X86 HyperVisor

陈岳 清华大学计算机科学与技术系

About Me

Github: https://github.com/cylindrical2002

hcHyper: https://github.com/cylindrical2002/hcHyper

hcHyper 是我本学期操作系统课程的大作业，目前能够在 **ArceOS** 的环境支持下运行 **x86** 和 **ARM** 下的 **NimbOS** （ 我的工作 ），以及能够在 **RISC-V** 架构下运行 **Linux** 和 **rCore**（ 主要是齐呈祥学长的工作 ）。

本次以及下次的分享都是围绕 **hcHyper** 展开。

RUST X86 Hypervisor

在 hcHyper 之前，RUST 语言编写的 X86 Hypervisor 大致有：

- x86: https://github.com/rcore-os/RVM
- x86: https://github.com/rcore-os/RVM1.5
- x86: https://github.com/rcore-os/RVM-Tutorial

其中 RVM-Tutorial 是 hcHyper 中 X86 部分的前身。

X86 Virtualization Basics

此前已有两位学长分享过 **ARM** 和 **RISC-V** 两大架构在指令集层面的虚拟化支持，本次我来分享 **X86** 架构在指令集层面的虚拟化支持。

不同于 **ARM** 和 **RISC-V** 具有比较统一的指令集拓展，**X86** 架构在指令集层面的虚拟化支持比较复杂，**AMD** 和 **Intel** 使用的虚拟化指令并不完全一样。我在 **hcHyper** 中仅仅完成了 **Intel** 版本的 **X86** 虚拟化支持，因此我们本次的只分享 **Intel** 版本的 **X86** 在指令集层面对于虚拟化的支持

X86 Virtualization Basics

此前已有两位学长分享过 **ARM** 和 **RISC-V** 两大架构在指令集层面的虚拟化支持，本次我来分享 **X86** 架构在指令集层面的虚拟化支持。

不同于 **ARM** 和 **RISC-V** 具有比较统一的指令集拓展，**X86** 架构在指令集层面的虚拟化支持比较复杂，**AMD** 和 **Intel** 使用的虚拟化指令并不完全一样。我在 **hcHyper** 中仅仅完成了 **Intel** 版本的 **X86** 虚拟化支持，因此我们本次的只分享 **Intel** 版本的 **X86** 在指令集层面对于虚拟化的支持

X86 Virtualization Basics

- Host 模式：系统的最高特权级，运行 hypervisor。
- Guest 模式：比 host 模式特权级低，但能执行 OS 级别的特权指令，运行 guest OS 等需要 OS 级特权操作的软件。
- Hypervisor：也叫 virtual-machine monitors (VMM)，运行在具有最高特权级的 host 模式，拥有硬件资源的完全控制权限，并管理其上运行的 guest 软件。
- Guest 软件：运行在 guest 模式的软件（一般是一个操作系统，即 guest OS），比 hypervisor 特权级低，执行特定指令时会被 hypervisor 拦截（intercept），在受控情况下访问硬件资源。
- VM entry：从 host 模式切换到 guest 模式，开始执行 guest 软件的代码。
- VM exit：从 guest 模式切换回 host 模式（如执行特定指令、发生中断），开始执行 hypervisor 的代码。
- VCPU：由 hypervisor 虚拟出来的，运行 guest 软件所需的每 CPU 的私有状态。类似于传统 OS 中的线程，一个 guest OS 可具有多个 vCPU，vCPU 数量也可以多于物理 CPU 数量。
- Guest VM：即我们通常说的虚拟机，除了包含一个或多个 vCPU 外，还包含其他全局的系统状态，如 guest 物理内存、虚拟设备等。类似于传统 OS 中的进程，hypervisor 可创建多个 guest VM，各运行一个 guest OS。

Virtual-Machine eXtensions

支持虚拟化的 Intel 处理器支持 VMX 操作模式

Hypervisor 通过执行VMXON指令进入VMX操作模式。

Hypervisor 使用 VMLAUNCH 指令进行 Guest 启动

虚拟机通过 VMCALL 指令退出 Guest 的上下文，退出将控制转移到由 Hypervisor 指定的入口点。Hypervisor 可以根据 Guest 退出的原因采取适当的操作，并可以通过 VMRESUME 重新启动 Guest 。

