x86系统架构概览-读书笔记

2021113117 王宇轩

我阅读的主要是Intel 64 and IA-32 Architectures Software Developer's manual Volume 3这本手册。我遇到的第一个问题是，我分不清这些概念：x86、x86-64、x64、IA-32、IA-32e、IA-64……我首先去查了一下这个问题。

> **x86**：一种处理器指令集架构（ISA），最初由Intel公司开发，最早出现在Intel 8086处理器（16位）上，后来发展为在各种Intel处理器上广泛使用的标准架构。
>
> **IA-32**（Intel Architecture 32-bit）：IA-32是Intel的32位扩展处理器架构，也被称为x86-32。它是基于原始 x86 架构的扩展。
>
> **IA-64**（Intel Architecture 64-bit）：Intel公司开发的一种独立的64位处理器架构，最早应用在Intel Itanium（安腾）系列处理器上，指令集与x86系列不兼容（过于激进），市场反响极差，已于2021年完全停产，而x86-64架构成为更为常见的64位架构。
>
> **x86-64**：由AMD公司引入的对x86架构的64位扩展，在安腾的第二天公布，也称**x64**、**AMD 64**，兼容x86架构，并且提供了更大的内存寻址范围和更高的处理能力。现代的个人计算机和服务器多数采用x86-64架构。
>
> **Intel 64**：Intel推出的64位扩展处理器架构，一开始叫EM64T，后改为**IA-32e**（Intel Architecture 32-bit extensions），又称为Intel 64（就是不提AMD），在IA-32的基础上加入了对64位处理的支持。IA-32e架构支持64位寻址和64位数据处理。两种子模式：64位模式（IA-32e模式）和兼容模式。

以及后续的一些疑问，都会补充在这里：

- **段（segment）**：在手册的1.3.5。处理器采用字节寻址；地址空间的概念；处理器支持分段寻址：程序有许多独立空间称为段，如栈段、代码段。
- **线性地址**：逻辑地址到物理地址变换的中间层，由逻辑地址（程序提供、段中的偏移量）和基地址（段选择器）组成。

1. 系统级体系架构概览

1.1 全局/局部描述符表

- 保护模式下，内存访问都要通过描述符表。全局/局部描述符表储存一条条的**段描述符**，包含了段基地址、访问权限、类型和使用情况。每条段描述符都被一个段选择器所指向，段选择器会给使用它的程序提供其指向的段描述符在表中的位置、是在L还是G（一个flag）[1]、以及访问权限信息。
- 段中的字节：段选择器（→段描述符→段基地址）+ 偏移量。还要考虑CPL。
- CPL（Current Privilege Level）：书中这样子措辞：当前运行的代码在某个特权等级下操作。
- 有些图示会简化，段选择器直接指向一个段，但实际总会经过GDT/LDT。
- GDTR/LDTR：储存*DT的基的**线性地址**。
- 在IA-32e的两种子模式下，*DTR都扩展成64位， *DT在只在64位模式下扩展以支持64位基地址：LDT中的描述符是固定长度为16字节的结构，有64位的基地址和其他属性。

Q&A

1. **G/L?**
   GDT在操作系统启动时创建并初始化，在整个系统运行期间保持不变。其中的描述符定义了内核和用户空间的内存段，操作系统通过GDT管理和保护内存，实现不同特权级别的访问控制。LDT是每个任务（进程）独立拥有的描述符表，存储任务私有的段描述符。在任务切换时可以切换LDT。

1.2 系统段、段描述符和门

- 代码段、数据段、栈段组成了程序的运行环境；此架构定义了两个**系统段**：任务状态段（TSS）和LGT，而GDT不算，因为它不是靠段选择器和段描述符来访问的[1]，TSS和LDT都有相应的段选择器/描述符。
- **门**：是架构中定义的一种特殊的**描述符**，为低特权级的程序访问系统程序或handler提供了一种受保护的通道。有调用门、任务门、中断门、陷阱门。

  > 调用门：用于低特权级代码转移到高特权级代码;
  > 任务门：用于不同任务之间的调度;
  > 中断门：用于异步执行中断处理程序;
  > 陷阱门：也用于执行中断处理程序，不过这里的中断是处理器内部产生的;

- 例如对调用门的调用：调用者的代码段特权等级**不高于**被调用者的代码段——调用者向调用门提供选择器，然后，处理器在调用门上执行访问权限检查，将CPL与调用门的特权级别和调用门所指向的目的代码段进行比较。

  > 当用户请求调用门时，操作系统会进行如下特权级检查：
  > 当前特权级CPL（用户程序）和请求特权级RPL，必须**高于或等于**调用门中的DPL；即在数值上CPL <= DPL，RPL <= DPL。（注意：这是调用门描述符里的 DPL）。当前特权级CPL

> （用户程序），必须**低于或等于**目标代码段中的DPL；即在数值上CPL >= 目标代码段描述符中的DPL。

- 若通过检查，处理器就会从调用门中获取目标段的段选择子和偏移量。如果调用想改变权限等级，处理器也会切换到目标级别的堆栈，新堆栈的段选择器是从当前任务的TSS中获取的。
- 门还可以便利16位和32位代码段之间的转换。
- 在IA-32e模式下，以下描述符:LDT描述符，64位TSS，调用门，中断门和陷阱门是16字节描述符（扩展，允许64位基）。调用门促进64位模式和兼容模式之间的转换。IA-32e模式不支持任务门。在特权级别更改时，不会从TSS读取堆栈段选择器，而它们被设置为NULL。

