — The enclave values for the base addresses of the FS and GS segments must be canonical. On processors that supports 5-level paging, these checks are for 57-bit canonicality, regardless of the current paging mode.

- The EEXIT leaf function exits the currently executing enclave and branches to a specified address. For an exit from a 64-bit enclave, the processor checks whether that target linear address is canonical, generating a general-protection exception (#GP) if it is not. If 4-level paging is active, it must be 48-bit canonical; if 5-level paging is active, it need only be 57-bit canonical.

As noted in Section 2.7, executions of BNDLDX and BNDSTX in a 64-bit enclave always operate as if MAWAU = 0.

3 Linear-Address Expansion and VMX Transitions

As noted in Section 1.2, VM entries and VM exits manipulate numerous processor registers that contain linear addresses. The transitions respect the processor's linear-address width in a manner based on canonicality.

As discussed in Chapter 2, processors that support 5-level paging expand the linear-address width from 48 bits to 57 bits. That expansion changes the operation of VMX transitions. Changes to VM entries are detailed in Section 3.1, while changes to VM exits are given in Section 3.2.

3.1 Linear-Address Expansion and VM Entries

Certain fields in the VMCS correspond to registers that contain linear addresses. VM entries confirm those fields contain values that are canonical. This checking is based on the linear-address width supported by the processor (e.g., is based on 57-bit canonicality if the processor supports 5-level paging). The following are the fields to which this applies.

- In the host-state area:
    - — The fields for the IA32_SYSENTER_EIP and IA32_SYSENTER_ESP MSRs.
    - — The base-address fields for FS, GS, TR, GDTR, and IDTR.
- In the guest-state area:
    - — The fields for the IA32_SYSENTER_EIP and IA32_SYSENTER_ESP MSRs.
    - — The base-address fields for FS, GS, TR, GDTR, and IDTR.
    - — The base-address field for LDTR (if LDTR will be usable).
    - — The field for the IA32_BNDCFGS MSR (if VM entry is loading that MSR).

A VM entry to 64-bit mode also performs a check on the RIP field in the guest-state area of the current VMCS. If the VM entry would result in 4-level paging, it checks that bits 63:48 of the guest RIP field are identical; if it would result in 5-level paging, that check is on bits 63:57.[1]

3.2 Linear-Address Expansion and VM Exits

VM exits save the state of certain registers into the guest-state area of the VMCS. Some of these registers contain linear addresses. As discussed in Section 1.1, the CPU generally ensures that the values in these registers respect the CPU's linear-address width. As a result, the values the VM exits save for these registers will do the same.

[1] Note that these checks do not confirm that the guest RIP field is canonical relative to the paging mode being entered. For example, bits 63:47 are identical in a 48-bit canonical address. However, VM entry to 4-level paging may load RIP with a value in which bit 47 differs from that of bits 63:48.

There is a special case for LDTR base address. If LDTR was not usable at the time of a VM exit, the value saved for the base address is undefined. However, this undefined value is always 48-bit canonical on processors that do not support 5-level paging and is always 57-bit canonical on processors that do support 5-level paging.

VM exits load the state of certain registers from the host-state area of the VMCS. Some of these registers contain linear addresses. Each VM exit ensures that the value of each of the following registers is canonical: the IA32_SYSENTER_EIP and IA32_SYSENTER_ESP MSRs; and the base addresses for FS, GS, TR, GDTR, and IDTR. How this is done depends on whether the processor supports 5-level paging.

- If the processor does not support 5-level paging, bits 47:0 of the register are loaded from the field in the host-state area; the value of bit 47 is then sign-extended into bits 63:48 of the register.
- If the processor does support 5-level paging, bits 56:0 of the register are loaded from the field in the host-state area; the value of bit 56 is then sign-extended into bits 63:57 of the register.

Again, there is a special case for LDTR. LDTR is always unusable after a VM exit. Its base address may be loaded with an undefined value. This undefined value is always 48-bit canonical on processors that do not support 5-level paging and is always 57-bit canonical on processors that do support 5-level paging.