Hypervisor 通过执行 VMXOFF 指令关闭自身并离开 VMX 操作模式。通过执行 VMXOFF 指令来实现。

Virtual-Machine eXtensions

在 **VMX** 操作模式下执行的操作称为 **VMX** 操作。**VMX** 操作分为 **Root** 和 **Non-Root** 两种。

VMX Root Operation 在虚拟机监视器（**VMM**）的上下文中执行，可以类比为 **RISC-V** 架构下在 **HS-Mode** 的操作，**通常**是由 **HyperVisor** 执行的。

VMX Non-Root Operation 是在虚拟机（**VM**）的上下文中执行的，可以类比为 **RISC-V** 架构下在 **VS-Mode** 和 **VU-Mode** 的操作，会受到虚拟机监视器的控制。

Virtual-Machine eXtensions

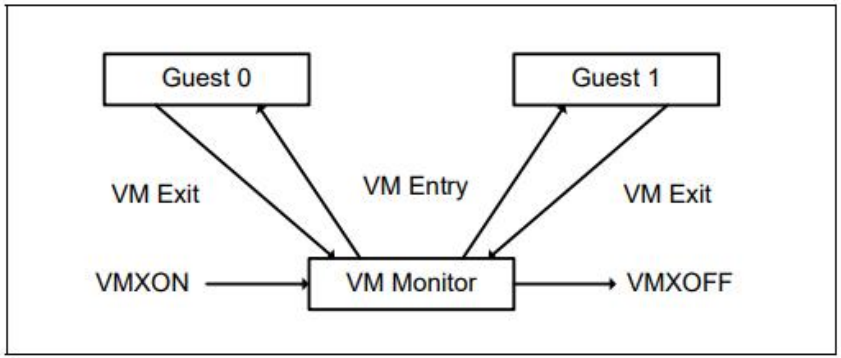
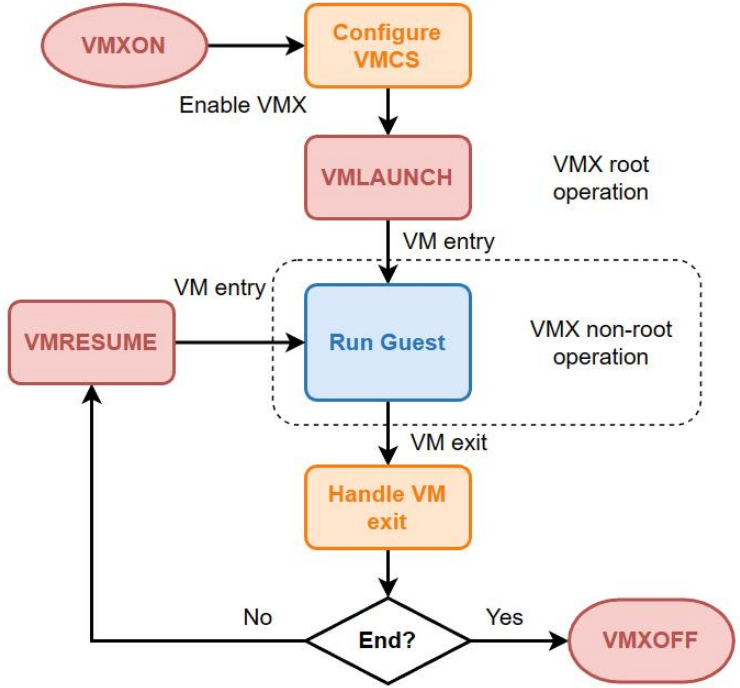


Figure 24-1. Interaction of a Virtual-Machine Monitor and Guests



VMXON /VMXOFF

在系统软件进入 **VMX** 操作之前，必须先发现处理器是否支持**VMX**。系统软件可以使用 **CPUID** 指令确定处理器是否支持**VMX**操作。如果 **CPUID.1:ECX.VMX = 1** ，则表示支持 **VMX** 操作模式。

VMX 架构被设计为可扩展的，未来处理器可以支持第一代 **VMX** 架构中没有的其他功能。可扩展 **VMX** 功能的可用性通过一组 **VMX** 能力 **MSR** 向软件报告。

