HIT-OS读书笔记（一）：x86系统架构概览

1.系统级体系结构概览

> 系统级架构由一组**寄存器**、**数据结构**和支持基本系统级操作的**指令**，如内存管理、中断和异常处理、任务管理和多处理器控制，下图为系统级寄存器和数据结构的概况。

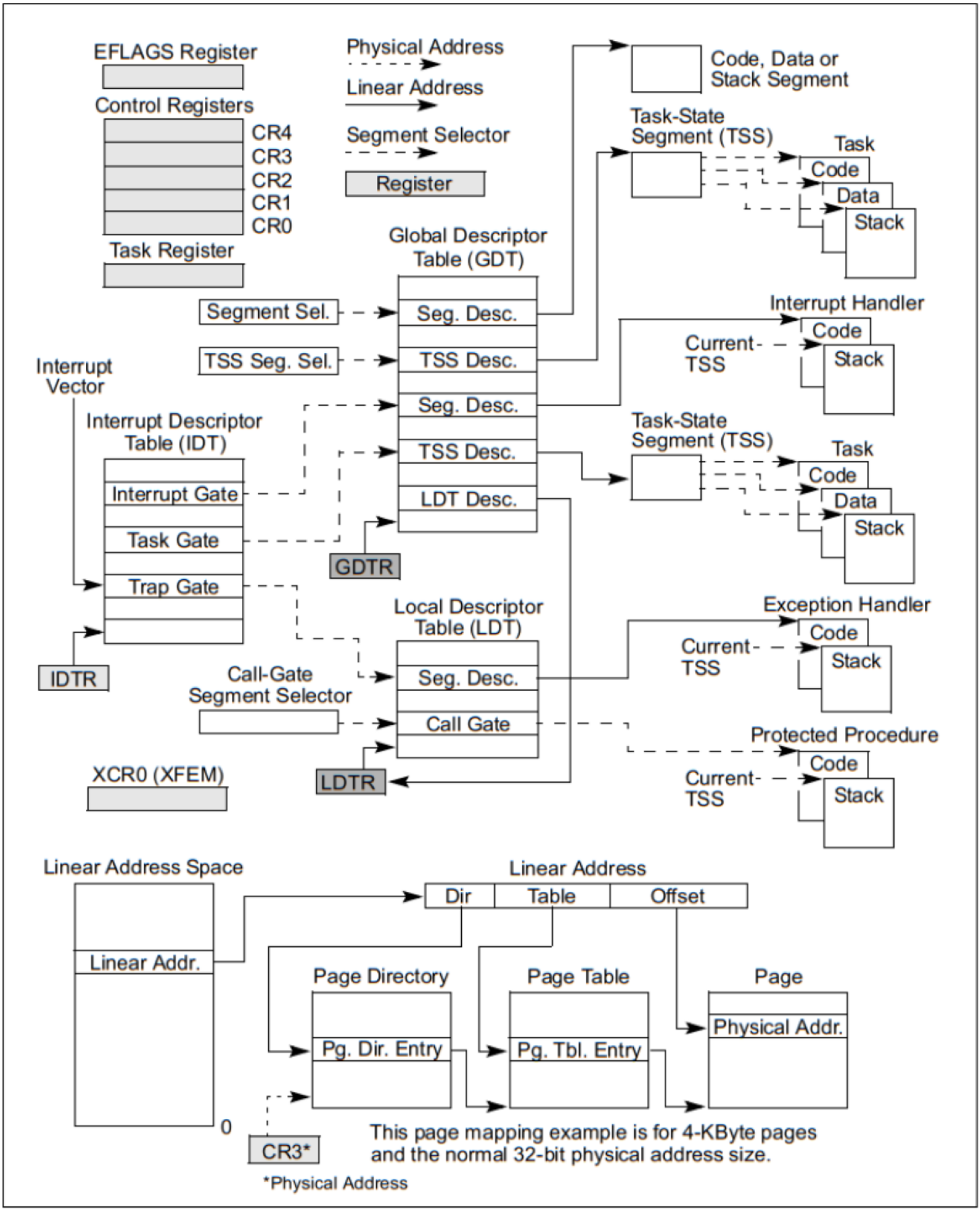


Figure 2-1. IA-32 System-Level Registers and Data Structures

1.1 Global and Local Descriptor Tables 全局与局部描述符表

在保护模式下，所有的内存访问要么通过全局描述符表（GDT）要么通过局部描述符表（LDT）。

- 全局描述符表GDT整个系统**只有一个**，包含操作系统使用的代码段、数据段、堆栈段的描述符以及各任务/程序的局部描述符表段，其基地址的线性地址包含在GDT寄存器（GDTR）中。
- 每个任务/程序有**一个独立的局部描述符LDT**，包含对应任务/程序**私有**的代码段、数据段、堆栈段的描述符以及对应任务/程序适用的门描述符，其线性地址包含在LDT寄存器（LDTR）中。

描述符表里存储的是**段描述符**，段描述符提供了段的基地址、访问特权，类型和用法信息等。并且每个段描述符都有一个相关联的**段选择符**。段选择符包含以下信息：

- GDT或LDT（与它相关的段描述符）里的一个**索引**
- 一个全局/局部**标志**（决定段选择符是指向GDT还是LDT）
- 访问权限等信息。

保护模式下访问段中的内容，必须同时提供**段选择符和偏移地址**。段选择符提供对段的段描述符的访问（在GDT或LDT中）。处理器从段描述符中获得线性地址空间中的段的基地址,而后通过偏移量定位至字节相对于基地址的位置。此机制可用于访问任何有效的代码、数据或堆栈段，前提是可以从处理器运行的当前特权级别（CPL）访问该段。CPL被定义为当前执行的代码段的保护级别。

1.2 System Segments, Segment Descriptors, and Gates

除了代码，数据和堆栈段空间之外，系统架构为程序的运行环境还定义了两个**系统段TSS和LDT**，TTS和LDT有专门为他们定义的**段描述符**。同时系统架构也定义了一系列称为门的**特殊描述符**（包括调用门，中断门、陷阱门和任务门），为安全访问处于不同特权等级上的那些系统过程和处理程序提供了方法。

1.2.1 System Segments 系统段

> *GDT不被视为段，因为它不能通过段选择器和段描述符访问*

系统段是为了实现存储管理机制所使用的一种特别的段，系统架构定义了两个系统段：任务状态段（TSS）和局部描述符表（LDT），TSS和LDT**对应具体的任务**，且均具有为它们定义的段描述符。