4 5-Level EPT

5-level EPT is a new mode for EPT. As its name suggests, it will translate guest-physical addresses by traversing a 5-level hierarchy of EPT paging structures. Because the process is otherwise unmodified, 5-level paging extends the processor's guest-physical-address width to 57 bits. (The additional 9 bits are used to select an entry from the fifth level of the hierarchy.) For clarity, the original EPT mode will now be called **4-level EPT**.

The remainder of this chapter specifies architectural changes to 4-level EPT as well as those that define and are entailed by 5-level EPT. Section 4.1 describes how the expansion of the guest-physical-address width affects 4-level EPT. Section 4.2 specifies how the CPU enumerates 5-level EPT and how the feature is enabled by software. Section 4.3 details how 5-level EPT translates guest-physical addresses.

4.1 4-Level EPT: Guest-Physical-Address Limit

As explained in Section 1.3, 4-level EPT is limited to translating 48-bit guest-physical addresses.

This is not a problem on existing processors, because they limit the physical-address width to 46 bits (see Section 1.1). A processor's physical-address width also limits guest-physical addresses. That means that, on existing processors, any attempt to use a guest-physical address that sets a bit above the low 48 bits will cause a page-fault exception (#PF).

Processors that support 5-level paging are expected to support 52 physical-address bits. Such processors allow use of a guest-physical address that sets bits in the range 51:48; no #PF is generated.

A guest-physical address that sets bits in the range 51:48 cannot be translated by 4-level EPT. An attempt to access such an address when 4-level EPT is active causes an **EPT violation** (see Section 1.3).

EPT violations generate information about the exception in a value called the **exit qualification**. In general, EPT violations caused by attempts to access a guest-physical address that is too wide establish the exit qualification as is currently done for other EPT violations. Exceptions are made for bits 6:3 of the exit qualification, which report the access rights for the guest-physical address. The new EPT violations always clear these bits.

4.2 5-Level EPT: Enumeration and Enabling

This section describes how processors enumerate to software support for 5-level EPT and how software enables the processor to use that support.

4.2.1 Enumeration

Processors supporting EPT enumerate details related to EPT in the IA32_VMX_EPT_VPID_CAP MSR (index 48CH). Currently, IA32_VMX_EPT_VPID_CAP[bit 6] enumerates support for 4-level EPT. Processors that also support 5-level EPT will enumerate that fact by also setting IA32_VMX_EPT_VPID_CAP[bit 7].

The guest-physical-address width supported by a processor is not enumerated using the IA32_VMX_EPT_VPID_CAP MSR. This is because that width is always the same as the processor's maximum physical-address width as enumerated by CPUID.80000008H:EAX[bits 7:0].

4.2.2 Enabling by Software

A VMM enables EPT by setting the "enable EPT" VM-execution control in the current VMCS before using the VMCS for VM entry.

Specific details of EPT operation are determined by the extended-page-table pointer field (EPTP) in the VMCS. In particular, EPTP[bits 5:3] contain a value that is 1 less than the number of levels used by the EPT. On existing processors, this value must be 3, indicating 4-level EPT. (VM entry fails if a different value is used.) Processors that also support 5-level EPT will also allow the value 4 (indicating 5-level EPT).

In summary, VM entry on a processor that supports 5-level check EPTP[bits 5:3]. If the value is 3, the VM entry activates 4-level EPT. If the value is 4, the VM entry activates 5-level EPT. With any other value, VM entry fails.

4.3 5-Level EPT: Guest-Physical-Address Translation

Like 4-level EPT, 5-level EPT translates guest-physical addresses using a hierarchy of in-memory paging structures. Because 5-level EPT increases the guest-physical-address width to 57 bits (from the 48 bits supported by 4-level EPT), 5-level EPT allows up to 128 PBytes of guest-physical-address space to be accessed at any given time.

The following items describe in more detail the changes that 5-level EPT makes to the translation process.

- Translation begins by identifying a 4-KByte naturally aligned EPT PML5 table. It is located at the physical address specified in bits 51:12 of EPTP. An EPT PML5 table comprises 512 64-bit entries (EPT PML5Es). An EPT PML5E is selected using the physical address defined as follows.
  - Bits 63:52 are all 0.
  - Bits 51:12 are from EPTP.
  - Bits 11:3 are bits 56:48 of the guest-physical address.
  - Bits 2:0 are all 0.