VMXON /VMXOFF

在 **Hypervisor** 进入 **VMX** 操作模式之前，通过设置 **CR4.VMXE = 1** 来启用 **VMX**。执行 **VMXON** 指令后，进入 **VMX** 操作模式。如果以 **CR4.VMXE = 0** 执行 **VMXON**，将导致无效操作码异常。

一旦进入 **VMX** 操作模式，无法清除 **CR4.VMXE**。**Hypervisor** 通过执行 **VMXOFF** 指令离开 **VMX** 操作模式。在执行 **VMXOFF** 之后，可以在 **VMX** 操作模式之外清除 **CR4.VMXE**。

VMXON /VMXOFF

VMXON还受IA32_FEATURE_CONTROL MSR（MSR地址3AH）的控制。当逻辑处理器被复位时，该MSR被清零。该MSR的相关位为：

• 位 0 是锁定位。如果此位清零，VMXON 将导致通用保护异常。如果设置了锁定位，对该 MSR 的 WRMSR 操作将导致通用保护异常；直到重新上电复位条件，无法修改该 MSR。系统 BIOS 可以使用该位提供用于禁用 VMX 支持的 BIOS 设置选项。为在平台上启用 VMX 支持，BIOS 必须设置位 1、位 2、以及锁定位。

• 位 1 启用 SMX 操作模式（Safer Mode Extensions Reference）中的 VMXON 。如果此位清零，在 SMX 操作模式下执行 VMXON 将导致通用保护异常。在不支持 VMX 操作和 SMX 操作的逻辑处理器上设置此位会导致通用保护异常。

• 位 2 在 SMX 操作模式之外启用 VMXON。如果此位清零，在 SMX 操作之外执行 VMXON 将导致通用保护异常。在不支持 VMX 操作模式的逻辑处理器上设置此位会导致通用保护异常。

VMXON /VMXOFF

在执行 **VMXON** 之前，**Hypervisor** 应分配一个自然对齐的 **4KB** 内存区域，逻辑处理器可以用这个区域来支持 **VMX** 操作。这个区域称为 **VMXON** 区域。**VMXON** 区域的地址（ **VMXON** 指针）在 **VMXON** 指令的操作数中提供。

在执行 **VMXON** 之前，**Hypervisor** 应将 **VMCS** 修订标识符写入 **VMXON** 区域。（具体而言，应将 **31** 位的 **VMCS** 修订标识符 （该值可以在 **IA32_VMX_BASIC MSR** 的低 **31** 位中查到。） 写入 **VMXON** 区域的前 **4** 个字节的位 **30:0**；位 **31** 应清零。）无需以任何其他方式初始化 **VMXON** 区域。**Hypervisor** 应为每个逻辑处理器使用单独的区域，并且在该逻辑处理器上执行 **VMXON** 和 **VMXOFF** 之间不应访问或修改逻辑处理器的 **VMXON** 区域。否则可能导致不可预测的行为

VMXON /VMXOFF

VMXON 指针受到的限制：

1. VMXON 指针必须是 4KB 对齐的（位 11:0 必须为零）。
2. VMXON 指针不能设置超出处理器物理地址宽度的任何位。软件可以通过使用 EAX 中的 80000008H 执行 CPUID 指令来确定处理器的物理地址宽度。物理地址宽度以 EAX 的位 7:0 返回。如果 IA32_VMX_BASIC[48] 读取为 1，则 VMXON 指针不能设置范围在 63:32 之间的任何位

VMXON 区域收到的限制：

1. VMXON 区域所需的内存量与 VMCS 区域相同。该大小是实现特定的，可以通过查阅 VMX 能力 MSR IA32_VMX_BASIC 来确定

VMCS Basic

VMX Non-Root Operation 和 **VMX Transition** 由虚拟机控制结构（**VMCS**）控制。

对 **VMCS** 的访问是通过处理器状态的一个组件来管理的，该组件称为 **VMCS** 指针（每个逻辑处理器一个）。**VMCS** 指针的值是 **VMCS** 的 **64** 位地址。使用 **VMPTRST** 和 **VMPTRLD** 指令可以读取和写入 **VMCS** 指针。**Hypervisor** 使用 **VMREAD** 、 **VMWRITE** 和 **VMCLEAR** 指令配置 **VMCS**