1.2.1.1 LDT(局部描述符表)

- 每个任务都配有一个LDT，LDT基地址、界限等信息存放在任务对应的TCB中；
- 通过局部描述符寄存器GDTR进行定位；
- 将LDT视为一种特殊的内存段，则可为每一个LDT创建一个LDT描述符，将描述符存放到GDT中；
- 访问LDT时：GDTR==>访问GDT==>LDT描述符==>访问LDT(==>加载到LDTR)；

1.2.1.2 TSS(任务状态段)

在一个多任务环境中，当发生了任务切换，需保护现场，因此每个任务的应当用一个额外的内存区域保存相关信息，即任务状态段(TSS)

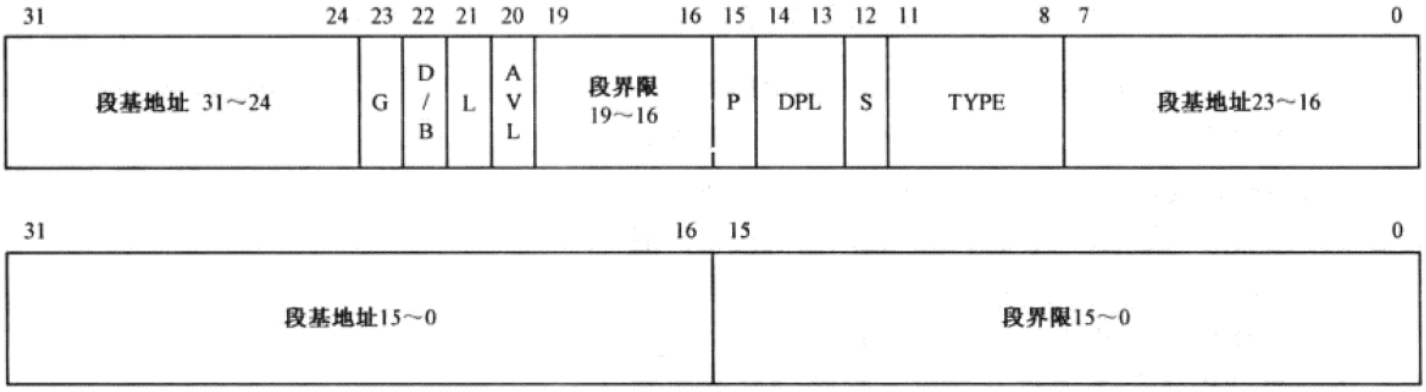
- 每个任务都配有一个TSS，TSS基地址、界限等信息可以存放在任务对应的TCB中；
- 通过任务寄存器TR进行定位；
- 将TSS视为一种特殊的内存段，则可为每一个TSS创建一个TSS描述符，将描述符存放到GDT中；
- 访问TSS时：GDTR==>访问GDT==>TSS描述符==>访问TSS(==>加载到TR)；

1.2.2. Segment Descriptors 段描述符

在x86体系结构中，段描述符（Segment Descriptor）是用于描述内存段（segment）特性的数据结构。每个段描述符包含有关内存段的信息，如基址、段限制、访问权限、段类型等。这些信息用于进行内存访问控制、内存隔离和数据结构的定义。

段描述符是内存管理和访问控制的关键。操作系统通过在GDT和LDT中创建适当的段描述符来管理内存，确保不同进程之间的内存隔离和访问权限。每个进程可以有自己的段描述符，这些描述符在进程切换时被加载到处理器中，以便控制内存访问。

在x86架构中，一个段描述符通常是64位（8字节）长。以下是段描述符的结构，下面是低32位，上面是高32位：



- **段基地址Base**：段的基址（Base Address）字段，指定了段的起始地址。
- **段界限Limit**：段限制（Segment Limit）字段，指定了段的大小，与G有关联。
- **G**：粒度位，决定段界限单位及扩展范围
  - 0 段界限以字节为单位，段的扩展范围1B~1MB
  - 1 段界限以4KB为单位，段的扩展范围4KB~4GB
- **AVL**：Available（AVL）字段，一般用于操作系统或应用程序自定义的目的。
- **P**：Present（P）字段，表示段是否存在于内存中。
- **DPL**：特权级别（Descriptor Privilege Level）字段，指定了访问该段的特权级别,取值范围是0（最高特权）到3（最低特权）

  > *当一个程序或任务尝试访问一个段时，CPU会检查段描述符中的DPL和当前执行的代码所在的特权级别（CPL）。只有当CPL小于或等于段描述符中的DPL时，访问才会被允许。*

- **S**：S字段，表示描述符的类型。
  - 1表示段描述符
  - 0表示系统描述符。
- **TYPE**：段类型字段，指定了段的属性和用途，如代码段、数据段等。
  - **代码段（Code Segment）**： 用于存放执行代码的内存段。
  - **数据段（Data Segment）**： 用于存放数据的内存段。
  - **堆栈段（Stack Segment）**： 用于存放堆栈数据的内存段。
  - **系统段（System Segment）**： 包括任务状态段、TSS段等。

1.2.3 Gates 门

门（调用、中断、任务或陷阱）用于**跨段转移执行控制**，权限级别检查的完成方式取决于所使用的目标类型和指令。

类型	区别
调用门	调用门使用 `CALL/JMP` 指令。调用门将控制从较低权限代码转移到与当前特权级相同或较高权限的代码段， 门DPL用于确定哪些权限级别可以访问门。调用门正在(或可能已经)逐渐被放弃，转而支持更快的 `SYSENTER/SYSEXIT` 机制。
任务门	任务门用于硬件多任务支持。硬件任务切换可以自动发生(`CALL/JMP` 到任务门描述符)，或者在设置NT标志时通过中断或 `IRET` 。 它的工作方式与中断门或陷阱门相同。没有使用任务门是因为内核通常希望在任务切换时完成额外的工作。
中断描述符	中断、陷阱、任务门一起被称为中断描述符表（IDT），存储在IDT中。除了参数从一个特权堆栈到另一个特权堆栈的传输之外， 它们的工作方式与调用门相同。一个区别是中断门会清除EFLAGS中的IF位，而陷阱门则不会， 这使它们成为处理硬件中断的理想选择。陷阱门广泛用于硬件辅助虚拟化。

门描述符与段描述符

类型	区别
段描述符	用于描述符内存段，比如：数据段、代码段、堆栈段，存储段基址
门描述符	用于描述可执行代码，比如：一段程序、一个过程（例程、子程序）或者一个任务，存储段选择子