Q&A

1. **GDT的访问方式?**
   TSS、LDT每个任务都配有一个，而GDT是系统的数据结构，整个系统只有一个GDT，通过全局描述符寄存器GDTR进行定位。

1.3 任务状态段TSS和任务门

- TSS定义一个任务执行的环境。包含：通用寄存器、段寄存器、EFLAGS寄存器、指针寄存器EIP，和对于三个栈（每个特权等级一个）的堆栈指针的寄存器...的状态，以及该任务的LDT段选择器和分页层次结构的基地址。
- 保护模式下，所有程序执行都发生在一个任务的上下文（称当前任务）中，当前任务的TSS的段选择器储存在任务寄存器中，就是图2-1中的Task Register。
- 最简便的切换任务方法就是调用或跳转，CALL或JMP指令给出新任务的TSS段选择器。处理器处理任务切换：
  1. 在当前TSS中储存当前任务状态；
  2. 用新任务的段选择器装载任务寄存器；
  3. 通过GDT中的一个段描述符访问新TSS；
  4. 将新任务的状态从新的TSS中加载到通用寄存器、段寄存器、LDTR、控制寄存器CR3（存有分页层次结构的基地址）、EFLAGS寄存器和EIP寄存器；
  5. 开始新任务的执行。
- 也可通过任务门来访问新任务。任务门与调用门相似，但其通过段选择器访问的是一个TSS而不是代码段。
- IA-32e模式不支持硬件任务转换，但TSS依旧存在，而任务寄存器的长度被扩展，可以保存TSS的64位基地址。该模式下TSS的基地址由其描述符进行指定。64位的TSS中存有以下信息： 每一个优先级的堆栈指针地址；中断堆栈表的指针地址；IO权限位图（IO-permission bitmap）的偏移地址（相对于TSS的基地址而言）。

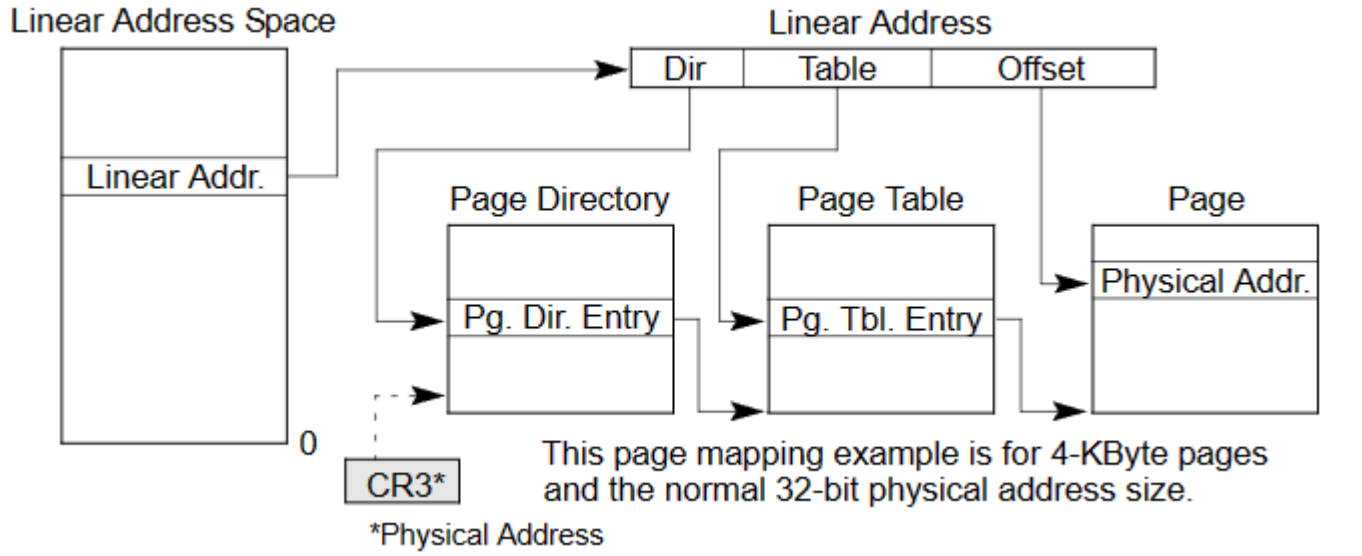
1.4 中断和异常处理

- 外部中断、软件中断和异常通过中断描述符表IDT处理。IDT储存一组**门描述符**，它们提供对中断和异常处理程序的访问，可以是中断、陷阱或任务门的描述符。IDT的线性基地址储存在IDTR。
- 基于与GDT相同的原理，IDT**不是段**。