Hypervisor 可以为支持的每个虚拟机使用不同的 **VMCS** 。对于具有多个逻辑处理器（虚拟处理器）的虚拟机，**Hypervisor** 可以为每个虚拟处理器使用不同的 **VMCS** 。

VMCS Basic

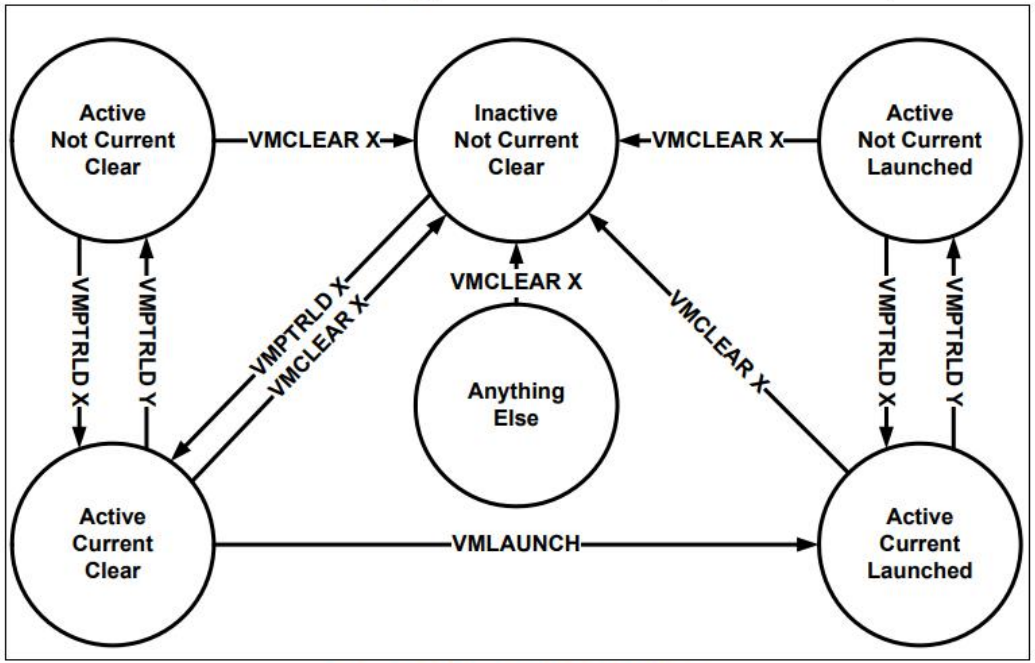


Figure 25-1. States of VMCS X

VMCS Basic

VMCS 区域的前 4 个字节包含位 30:0 的 VMCS 修订标识符。维护 VMCS 数据采用不同格式的处理器使用不同的 VMCS 修订标识符，Hypervisor 可以通过读取 VMX 能力 MSR IA32_VMX_BASIC 来发现处理器使用的 VMCS 修订标识符。。这些标识符使软件能够避免在使用不同格式的处理器上错误处理另一个处理器格式化的 VMCS 区域。这 4 字节区域的第 31 位指示 VMCS 是否为阴影VMCS

类似于 VMXON 区域，Hypervisor 在使用 VMCS 之前应将 VMCS 修订标识符写入 VMCS 区域。处理器不会写入 VMCS 修订标识符；如果 VMPTRLD 的操作数引用的 VMCS 区域的 VMCS 修订标识符与处理器使用的不同，VMPTRLD 操作将失败；如果阴影 VMCS 指示器为1且处理器不支持阴影 VMCS，则 VMPTRLD 操作也会失败。Hypervisor 可以通过读取 VMX 能力 MSR IA32_VMX_PROCBASED_CTLS2 来发现是否支持此设置。

VMCS Basic

VMCS 区域的下一个 4 个字节用于VMX中止指示器。这些位的内容不以任何方式控制处理器的操作。如果发生 VMX 中止，逻辑处理器会将一个非零值写入这些位。Hypervisor 也可以写入此字段。

VMCS 区域的其余部分用于 VMCS 数据（控制 VMX Non-Root Operation 和 VMX Transitions 的部分）。这些数据的格式是特定的。为确保在VMX操作中获得正确的行为，软件应该将VMCS区域和相关结构维护在写回缓存内存中。未来的实现可能允许或要求使用不同的内存类型。软件应该参考 VMX 能力 MSR IA32_VMX_BASIC 进行咨询。