1.3 Task-State Segments and Task Gates 任务状态段和任务门

1.3.1 任务状态段TTS

TSS定义了任务执行环境的**状态**，它包括通用寄存器、段寄存器、EFLAGS寄存器、EIP寄存器和段选择符以及三个不同特权级别堆栈段的堆栈指针的状态。同时TSS也包括与任务相关联的LDT的段选择符和页表的基址。

保护模式下的所有程序执行都发生在任务(称为当前任务)的上下文中。当前任务的TSS的段选择器存储在任务寄存器TR中。切换到任务的最简单方法是通过 `CALL` 或 `JMP` 指令调用或跳转到新任务，新任务的TSS的段选择符由指令中给出

在切换任务时，处理器执行以下动作:

> 1. 将当前任务的状态存储在当前TSS中。
> 2. 用新任务的段选择器加载任务寄存器。
> 3. 通过GDT中的段描述符访问新的TSS。
> 4. 将新的TSS中新任务的状态加载到通用寄存器、段寄存器、LDTR、控制寄存器CR3(页表基址)、EFLAGS寄存器和EIP注册。
> 5. 开始执行新任务。任务也可以通过任务门访问。任务门类似于调用门，但它提供(通过段选择符)对TSS而不是代码段的访问。

1.3.2 任务门Task Gates

任务门允许操作系统在进行任务切换时，以一种更加灵活的方式将**控制权**从一个任务（或进程）**转移**到另一个任务。其包含一个16位的选择子（Selector），该选择子指向全局描述符表（GDT）或本地描述符表（LDT）中的一个描述符。描述符中存储了目标任务（或进程）的位置和特权级等信息。

当处理器执行任务门时，它会从描述符中获取目标任务的位置（代码段选择子和偏移量），然后跳转到目标任务的代码执行点。这种方式允许任务切换时的平滑转移，而无需手动保存和恢复所有寄存器的状态。

任务门的使用有助于实现**多任务**系统，允许操作系统在不同任务之间进行切换，以实现**并发和多线程**的目标。

1.4 Interrupt and Exception Handling 中断和异常处理

中断和异常是指示系统、处理器或当前执行的程序或任务中存在需要处理器注意的事件。它们通常导致将执行从当前运行的程序或任务强制转移到中断/异常处理程序，而处理器为响应中断或异常而采取的操作称为中断和异常服务/处理。

1.4.1 中断Interrupts

中断在程序执行过程中随机发生，以响应来自硬件的信号。系统硬件使用中断来处理处理器外部的事件，例如为外围设备提供服务的请求。软件也可以通过执行 `INT n` 指令来产生中断。处理器从两个来源接收中断:

- **外部(硬件生成)中断**：外部中断通过处理器上的引脚或通过本地APIC接收。
- **软件生成的中断**： `INT n` 指令允许通过提供中断向量号作为操作数从软件内部生成中断。例如， `INT 35` 指令强制对 `INT 35` 的中断处理程序进行隐式调用。

1.4.2 异常Exception

当处理器在执行指令(如除零)时检测到错误情况时，就会发生异常。处理器检测各种错误情况，包括保护违反、页面错误和机器内部错误。在不失进程执行连续性的同时，按引起异常的指令能否重新被执行和其报告方式，异常可分为**错误**、**陷阱**和**终止**三种情况。

Pentium 4、Intel Xeon、P6系列和Pentium处理器的机器检查架构也允许在检测到内部硬件错误和总线错误时生成机器检查异常。处理器从三个来源接收异常:

- **处理器检测到的程序错误异常**：处理器在应用程序、操作系统或执行程序的执行过程中检测到程序错误时，会产生一个或多个异常。Intel 64和IA-32体系结构为每个处理器可检测的异常定义了一个向量。异常分为故障、陷阱和中止。
- **软件生成异常**： `INTO` 、 `INT 3` 和 `BOUND` 指令允许在软件中生成异常。这些指令允许在指令流中的点执行异常条件检查。例如， `INT 3` 会导致生成断点异常。
- **机器检查异常**：P6系列和Pentium处理器提供内部和外部机器检查机制，用于检查内部芯片硬件和总线事务的操作。这些机制相互依赖，组成了扩展异常机制。当检测到机器检查错误时，处理器发出机器检查异常信号(向量18)并返回错误代码。

1.4.3 中断/异常处理

外部中断、软件中断和异常是通过中断描述符表(IDT)处理的。IDT为每一个异常或中断向量对应的例程或任务分配了一个**门描述符**，这些门描述符提供对中断和异常处理程序的访问。IDT基址的线性地址包含在IDT寄存器(IDTR)中。IDT中的门描述符可以是**中断、陷阱或任务门**描述符。

> *和GDT一样，IDT也不被视为是一个段*

处理器对异常和中断调用的处理方式与用 `CALL` 指令调用例程和任务的处理十分相近。访问中断或异常处理程序时，处理器首先从内部硬件、外部中断控制器或通过 `INT` 、 `INTO` 、 `INT 3` 或 `BOUND` 指令从软件接收一个中断向量(中断号)，中断向量提供IDT的索引。

- 如果选择的门描述符是中断门或陷阱门，则以类似于通过调用门调用过程的方式访问相关的处理程序过程。
- 如果描述符是任务门，则通过任务开关访问处理程序。

当处理器**执行异常处理程序或中断处理程序的调用**时:

- 如果处理程序过程将在数字上较低的特权级别上执行，则会发生堆栈切换。发生堆叠切换时:
  1. 处理程序要使用的堆栈的段选择器和堆栈指针是从当前执行任务的TSS中获得的。在这个新的堆栈上，处理器推送被中断过程的堆栈段选择器和堆栈指针。
  2. 处理器然后保存EFLAGS, CS和EIP寄存器在新堆栈上的当前状态。
  3. 如果异常使得错误码被保存，则将错误码在EIP值之后压入新堆栈。
  4. 使用IRET指令在返回时切换回被中断过程的堆栈。
- 如果处理程序过程将以与被中断过程相同的特权级别执行
  1. 处理器保存当前堆栈中EFLAGS、CS和EIP寄存器的当前状态。
  2. 如果异常使得错误码被保存，则将错误码在EIP值之后压入当前堆栈。
  3. 使用IRET指令从异常处理程序或中断处理程序中返回