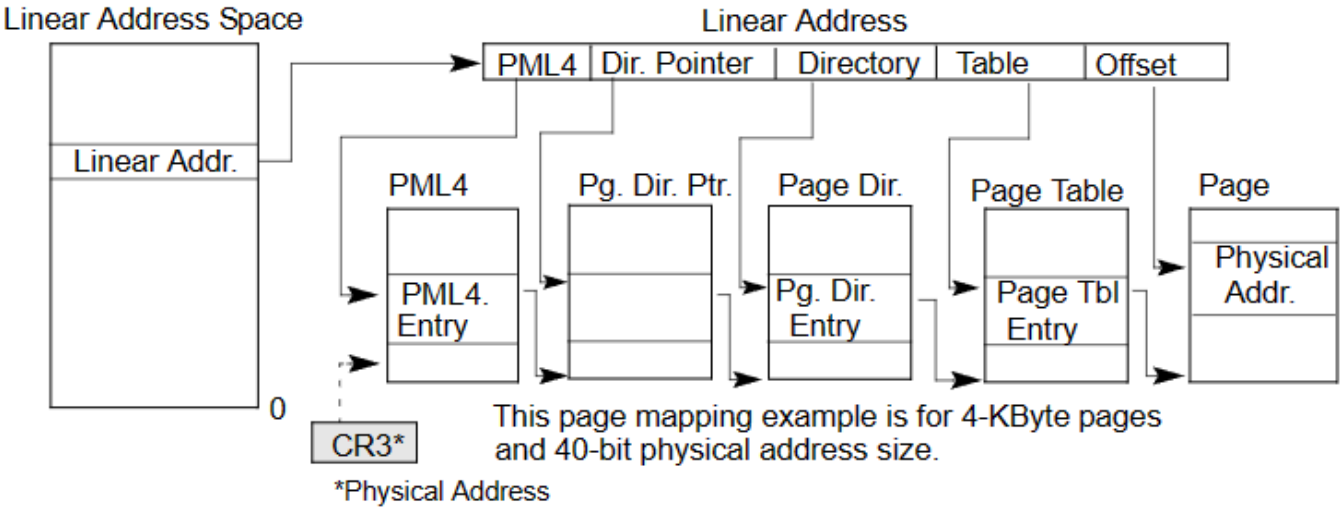
- 访问中断和异常处理程序：处理器首先从内部硬件、外部中断控制器或者通过软件的INT、INTO、INT3或BOUND指令获取中断向量（中断序号），其中提供了一个IDT的索引。然后，如果选中的门描述符是中断门或陷阱门，则通过与调用门调用相似的方式访问处理程序；如果描述符是任务门，则通过任务切换来访问处理程序。
- IA-32e模式下，中断描述符扩展到了16字节，从而可以存储64位的基地址。这种情况适用于64位模式和兼容模式IDTR寄存器也被扩展来保存64位的基地址。任务门不受支持。

1.5 内存管理

- 系统架构支持直接物理寻址内存或虚拟内存（通过**分页**）。使用物理寻址时，线性地址被视为物理地址。使用分页时，可以对所有代码、数据、堆栈和系统段（包括GDT和IDT）进行分页，只有**最近访问的页面保存在物理内存中**。
- 物理内存的中页（**页帧**）的地址存在**分页结构**中，这些**结构存于物理内存**。
- 分页结构层次的物理基地址包含在控制寄存器CR3中。分页结构中的条目决定了页帧的物理基地址、访问权限和内存管理信息。
- 为使用这个机制，把线性地址分为几部分，每部分分别作为分页结构和最终页帧的偏移量。一个系统可以有一个或几个的分页结构层次，例如，每个任务都可以有一个自己的。



- IA-32e模式，物理内存页被一组系统数据结构管理，其中在兼容模式和64位模式下，四层数据结构被采用：**PML4、一组页目录指针表、一些组页目录、一些组页表**。其中的条目包含下一级的物理基地址（对页表来说就是页帧）、访问权限和内存管理信息。PML4的基地址存于CR3。



1.6 系统寄存器

- 为协助处理器初始化和控制操作系统，系统架构在EFLAGS寄存器和几个系统寄存器中提供了几个系统标志位。
  1. EFLAGS中的标志位和IOPL字段控制任务和模式的转换、中断处理、跟踪指令和访问权限。
  2. 控制寄存器（CR0、2、3、4）包含各种用于控制系统级操作的标志位和数据段。这些寄存器中的其他标志位用来表示在操作系统或程序中特定处理器能力是否支持。
  3. 调试寄存器：储存调试程序时的断点。
  4. L、G、IDTR寄存器存储各自表的线性基地址或大小（限制）。
  5. 任务寄存器：当前任务的TSS的线性地址和大小。
  6. 特定于模型的寄存器（MSRs），是一组主要对操作系统或执行过程可用的寄存器（即运行在特权级别为0的代码）。这些寄存器控制诸如调试扩展、性能监视计数器、机器检查体系结构和内存类型范围（MTRRs）等项。其数量和功能随不同处理器而不同。
- 大多数系统都限制应用程序对系统寄存器（除EFLAGS寄存器）的访问。然而，系统可以被设计成所有的程序和过程都运行在最高特权级别（特权级别0）。此时允许应用程序修改系统寄存器。
- 在IA-32e模式下，GDTR、IDTR、LDTR和TR被扩展为足以保存64位基地址的长度，EFLAGS寄存器扩展为64位的RFLAGS寄存器，CR0-CR4同样被扩展为64位。同时，该模式下系统使用CR8进行对任务优先级寄存器（task priority register, TPR）的访问，由此操作系统便可以控制调整外部中断的优先级。在64位模式下，调试寄存器DR0-DR7为64位；在兼容模式下，DR0-DR3中的地址匹配的单位也是64位的。另外，该模式下可以启用扩展功能启用寄存器（extended feature enable register, IA_32EFER），该寄存器根据系统模型的不同将改变。IA_32EFER用于控制IA_32e模式和IA_32e模式操作的激活。