VMCS Basic

Table 25-1. Format of the VMCS Region

Byte Offset	Contents
0	Bits 30:0: VMCS revision identifier Bit 31: shadow-VMCS indicator (see Section 25.10)
4	VMX-abort indicator
8	VMCS data (implementation-specific format)

VMCS Basic

VMCS数据被组织成六个逻辑组：

- **Guest-state area**。处理器状态在 VM 退出时保存到客户状态区域中，并在 VM 进入时从该区域加载。
- **Host-state area**。处理器状态在 VM 退出时从主机状态区域加载。
- **VM-execution control fields**。这些字段控制 VMX Non-Root Operation 中的处理器行为。它们部分确定了 VM 退出的原因。
- **VM-exit control fields**。这些字段控制 VM 退出。
- **VM-entry control fields**。这些字段控制 VM 进入。
- **VM-exit information fields**。这些字段接收有关 VM 退出的信息，并描述 VM 退出的原因和性质。在某些处理器上，这些字段是只读的

Guest-State Area

VMCS guest 状态会在发生 VM entry 时从 VMCS 自动加载进处理器中，并在 VM exit 时自动保存到 VMCS 中。这些状态主要包括：

控制寄存器：CR0、CR3、CR4。

指令指针与栈指针：RIP、RSP、RFLAGS。

完整的段寄存器：即 CS、SS、DS、ES、FS、GS、TR 段的选择子、基址、界限与访问权限。

GDTR 与 IDTR 的基址与界限。

一些 MSR，如 IA32_PAT、IA32_EFER。

Host-State Area

VMCS host 状态会在发生 VM exit 而从 non-root 切换回 root 时，从 VMCS 自动加载进处理器中，但无需在 VM entry 时保存。这些状态主要包括：

控制寄存器：CR0、CR3、CR4。
指令指针与栈指针：RIP、RSP。
段选择子 (selector)：CS、SS、DS、ES、FS、GS、TR。
段基址 (base address)：FS base、GS base、TR base、GDTR base、IDTR base。
一些 MSR，如 IA32_PAT、IA32_EFER。

VM-Execution Control Fields

这些字段用于控制在 non-root 模式运行时处理器的行为。常用的有以下几个：

Pin-based VM-execution controls：用于配置 **hypervisor** 对 **guest** 异步事件的拦截（例如中断）。

Processor-based VM-execution controls：又可分为 **primary processor-based** 和 **secondary processor-based**，用于配置 **hypervisor** 对 **guest** 同步事件的拦截（例如执行特定的指令）。

Exception bitmap：用于配置 **hypervsior** 对 **guest** 异常的拦截。

I/O-bitmap address：用于配置 **hypervisor** 对 **guest** 读写特定 **I/O** 端口的拦截。

MSR-bitmap address：用于配置 **hypervisor** 对 **guest** 读写特定 **MSR** 的拦截。

Extended-page-table pointer：指定扩展页表（**EPT**）的基址。

VM-Exit Control Fields

VM-Exit Control Fields

这些字段用于控制在 **VM exit** 发生时处理器的行为。除了 **VM-exit controls** 外，还有：

VM-exit MSR-store count、**VM-exit MSR-store address**：**VM exit** 时要保存的 **(guest) MSR** 数量与内存区域。

VM-exit MSR-load count、**VM-exit MSR-load address**：**VM exit** 时要载入的 **(host) MSR** 数量与内存区域。

VM-Entry Control Fields

这些字段用于控制在 VM entry 发生时处理器的行为。除了 VM-entry controls 外，还有：

VM-entry MSR-load count、VM-entry MSR-load address：VM entry 时要载入的 (guest) MSR 数量与内存区域。

VM-entry interruption-information field：用于向 guest 注入虚拟中断或异常。

VM-Exit Infomation Fields

这些字段用于控制在 VM entry 发生时处理器的行为。除了 VM-entry controls 外，还有：

Exit reason：VM Exit 的原因

Exit qualification：此字段包含关于部分特殊原因导致 VM Exit 的附加信息

Guest-linear address：此字段用于以下情况：尝试使用内存操作数执行 LMSW 而导致的 VM 退出。尝试执行 INS 或 OUTS 而导致的 VM 退出；在 I/O 指令退役后立即到达的系统管理中断（SMI）导致的 VM 退出；EPT违规引起的某些VM退出。