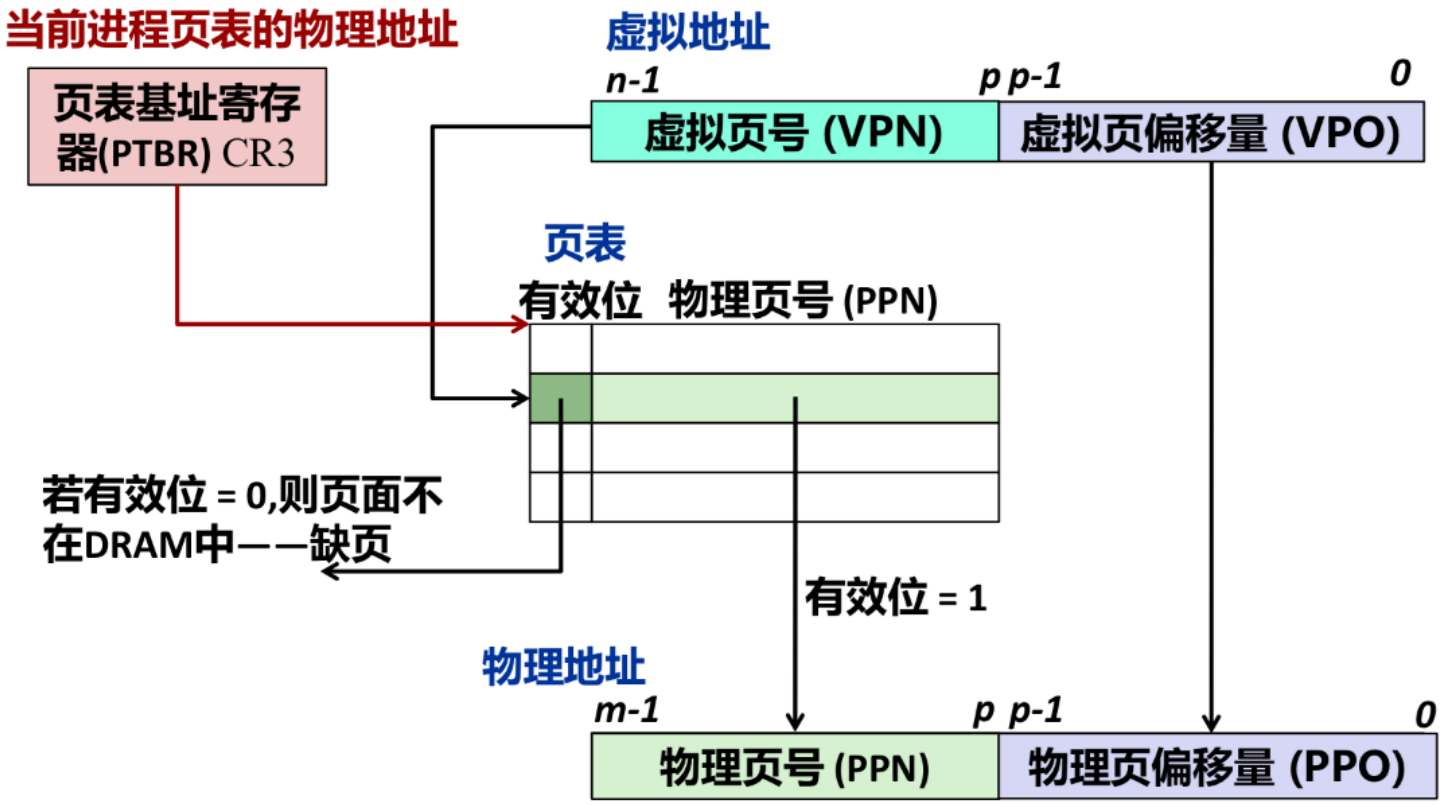
1.5 Memory Management

系统架构支持**直接物理寻址内存**或**虚拟内存(通过分页)**。系统结构支持直接物理地址或虚拟内存。

- 使用直接物理地址时，线性地址就被看做是物理地址；
- 而通过页表使用虚拟内存时。所有的代码、数据、栈信息等都可以将最近访问的页留在内存中，进行分页。页表条目包含也标的物理地址、访问权限和内存管理信息，而页表包含页框的上述信息。

页(有时称为页帧)在物理内存中的位置包含在分页结构中。这些结构驻留在物理内存中。

分页结构层次结构的基本物理地址包含在控制寄存器CR3中。分页结构中的条目确定页框架基的物理地址、访问权限和内存管理信息。其地址结构由两部构成，前一部分是虚拟页号(VPN)，后一部分为虚拟页偏移量(VPO)。如图所示，页号用于在页表中查找对应的页表项，页表项中包含了该虚拟页所映射的物理页号以及访问权限等信息，通过将物理页号和页内偏移量组合得到最终的物理地址。



> *Inter Core i7地址翻译采用TLB和四级页表的结合使用的方式，能够提高地址翻译的效率，减少对内存访问的次数，从而提升系统性能。*

这种分页机制将程序的逻辑地址空间和物理内存划分为等长的页面，这种分配方式便于维护，且不容易产生碎块，同时一个系统可以有一个或多个分页结构层次结构。例如，每个任务可以有自己的层次结构。

1.6 System Registers 系统寄存器

为了协助初始化处理器和控制系统操作，系统架构提供了EFLAGS寄存器和多个系统寄存器中的系统标志：

- **EFLAGS寄存器**中的系统标志和IOPL字段控制任务和模式切换、中断处理、指令跟踪和访问权限。
- **控制寄存器**（CR0、CR2、CR3和CR4）包含用于控制系统级操作的各种标志和数据字段。这些寄存器中的其他标志用于指示操作系统或执行体中特定处理器功能的支持。
- **调试寄存器**允许设置断点，用于调试程序和系统软件。
- **GDTR**、**LDTR**和**IDTR寄存器**包含其各自表的线性地址和大小（限制）。
- **任务寄存器TR**包含当前任务的TSS的线性地址和大小。
- **模型特定寄存器**（MSR）是一组主要供操作系统或执行体过程（即以特权级0运行的代码）使用的寄存器。这些寄存器控制诸如调试扩展、性能监控计数器、机器检查体系结构和内存类型范围（MTRR）之类的项目。

这些寄存器的数量和功能因Intel 64和IA-32处理器系列的不同产品而异。

2. 实模式和保护模式转换

最早期的8086 CPU只有一种实模式工作方式，而80386以及现在的奔腾，酷睿等等CPU为了向前兼容都保留了实模式。

> *现代操作系统在在加电冷启动，或者重置reset后，处理器处于实地址操作模式。CR0寄存器的PE标志位可以用于控制处理器在实模式与保护模式之间的切换。*

- **实地址模式**——这种操作模式提供了Intel 8086处理器的编程环境，并提供了一些扩展(例如切换到保护模式或系统管理模式的能力)。
  - 16位操作：使用16位寄存器和20位地址总线，寻址范围为0x00000-0xFFFFF（1MB）。
  - 单任务操作：只能运行一个程序，无法进行多任务处理。
  - 无保护和特权级限制：没有内存保护和特权级限制，任何程序可以直接访问系统内存和设备。