2. 实模式和保护模式转换

2.1 支持的操作模式

- **IA-32**：三种操作模式，一种准操作模式
  1. **保护模式**：处理器的本机操作模式[1]。提供一套丰富的架构特性，灵活性，高性能和向后兼容现有的软件基础。
  2. **实地址模式**：提供了Intel 8086处理器的编程环境，及一些扩展（如切换到保护模式或系统管理模式的能力）。
  3. **系统管理模式（SMM）**：所有自Intel386 SL始的IA-32处理器的架构特性，为操作系统或可执行程序提供透明的机制，来实现电源管理和OEM差异化功能[2]。SMM模式通过一个外部系统中断引脚SMI#来进入，其产生一个系统管理中断SMI。SMM模式下，处理器切换到一个单独的地址空间，同时保存当前运行的程序或任务的上下文，然后可以透明地执行SMM特定的代码。从SMM返回后，处理器回到SMI之前的状态。
  4. **虚拟8086模式**：保护模式下处理器支持的准操作模式，允许处理器在保护的、多任务的环境中执行8086软件。

- **Intel 64**：前四种+**IA-32e模式**：此模式下处理器支持两种子模式：**兼容模式**（允许大多数旧的受保护模式应用程序[3]不加更改地运行）和**64位模式**（提供64位线性地址，支持大于64GB的寻址空间）。

Q&A

1. **“本机操作模式”是什么意思?**
   保护模式是处理器所支持的、较为先进和全面的操作模式，允许操作系统和应用程序充分利用处理器的功能和特性。
2. **什么叫“透明”？什么是“OEM”?**
   OEM（Original Equipment Manufacturer）指的是原始设备制造商，就是制造和销售计算机硬件设备的公司。系统管理模式在后台默默地处理电源管理和OEM（原始设备制造商）差异化功能，而不会对应用程序的正常运行产生干扰或要求应用程序进行特定的适应。这样，操作系统和应用程序的开发者可以专注于自己的任务，而无需担心或处理与电源管理和硬件差异等相关的底层细节。
3. **有关legacy applications：**
   就是那些在过去开发的、基于旧技术或平台的应用程序。