Guest-physical address：此字段用于由于EPT违规和EPT配置错误导致的VM退出。

VM-Exit Infomation Fields

这些字段用于控制在 VM entry 发生时处理器的行为。除了 VM-entry controls 外，还有：

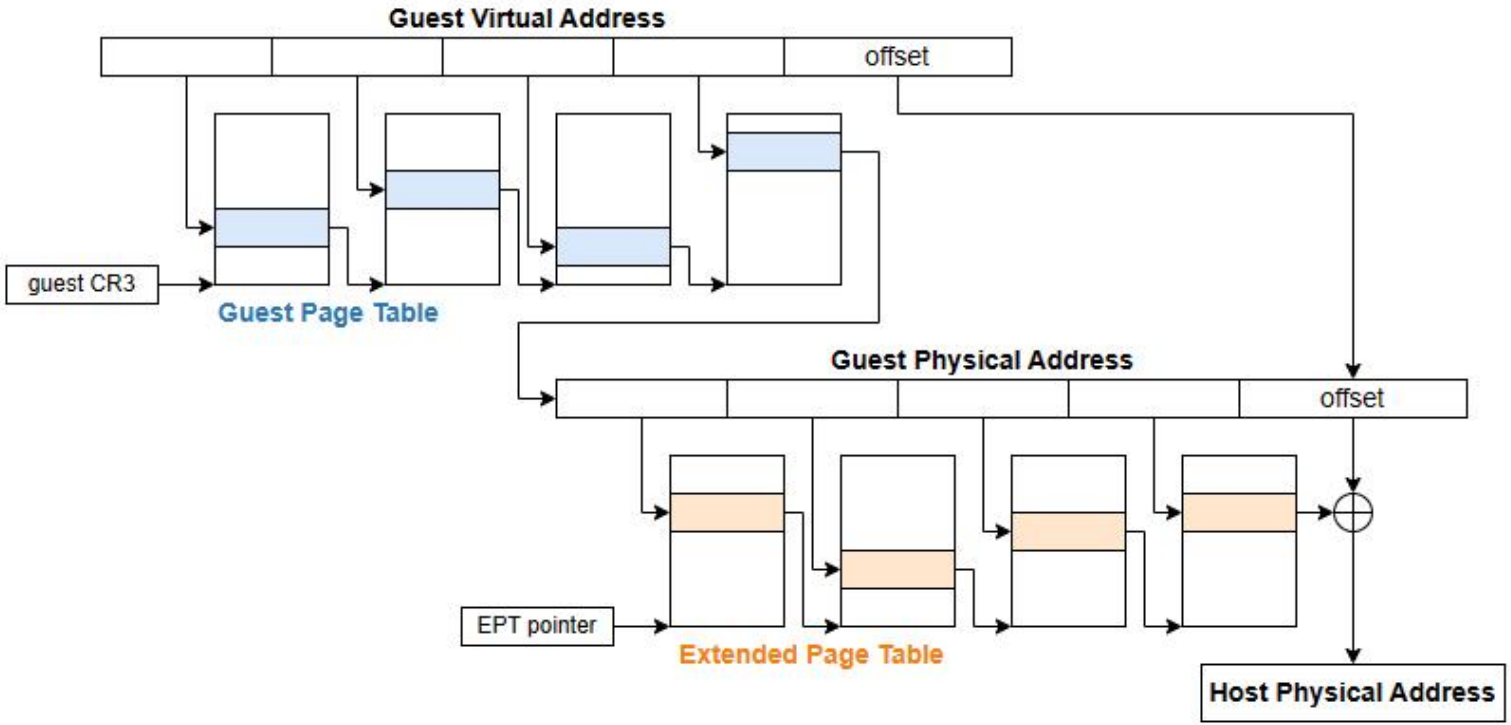
Exit reason：VM Exit 的原因

Exit qualification：此字段包含关于部分特殊原因导致 VM Exit 的附加信息

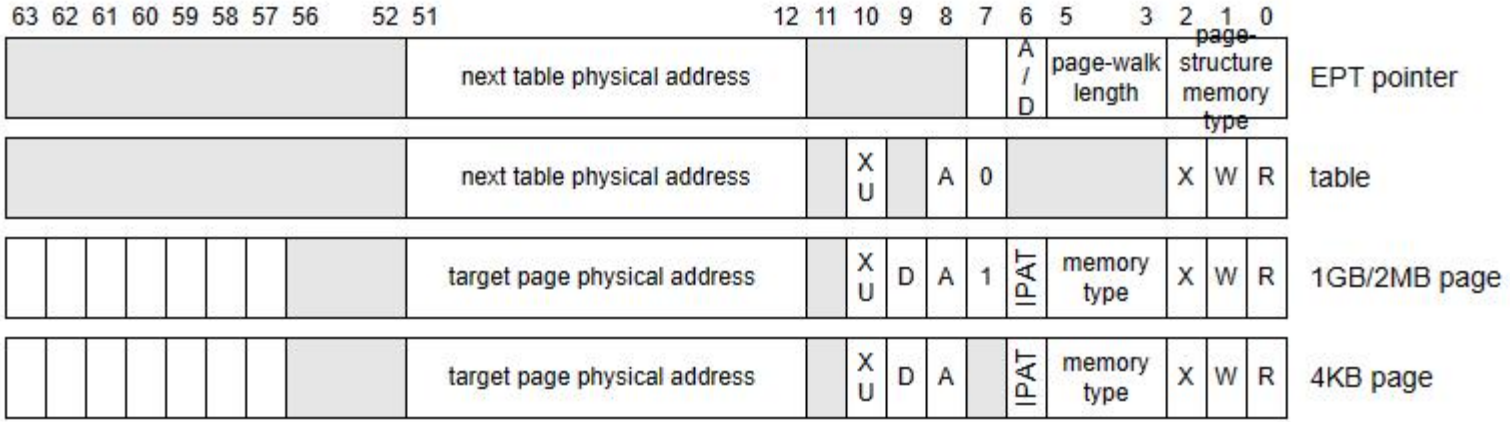
Guest-linear address：此字段用于以下情况：尝试使用内存操作数执行 LMSW 而导致的 VM 退出。尝试执行 INS 或 OUTS 而导致的 VM 退出；在 I/O 指令退役后立即到达的系统管理中断（SMI）导致的 VM 退出；EPT违规引起的某些VM退出。

Guest-physical address：此字段用于由于EPT违规和EPT配置错误导致的VM退出。

EPT


Guest Virtual Address
offset
guest CR3
Guest Page Table
Guest Physical Address
offset
EPT pointer
Extended Page Table
Host Physical Address

EPT



R: Read access
W: Write access
X: Execute access
A: Accessed flag
D: Dirty flag
IPAT: Ignore PAT memory type
XU: Execute access for user-mode linear addresses

Memory type:
0: Uncacheable (UC)
1: Write Combining (WC)