- 段地址转换：采用分段机制，段基地址由16位段寄存器值左移4位表达。

> *8086CPU将1MB存储空间分成许多逻辑段，每个段最大限长为64KB*

- **保护模式**——这是处理器的本地操作模式。它提供了一套丰富的架构特性、灵活性、高性能和对现有软件库的向后兼容性。
  - 32位操作：使用32位寄存器和32位地址总线，寻址范围为0x00000000-0xFFFFFFFF（4GB）。
  - 多任务操作：支持多任务处理，可以同时运行多个程序，每个程序有自己的内存空间。
  - 内存保护和特权级：提供内存保护机制，可以设置不同的特权级别（0-3级），限制程序对内存和设备的访问。
  - 分段和分页机制：采用分段和分页机制，使内存管理更灵活和高效。

> *保护模式下，段寄存器不再存储段基址，而是存储段选择子，不再需要段寄存器左移加偏移。真正的段基址存在描述符高速缓存中。*

而从**实模式切换到保护模式**大致可以分为以下几个步骤：

1. **屏蔽中断:** 在16位实模式下的中断由BIOS处理，进入保护模式后，中断将交给中断描述符表IDT里规定的函数处理，在刚进入保护模式时IDTR寄存器的初始值为0，一旦发生中断（例如BIOS的时钟中断）就将导致CPU发生异常，所以需要首先屏蔽中断。
2. **初始化全局描述符表（GDT）：**使用 `LGDT` 指令将GDT的地址和大小加载到GDTR寄存器中。
3. **设置CR0寄存器:** CR0是系统内的32位控制寄存器之一，可以控制CPU的一些重要特性。其中最低位是保护允许位（Protected Mode Enable, PE），使用 `MOV` 指令将**PE位置1**后CPU进入保护模式。
4. **执行跳转：** 执行 `jmp 08h:PModeMain` 指令跳转到保护模式下的初始化代码，同时清空CPU的流水线（在实地址模式下按16位处理进入流水线的指令）
5. **初始化段寄存器和栈指针:** 初始化其他的段寄存器如数据段寄存器ds，拓展段寄存器es，栈段ss以及fs，gs两个由操作系统使用的段。

而保护模式切换回实地址模式的过程与此类似，主要涉及清除控制寄存器CR0中的保护模式位，然后加载适当的实模式代码段描述符，并跳转到实模式代码执行。

3. 80x86系统指令寄存器

3.1 EFLAGES标志寄存器

EFLAGES寄存器中的系统标志和IOPL域用于控制I/O、可屏蔽硬件中断、调试、任务切换和虚拟8086模式。只有特权级代码(通常是操作系统或执行代码)可以修改这些位。