2.2 操作模式转换

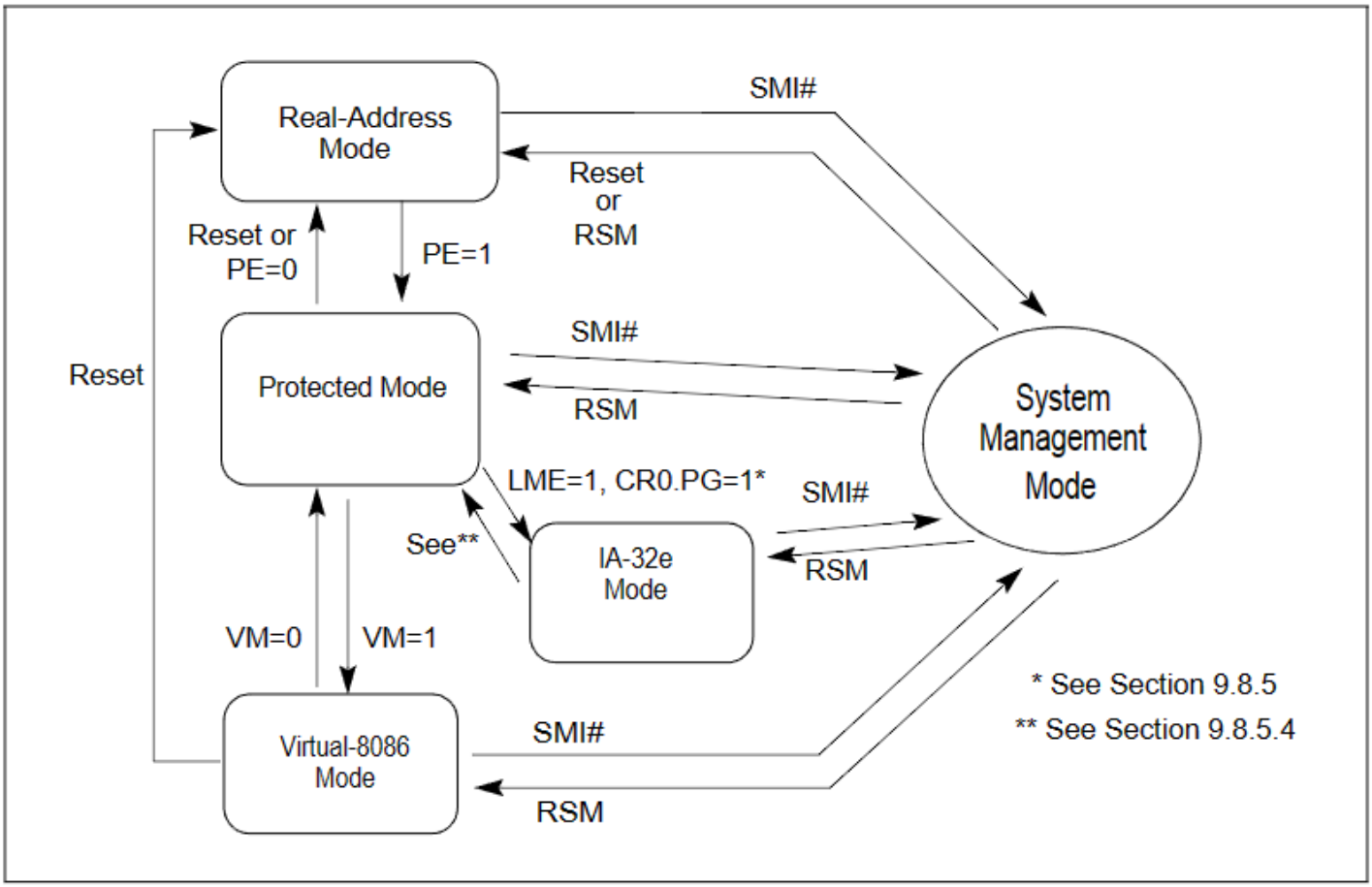


Figure 2-3. Transitions Among the Processor's Operating Modes

- 上电或复位后，处理器被置于**实地址模式**，随后CR0中的**PE标志位**控制处理器运行在实地址模式还是保护模式。

- EFLAGS寄存器中的VM标志确定处理器是工作在保护模式还是虚拟8086模式，这两种模式之间的转换通常是作为任务切换的一部分或作为中断或异常处理程序的返回来执行的。
- IA32_EFER中的LMA位（第十位）确定处理器是否运行在IA-32e模式，若是则由代码段中的CS.L位来确定子模式为64位还是兼容模式。处理器通过启用分页和设置LME位（IA32_EFER第八位）来从保护模式进入IA-32e模式。
- 处理器运行在其他四种模式时，会在收到SMI时切换至SMM。执行SRM指令（Return from system management mode，系统指令之一）时，处理器总会返回至SMI出现前其所在的模式。

2.2.1 切换至保护模式

- 一旦进入保护模式，软件通常不需要返回到实地址模式。要运行以实地址模式（8086模式）编写的软件，通常以**虚拟8086模式**运行比切换回实地址模式更方便。
- 切换至保护模式前，必须将系统数据结构和代码模块的最小集合加载到内存中：一个IDT、GDT、TSS、LDT（可选），至少一个页目录和页表（如果启用分页），一个启用保护模式后要执行代码的代码段，一个或多个包含必要中断和异常处理程序的代码模块。
- 处理器能切换至保护模式前，软件初始化代码还必须初始化下列系统寄存器：GDTR，IDTR（可选，也可以在切换到保护模式后、启用中断之前立即初始化），控制寄存器CR1-4，内存范围寄存器（MTRRs，仅部分型号处理器）。
- 初始化后，通过执行MOV CR0指令在CR0寄存器中设置PE标志（位0）来进入保护模式，该指令中也可以将寄存器CR0中的PG标志设置为启用分页。保护模式下的执行以CPL为0开始。
- Intel 64和IA-32处理器对切换保护模式的要求有细微差异，手册9.9.1列出了一些确保兼容性的建议。

2.2.2 切换回实模式

- 实模式初始化：上电或复位后，处理器处于实地址模式并从物理地址 FFFFFFF0H开始执行初始化代码。软件初始化代码必须首先设置处理基本系统功能所需的数据结构，例如用于处理中断和异常的实模式IDT。如果处理器要保持实模式[1]，软件必须加载额外的操作系统或执行代码模块和数据结构，以便在实数地址模式下可靠地执行应用程序。
- 如果软件用MOV CR0指令清除CR0的PE位，处理器就会回到实模式。基本流程：禁用中断；关闭分页；设置寄存器等符合实模式。

Q&A

1.**有关切换至实地址模式**:
一般不会从保护模式切换到实地址模式；某些特定的操作系统或应用程序可能需要在特定的情况下切换到实地址模式，例如访问特定的硬件设备、执行某些底层操作或运行特定的实模式软件；在虚拟化和调试环境中，可能需要从保护模式切换到实地址模式来模拟或调试早期的操作系统或硬件环境，但用虚拟8086模式更方便。