Because an EPT PML5E is identified using bits 56:48 of the guest-physical address, it controls access to a 256-TByte region of the linear-address space. The format of an EPT PML5E is given in Table 4-1.

Table 4-1. Format of an EPT PML5 Entry (EPT PML5E)

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 256-TByte region controlled by this entry.
1	Write access; indicates whether writes are allowed from the 256-TByte region controlled by this entry.
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 256-TByte region controlled by this entry. If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 256-TByte region controlled by this entry.

Table 4-1. Format of an EPT PML5 Entry (EPT PML5E) (Continued)

Bit Position(s)	Contents
7:3	Reserved (must be 0).
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 256-TByte region controlled by this entry. Ignored if bit 6 of EPTP is 0.
9	Ignored.
10	Execute access for user-mode linear addresses. If the “mode-based execute control for EPT” VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 256-TByte region controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored.
M–1:12	Physical address of 4-KByte aligned EPT PML4 table referenced by this entry.
51:M	Reserved (must be 0).
63:52	Ignored.

- The next step of the translation process identifies a 4-KByte naturally aligned EPT PML4 table. It is located at the physical address specified in bits 51:12 of the EPT PML5E (see Table 4-1). An EPT PML4 table comprises 512 64-bit entries (EPT PML4Es). An EPT PML4E is selected using the physical address defined as follows.
  - — Bits 51:12 are from the EPT PML5E.
  - — Bits 11:3 are bits 47:39 of the guest-physical address.
  - — Bits 2:0 are all 0.

  Because an EPT PML4E is identified using bits 56:39 of the guest-physical address, it controls access to a 512-GByte region of the guest-physical-address space.

Once the EPT PML4E is identified, bits 38:0 of the guest-physical address determine the remainder of the translation process exactly as is done for 4-level EPT. As suggested in Table 4-1, the values of bits 2:0 and bit 10 of the EPT PML5E are used normally (in combination with the corresponding bits in other EPT paging-structure entries) to determine whether EPT violations occur. The accessed flag (bit 8) in the EPT PML5E is updated as is done for other EPT paging-structure entries.

4.4 5-Level EPT and EPTP Switching

The value of EPTP may be modified in VMX non-root operation by invoking VM function 0 (EPTP switching). This is done by executing the VMFUNC instruction with value 0 in the EAX register. Invocation of VM function 0 loads EPTP with a value selected from a data structure in memory.

Before loading EPTP in this way, the processor first confirms that the value to be loaded is valid. The definition of a valid EPTP value depends on whether the processor supports 5-level EPT.

- If the processor does not support 5-level EPT, an EPTP value in memory is considered valid if it would not cause VM entry to fail (e.g., it does not set any reserved bits).
- If the processor does support 5-level EPT, an EPTP value in memory is considered valid only if it would not cause VM entry to fail (as above) **and** if its value in bits 5:3 (which controls the number of EPT levels) is the same as that of the current value of EPTP.

The implication is that an invocation of VM function 0 cannot change the EPT mode between 4-level EPT and 5-level EPT.

5 Intel® Virtualization Technology for Directed I/O

Intel® Virtualization Technology for Directed I/O includes a feature called **DMA remapping**.

DMA remapping provides hardware support for isolation of device accesses to memory. When a device attempts to access system memory, DMA-remapping hardware intercepts the access and utilizes paging structures to determine whether the access can be permitted; it also determines the actual location to access.

The DMA-remapping hardware may support two levels of address translation. One level may translate a linear address to a guest-physical address, while a second level may remap the guest-physical address to physical address.

The first-level translation uses paging structures with the same format as those used for ordinary paging. The second-level translation uses paging structures with the same format as those used for EPT.

It is expected that, on platforms that support wider linear and guest-physical addresses (using 5-level paging and 5-level EPT, respectively), the DMA-remapping hardware will be similarly enhanced to support those wider addresses with 5-level translation processes.

This enhanced support for DMA remapping will be detailed in a future revision of the *Intel® Virtualization Technology for Directed I/O Architecture Specification*.