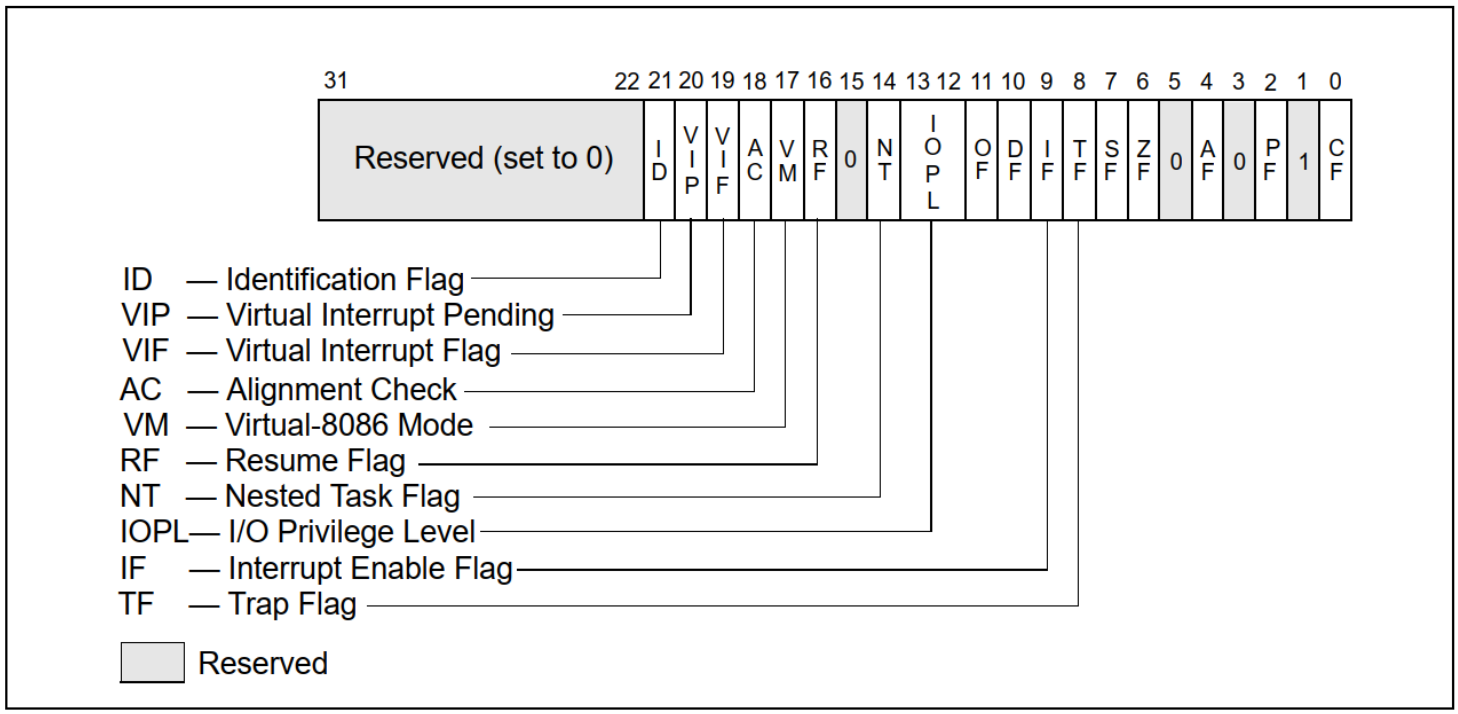


Figure 2-5. System Flags in the EFLAGS Register

其中的**系统标志和IOPL**包括：

标志	意义	作用
TF	**陷阱**标志	置1是调试状态下的**单步执行**，置0禁用单步执行， 单步执行模式下处理器在每条指令后能够产生一个调试异常便于查看执行程序状态， 可以使用 `POPF` 、 `POPFD` 或 `IRET` 指令修改TF标志
IF	**中断允许**标志	控制处理器对**可屏蔽**硬件中断请求的响应，置1是响应可屏蔽硬件中断，置0为禁止响应可屏蔽硬件中断， 不影响异常和不可屏蔽中断（NMI）的产生。 控制寄存器CR4中的CPL,IOPL和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	**I/O特权**域	指出当前程序/任务的**I/O特权级**，当前程序/任务的CPL必须小于或等于IOPL才可以访问I/O地址空间。 只有当CPL为0时，该字段才能被 `POPF` 和 `IRET` 指令修改。
NT	**嵌套任务**标志	控制**中断和调用**任务的链接。处理器在调用由 `CALL` 指令、中断或异常发起的任务时设置此标志。 它在由 `IRET` 指令启动的任务返回时检查并修改此标志。该标志可以通过 `POPF/POPFD` 指令显式设置或清除
RF	**恢复**标志	控制处理器对**断点指令条件**的响应，其主要功能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 (生效后自动清除)。置1将暂时禁用为指令断点生成的调试异常(尽管其他异常条件可能导致生成异常)， 置0指令断点将生成调试异常。
VM	**虚拟8086**模式	置1进入虚拟8086模式，置0返回保护模式
AC	**对齐检查**标志	将此标志和CR0中AM标志置1启用内存引用的对齐检查，置0禁用对齐检查。
VIF	**虚拟中断**标志	与VIP标志一起使用，包含IF标志的虚拟映像，处理器只有在CR4的VME或PVI为1且IOPL小于3时才能识别VIF标志。
VIP	**虚拟中断等待（*pending*）**	**由软件设置**，置1表明中断正在挂起;置0表示没有挂起的中断。此标志与VIF标志一起使用。处理器读取这个标志， 但不修改它。
ID	**识别**位	置1或0表明是否支持 `CPUID` 指令

> 当虚拟模式扩展生效时(当CR4的VME为1时)，IOPL也是控制IF标志修改和在虚拟8086模式下处理中断的机制之一。
> 当引用一个未对齐的操作数时，会产生对齐检查异常，例如奇数字节地址上的字，或者地址上的双字不是4的整数倍。对齐检查异常仅在用户模式下生成(特权级别3)。默认为特权级别0的内存引用，例如段描述符加载，即使是由在用户模式下执行的指令引起的，也不会生成此异常。

3.2 内存管理寄存器

处理器提供了四个内存管理寄存器(GDTR、LDTR、IDTR和TR），指定控制分段内存管理的数据结构的位置，而对于加载和存储这些寄存器则有相应的专有指令。

> ***相似之处：*** *四种内存管理寄存器保存基址均为保护模式下32位，IA-32e模式下64位。*
> ***不同之处：*** *GDTR和IDTR较为相似，但处理器初始化时寄存器中的基址是否必须改变需要区分；LDTR与TR较为相似，均对应与具体的任务/进程，且不会自动保存*

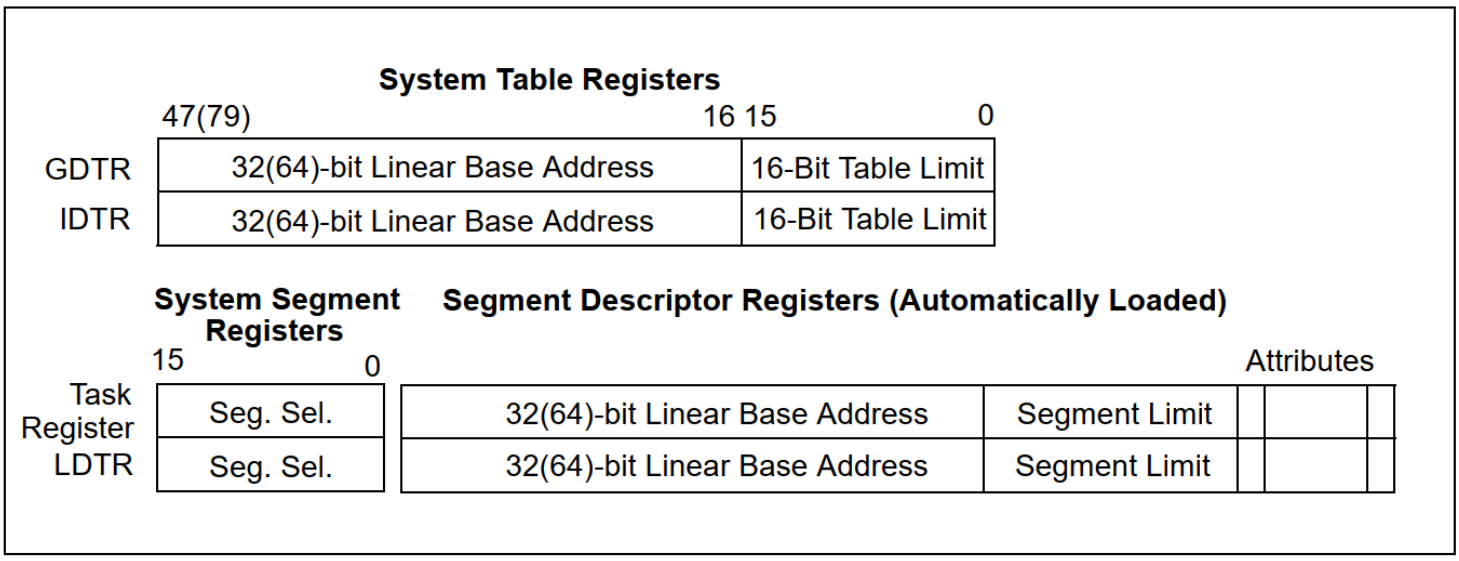


Figure 2-6. Memory Management Registers

3.2.1 全局描述符表寄存器GDTR:

GDTR保存基址(保护模式下32位；IA-32e模式下64位)和16位GDT表界限。

- 基址为GDT的0字节的线性地址，表界限为表中的字节数。
- `LGDT` 和 `SGDT` 指令分别加载和存储GDTR寄存器。
- 在处理器上电或复位时，基址被设置为默认值0，限制被设置为0FFFFH。对于保护模式，作为处理器初始化过程的一部分，**必须**将新的基址加载到GDTR中。