4: Write Through (WT)
5: Write Protected (WP)
6: Write Back (WB)

Ignored or reserved

EPT

当硬件使用 EPT 转换一个 gPA 时，如果中途发生了页面不存在，或是权限不匹配等错误时，会触发一个 VM exit，名为 EPT violation，类似普通页表中的缺页异常 (page fault)。一般情况下，发生 EPT violation，就是 guest 非法访问了一个 guest 物理地址，应该杀掉整个 guest 或报错。但我们也可以利用 EPT violation 实现一些功能。如实现页面交换、按需分配 guest 物理内存、虚拟化对设备的 MMIO 访问等。

VMCS 中有提供了一些与 EPT violation 有关的信息，比如 Exit qualification 会保存访问者的权限信息，用 bit 0/1/2 分别表示是一个 读/写/执行 访问导致了这个 EPT violation (类似缺页异常时的 error code)。此外 Guest-physical address 表示出错的 guest 物理地址，Guest-linear address 表示出错的 guest 虚拟地址等。

Reference

Intel Document: https://cdrdv2.intel.com/v1/dl/getContent/671447

RVM-Tutorial WiKi: https://github.com/equation314/RVM-Tutorial/wiki

ZhiHu: https://www.zhihu.com/column/c_1040263672760885248

WikiPedia: https://en.wikipedia.org/wiki/X86_virtualization#Intel_virtualization_(VT-x)