3. 80x86系统指令寄存器

3.1 标志寄存器EFLAGS

EFLAGS寄存器的系统标志和IOPL字段控制I/O、可屏蔽硬件中断、调试、任务切换和虚拟8086模式。只允许特权代码（通常是操作系统或执行代码）修改这些位。

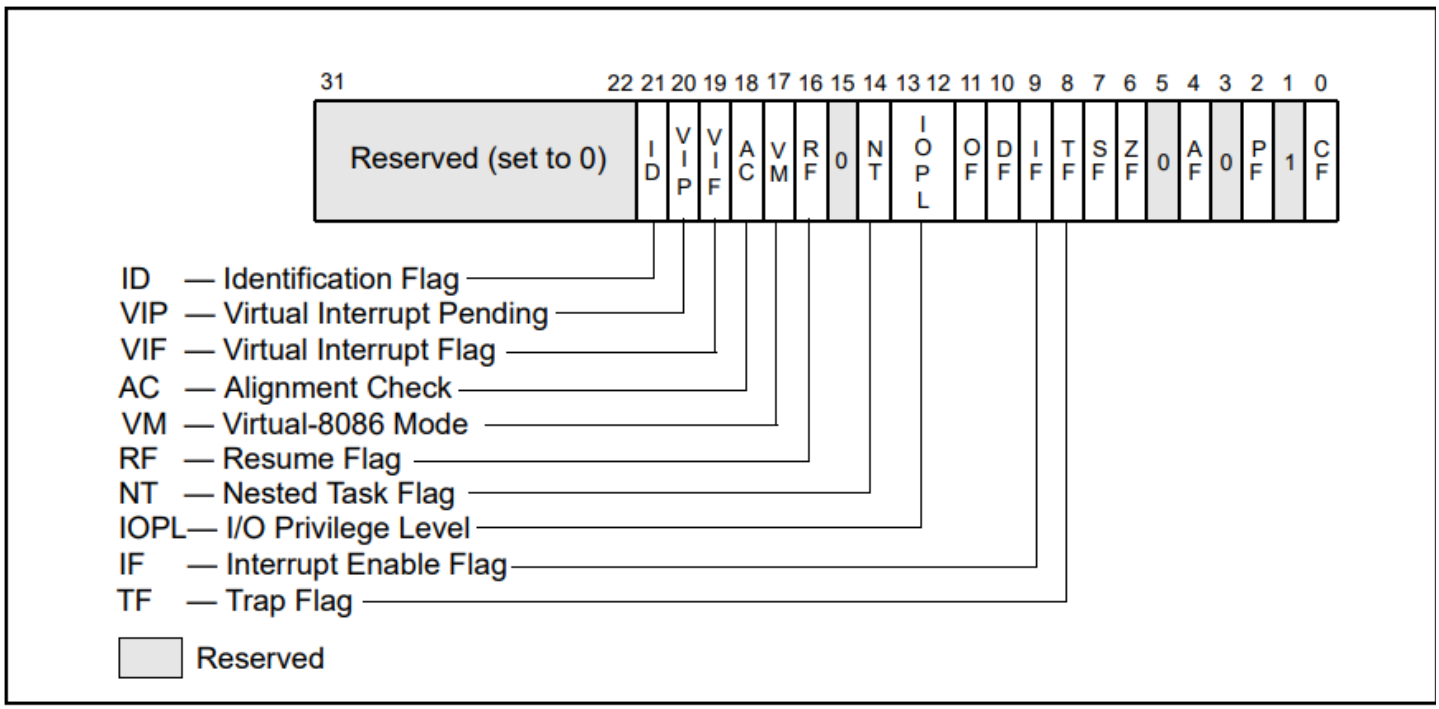


Figure 2-5. System Flags in the EFLAGS Register

- **TF**：置位来启用单步调试模式，清除取消。单步调试模式下，处理器在每一条指令之后都生成一个调试异常来检查程序的执行状态。应用程序使用POPF、POPFD或是IRET指令设置该标志位。
- **IF**：控制处理器对可屏蔽硬件中断请求的响应。置位表示响应可屏蔽硬件中断，清除禁止。IF标志不影响异常和不可屏蔽中断（NMI）的生成。CPL、IOPL和CR4中的VME标志的状态决定了IF标志是否可以被CLI、STI、POPF、POPFD和IRET指令修改。
- **IOPL**：指示当前运行的程序或任务的I/O优先级。正在运行的程序或者任务的CPL必须小于等于IOPL，才能访问I/O地址空间。只有在CPL为0时，POPF和IRET指令才能修改该字段。IOPL也是当虚拟模式扩展有效时（当CR4.VME=1时）控制IF标志的修改和虚拟8086模式中的中断处理的机制之一。
- **NT**：用于控制中断任务和调用任务的链接。处理器在调用由 CALL 指令、中断或异常启动的任务时设置。并在IRET指令启动的任务返回时检查和修改此标志。可以使用POPF/POPFD指令显式地设置或清除该标志，但更改该标志位的状态可能会在应用程序中产生意外异常。
- **RF**：用于控制处理器对指令断点条件的响应。若置位，处理器会暂时禁止为指令断点生成调试异常，清除取消。RF标志的主要用途是允许在由指令断点条件引起的调试异常之后重新启动指令。此外，调试软件必须在使用指令IRETD返回中断程序之前，在堆栈上的EFLAGS映像中设置该标志位，从而以防止指令断点导致另一个调试异常。
- **VM**：用于启用虚拟8086模式，清除后处理器将返回保护模式。

- **AC**：将其和CR0中的AM标志位置位后，可以启用内存引用的对齐检查。清除任意禁用。当引用未对齐的操作数（例如奇数字节地址处的字或地址不是四的整数倍的双字）时，处理器将生成对齐检查异常，该异常用户模式（优先级3）下生成。默认为优先级0的内存引用操作（例如段描述符加载）并不会生成此异常，即使该操作是由在用户模式下执行的指令引起的。对齐检查异常可用于检查数据的对齐情况。这在与需要对齐所有数据的处理器进行交换数据时非常重要。解释器也可以使用对齐检查异常来通过未对齐指针来将某些指针标记为特殊指针。这消除了检查每个指针的开销，并且仅在使用时处理特殊指针。
- **VIF**：包含IF标志的虚拟映像。与VIP标志一起使用。当且仅当CR4中的VME标志或PVI标志被置位，且IOPL小于3时，处理器识别VIF标志。（VME标志启用虚拟8086模式扩展；PVI标志启用受保护模式虚拟中断。）
- **VIP**：由软件设置，指示中断被挂起，清除指示没有中断被挂起。与VIF标志结合使用。处理器读取此标志但从不修改它。当且仅当控制寄存器CR4中的VME标志或PVI标志被置1且IOPL小于3时，处理器识别VIP标志。
- **ID**：程序或过程设置或清除此标志的能力表示支持CPUID指令。

在64位模式下，RFLAGS寄存器扩展到64位，保留高32位。RFLAGS（64位模式）和EFLAGS（兼容模式）的系统标志位如图2-5所示。
在IA-32e模式下，处理器不允许设置VM位，因为不支持virtual-8086模式（尝试设置位会被忽略）。此外，处理器不会设置NT位。然而，处理器确实允许软件设置NT位（注意，如果设置NT位，IRET在IA-32e模式下会导致一般保护故障）。
在IA-32e模式下，SYSCALL/SYSRET指令有一个可编程的方法来指定哪些位在RFLAGS/EFLAGS中被清除。这些指令保存/恢复EFLAGS/RFLAGS。