3.2.2 局部描述符表寄存器LDTR:

LDTR寄存器保存16位段选择器，基址(保护模式下32位；IA-32e模式下64位)，段界限和LDT描述符属性。

- 基址指定LDT段第0字节的线性地址，段界限指定了段中的字节数。
- `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。当使用 `LLDT` 指令在LDTR中加载段选择器时，LDT描述符中的基址、限制和描述符属性会自动加载到LDTR中。
- 当任务切换发生时，LDTR将自动加载用于新任务的LDT的段选择器和描述符。在将新的LDT信息写入寄存器之前，LDTR的内容不会自动保存。
- 在上电或处理器复位时，段选择器和基址被设置为默认值0，上限为0FFFFH。

3.2.3 中断描述符表寄存器IDTR:

IDTR寄存器保存基址(保护模式下32位；IA-32e模式下64位)和16位的IDT表界限。

- 基址为IDT的0字节的线性地址,表界限为表中的字节数。
- `LIDT` 和 `SIDT` 指令分别加载和存储IDTR寄存器。
- 在处理器上电或复位时，基址被设置为默认值0，限制被设置为0FFFFH。然后，作为处理器初始化过程的一部分，**可以**改变寄存器中的基址和限制。

3.2.4 任务寄存器TR:

任务寄存器保存16位段选择器，基址(保护模式下32位；IA-32e模式下64位)，段界限和当前任务TSS描述符属性。

- 选择器引用GDT中的TSS描述符。基址指定TSS的第0字节的线性地址，段界限指定TSS的字节数。
- `LTR` 和 `STR` 指令分别加载和存储任务寄存器的段选择器部分。当LTR指令在任务寄存器中加载段选择器时，来自TSS描述符的基址、限制和描述符属性会自动加载到任务寄存器中。
- 当任务切换发生时，任务寄存器将自动加载新任务的TSS的段选择器和描述符。在将新的TSS信息写入寄存器之前，任务寄存器的内容不会自动保存。
- 在处理器上电或复位时，基址被设置为默认值0，限制被设置为0FFFFH。

3.3 控制寄存器

控制寄存器（CR0、CR1、CR2、CR3和CR4）决定了处理器的运行模式和当前正在执行的任务的特征。

> *这些寄存器在所有32位模式和兼容模式下都是32位的，同时也能在64位模式下扩展到64位。*

- CR0—包含系统控制标志，这些标志控制着处理器的运行模式和状态
- CR1—保留
- CR2—包含缺页的线性地址(引起缺页的线性地址)
- CR3—包含页表的基地址和两个标志（PCD和PWT），PCD和PWT标志控制处理器内部数据缓存中缓存页(不控制页表信息的TLB缓存)。
- CR4—包含了一组能够支持一些体系结构扩展和指示操作系统或对特定处理器功能的执行支持的标志位，可以使用 `MOV` 指令读取加载（或修改）控制寄存器

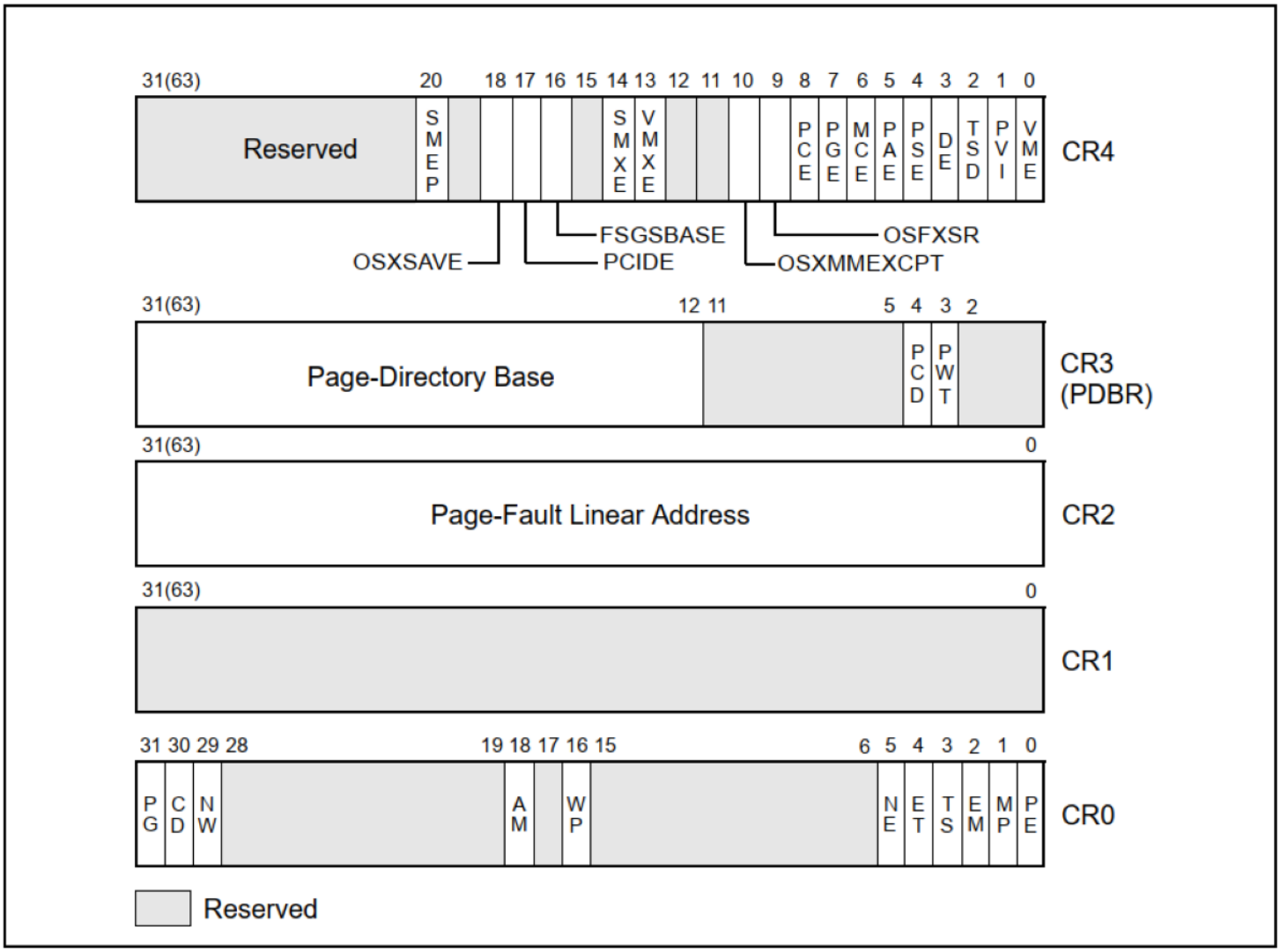


Figure 2-7. Control Registers

CR0各标志位说明

标志	说明
PG	**分页**标志,置1分页，置0不启用分页
CD	**禁用高速缓存**
NW	**不直写**，当NW和CD标志置0时，回写或直写被用来写命中缓存时的数据，并且启用失效循环。
AM	**对齐**屏蔽，置1时启用自动对齐检查，置0时禁用

标志	说明
WP	**写保护**，置1时禁止管理级的过程往用户级只读页中写，置0时允许管理级的过程往用户只读页中写。这个标志是用来创建（forking）一个新进程时实现写拷贝。
NE	**数值错误**，置1时启用内部x87 FPU错误报告机制，置0时启用类PC的x87 FPU错误报告机制。
ET	**扩展类型**，置1表示对intel 387D数学协处理器指令的支持（intel 386和intel 486处理器中）
TS	**任务切换**，允许x87 FPU、MMX、SSE、SSE2等指令上下文在任务切换上的保存延迟，处理器在每个任务开关上设置此标志，并在执行指令时进行测试。
EM	**仿真**，置1时表明处理器没有内部或外部的X87 FPU，置0时表明有X87 FPU，影响FPU、MMX、SSE、SSE2等指令
MP	**监测协处理器**，控制WAIT指令与TS标志的相关作用
PE	**保护模式**，置1时启用保护模式，置0时启用实模式。

CR3各标志位说明

标志	说明
PCD	禁用**页级缓存**，控制当前页目录是否被缓存。这个标志只影响处理器内部缓存。
PWT	**页级直写**，控制当页目录的直写（置1）或回写（置0）的缓存机制.

3.4 扩展控制寄存器(XCR0)

> *如果CPUID.01H:ECX.XSAVE为1，表示处理器支持一个或多个扩展控制寄存器(XCRs)。软件只有当CR4.OSXSAVE=1时才能访问XCR0。*

目前，唯一定义的这样的寄存器是XCR0。这个寄存器指定了操作系统在该处理器上启用的处理器状态集，例如x87 FPU、SSE、AVX状态以及将来Intel 64架构可能引入的其他处理器扩展状态。

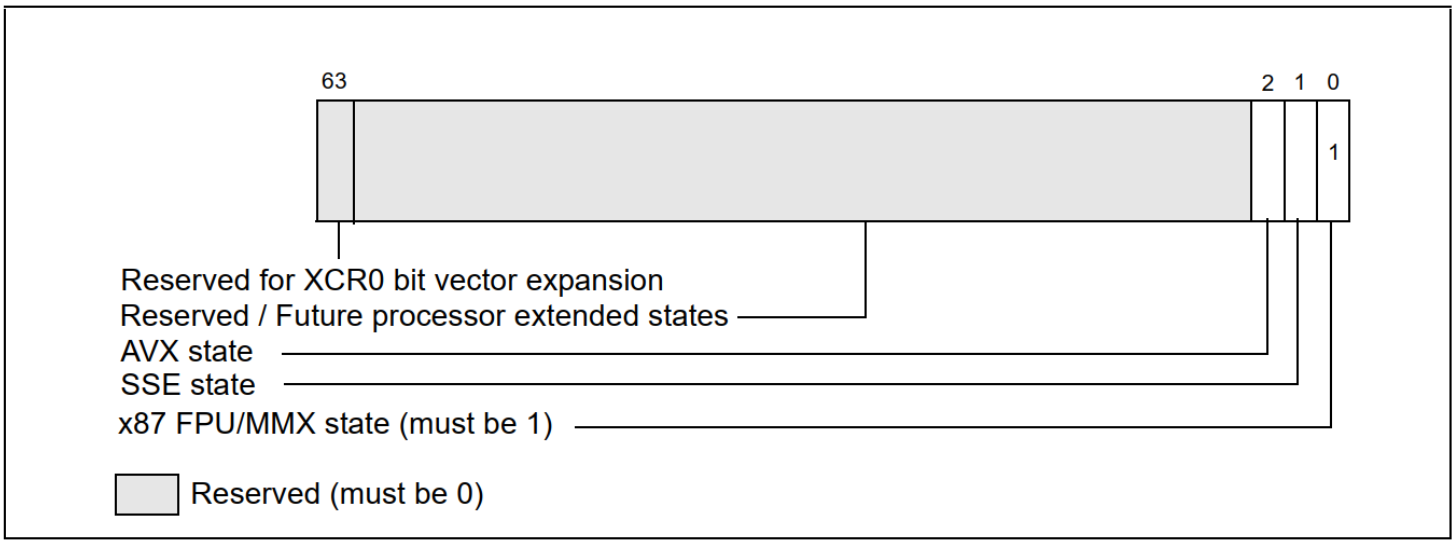


Figure 2-8. XCR0

4. 系统指令

系统指令处理系统级功能，如加载系统寄存器、管理缓存、管理中断或设置调试寄存器。

> *一些指令只能通过操作系统或0特权级程序执行,而另外一些指令可以被任何特权级别使用因而可以用于应用程序*

指令	指令描述	对应用程序（CPL为1或2）是否有用	是否受应用程序保护
LGDT	装载GDT寄存器，把GDT基地址和界限从内存中装载到GDTR中	否	是
SGDT	存储GDT寄存器，将GDTR中的GDT基地址和界限存储到内存中	否	否
LIDT	装载IDT寄存器，把IDT基地址和界限从内存中装载到IDTR中	否	是
SIDT	存储IDT寄存器，把IDTR中的IDT基地址和界限存储到内存中	否	否
LLDT	装载LDT寄存器，把LDT段选择符和段描述符从内存中装载到LDTR（段选择操作数也可以位于通用寄存器中）	否	是
SLDT	存储LDT寄存器，把LDTR中的LDT段选择符存储到内存或通用寄存器中	否	否
LTR	装载TS寄存器，把TSS段选择符和段描述符从内存中装载到TS中（段选择操作数也可以位于通用寄存器中）	否	是
STR	存储TS寄存器，把TS中当前任务TSS段选择符存储到内存或通用寄存器中	否	否