3.2 内存管理寄存器

四个：**GDTR、LDTR、IDTR、TR**，用于指定控制分段内存管理的数据结构的位置。

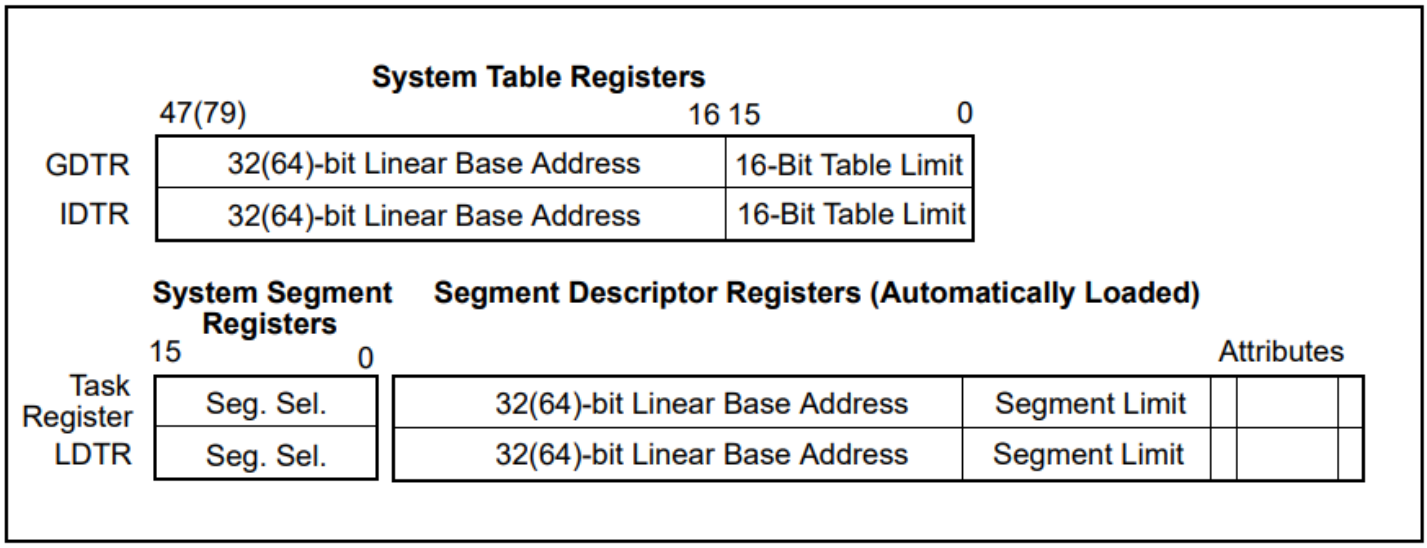


Figure 2-6. Memory Management Registers

- **全局描述符表寄存器GDTR**：储存GDT的**基地址**（保护模式下为32位，IA-32e模式下64位，指定GDT中第0字节的线性地址）和**表界限**（16位，指定表中字节数）。LGDT和SGDT指令分别加载

和存储GDTR。处理器上电或复位时，GDT基地址设置为默认值0，表限制设置为0FFFFH。处理器切换到保护模式时，必须将新的基地址加载到 GDTR 中。

- **局部描述符寄存器LDTR**：储存LDT的**段选择器**（16位）、**基地址**（保护模式下为32位，IA-32e模式下64位，指定LDT中第0字节的线性地址）、**段界限**（指定段中字节数）和**描述符属性**。LLDT和SLDT指令分别加载和存储LDTR寄存器的段选择器部分。包含LDT的段必须在GDT中有一个段描述符。当LLDT指令加载LDTR中的段选择器时：LDT描述符中的基地址、界限和描述符属性会自动加载到LDTR中。当发生任务切换时，LDTR会自动加载新任务的LDT的段选择器和描述符。在将新的LDT信息写入寄存器之前，处理器不会自动保存LDTR的内容。在处理器上电或复位时，段选择器和基地址设置为默认值0，段界限设置为0FFFFH。
- **中断描述符表寄存器IDTR**：储存IDT**基地址**和**表界限**。位数和内容、上电或复位时状态与GDTR同理。LIDT和SIDT指令分别加载和存储IDTR寄存器。
- **任务寄存器TR**：储存当前任务的TSS的**段选择器**（16位，引用GDT中的TSS描述符）、**基地址**、**段界限**和**描述符属性**。LTR和STR指令分别加载和存储任务寄存器的段选择器部分，过程类似LDT。

3.3 控制寄存器

控制寄存器（CR0、CR1、CR2、CR3 和 CR4）用于确定处理器的操作模式和当前执行任务的特性。CR在32位模式和兼容模式下都是32位的。在64位模式下，CR扩展为64位。MOV CRx指令用于操作寄存器位。

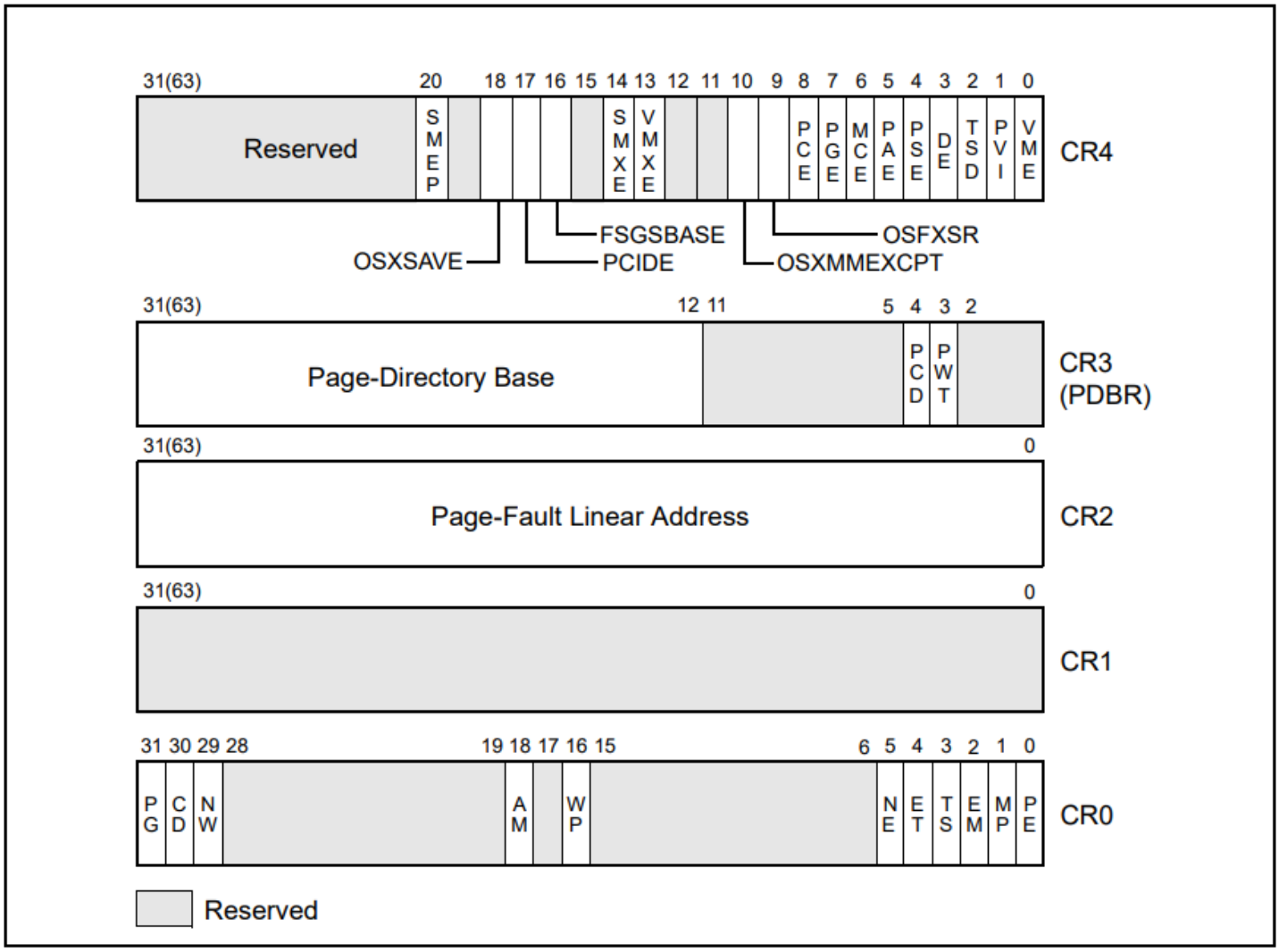


Figure 2-7. Control Registers

- CR0：存储控制处理器操作模式和状态的系统控制标志。CR0的63:32位必须为0，将非零值写入任何高32位都会导致一般保护异常。
- CR1：保留。
- CR2：存储页面错误线性地址（page-fault linear address，导致页面错误的线性地址）。CR2的所有64位均可由软件写入。
- CR3：存储分页结构层次结构的物理基地址，和两个标志位（PCD 和 PWT）。CR3仅指定基地址的最高有效位（减去低12位）；地址的低12位假定为0。因此，第一个分页结构必须与页面（4KB）边界对齐。 PCD和PWT标志控制处理器内部数据缓存中该分页结构的缓存（它们不控制页面目录信息的TLB缓存）。CR3的51:40位必须为0。

4. 系统指令

IA系统设有系统指令（system instructions）来处理系统级功能，例如加载系统寄存器、管理cache、管理中断或者设置调试寄存器等。许多系统指令操作只能由操作系统或者是执行最高优先级0的程序完成。其余的系统指令可以在任意优先级的程序中执行，因此可以在应用程序中使用。

几种重要的系统指令：

- **LGDT**（Load GDTR Register，加载GDTR寄存器）：将GDT线性基地址和表限制从内存加载到GDTR寄存器中。
- **SGDT**（Store GDTR Register，存储GDTR寄存器）：将GDT线性基地址和表限制从GDTR寄存器存储到内存中。
- **LIDT**（Load IDTR Register，加载IDTR寄存器）：将IDT基地址和表限制从内存加载到IDTR寄存器中。
- **SIDT**（Store IDTR Register，存储IDTR寄存器）：将IDT基址和限制从IDTR寄存器存储到内存中。
- **LLDT**（Load LDT Register，加载LDTR寄存器）：将LDT段选择器和段描述符从内存加载到LDTR中。(段选择操作数也可以位于通用寄存器中。)
- **SLDT**（Store LDT Register，存储LDT寄存器）：将LDTR寄存器中的LDT段选择器存储到内存或通用寄存器中。
- **LTR**（Load Task Register，加载TR）：将TSS的段选择器和段描述符从内存加载到任务寄存器中。（段选择器操作数也可以位于通用寄存器中）
- **STR**（Store Task Register，存储TR）：将当前任务TSS的段选择器从任务寄存器存储到内存或通用寄存器中。

以上8种指令都不是**对应用程序有用**（Useful to applications）的，即不是为应用程序提供，以便它们可以使用这些指令或功能来执行特定的操作或获得系统资源。其中，四条Load指令是**对应用程序受到保护**（Protected from Application）的，即应用程序无法直接访问或修改这些指令或功能。