# 保护模式下的内存管理

## 内存管理概览

Intel体系结构的内存管理有两部分：分段和分页

分段可以为每个程序或者任务提供单独的代码、数据和栈模块，保证了多个进程或者任务在同一个处理器运行而不相互干扰；

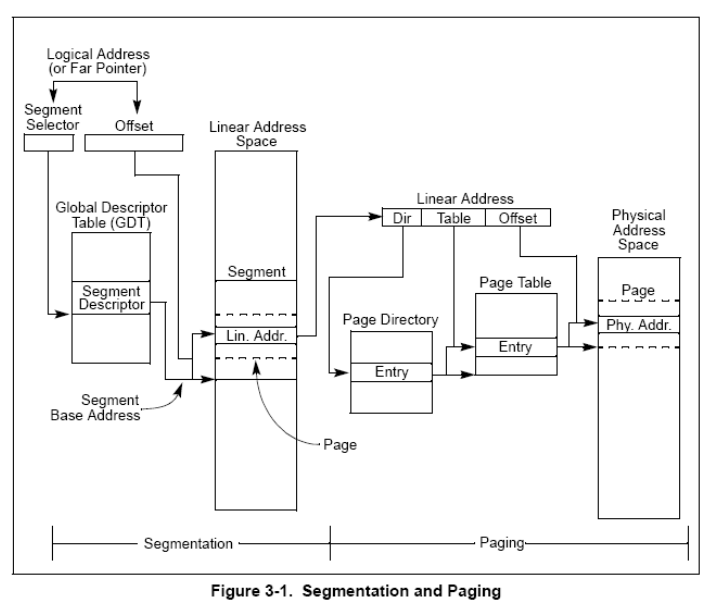
分页实现了虚拟内存系统，程序的执行代码需要被映射到物理内存中，当然分页也可以用来隔离多个任务；

在保护模式下，分段是必须的，分页是可选的；

**线性地址**：指**处理器的可寻址空间**，系统中的所有段都在处理器的线性地址空间内，分段将线性地址分为一个个小的受保护的地址空间

**逻辑地址**：可以确定一个**字节在一个特定段中的位置**；由段选择符和偏移量组成，段选择符是一个段的唯一标识，段描述符中包含了偏移量；**段基址加偏移量**就构成了处理器线性地址空间的线性地址

**物理地址**：如果系统没有采取分页机制，线性地址就可以直接映射到物理地址；物理地址就是处理器在地址总线上产生的地址范围；采用**分页机制**就可以实现线性地址空间的虚拟化，虚拟内存通过一个较小的**物理内存**（RAM、ROM）以及一些磁盘空间来模拟一个很大的线性地址空间，每个段被分成很多页（4KB），操作系统维护一个页目录和一组页表来跟踪这些页，当访问线性地空间时，处理器通过**页目录和页表将线性地址转化为物理地址**，如果被访问的不在当前的物理内存中，处理器产生中断，从磁盘读取这个页到内存，内存和磁盘的页交换对一个程序的执行来说是透明的

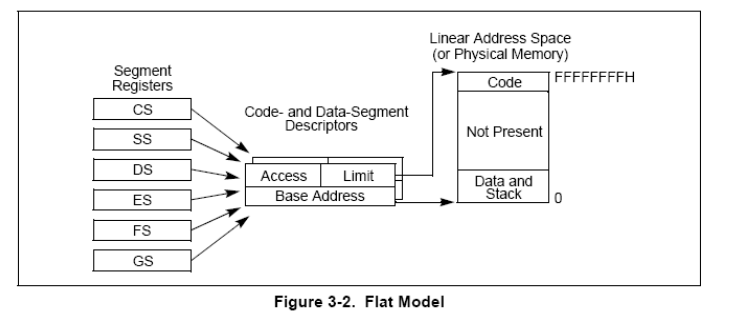


## 分段机制

### Basic Flat Model

该模式仅仅利用分段来保护程序，该模式下，操作系统和应用程序可以访问一个连续的、没有分段的地址空间，最大程度隐藏了intel架构的分段机制

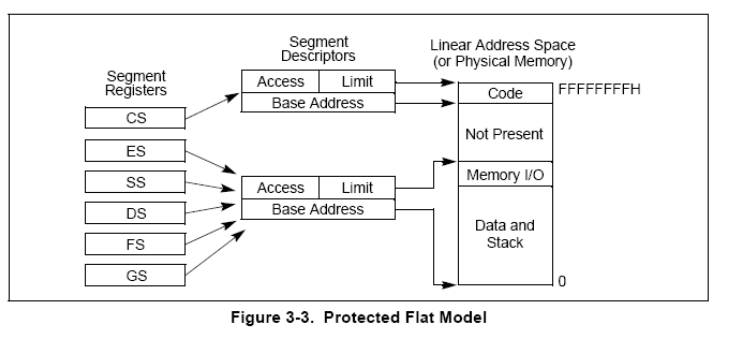
这个模型中，至少要建立两个段描述符，一个指向代码段，一个指向数据段。这两个段都要被映射到整个线性地址空间，两个段描述符都以地址0为基址，有同样的段限长4GB，即使所访问的地址处没有物理内存时，处理器也不会产生“超出内存范围”异常。ROM的地址通常位于物理地址空间的高端，处理器从0xfffffff0处开始执行。RAM位于地址空间的地段，数据段DS的初始基地址被置为0。



### Protected Flat Model

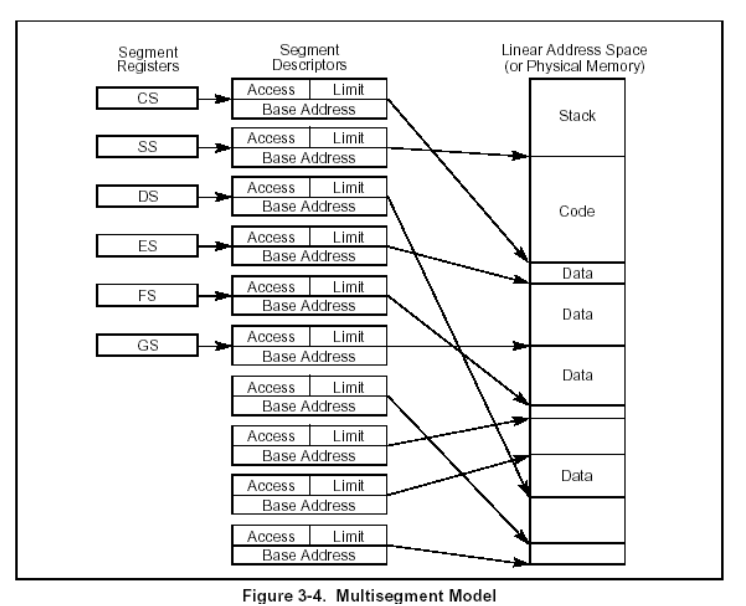
与上一种模式类似，只是段限长被设定在实际物理内存范围内，如果超出范围，会产生一个通用保护异常；

也可以在分页机制中分离普通用户和超级用户的代码和数据，只需定义优先权为3的代码段和数据段，优先级为0的代码段和数据段



### Multi-Segment Model

充分利用了分段机制，提供了对代码，数据结构以及程序的硬件级的强制保护。每个进程都被分配了自己的段描述符表以及自己的段。进程可以完全独立拥有这些分配到的段，也可以与其他进程共享这些段，单独的程序对段和执行环境的访问由硬件控制



## 逻辑地址和线性地址的转化

### Segment Selectors

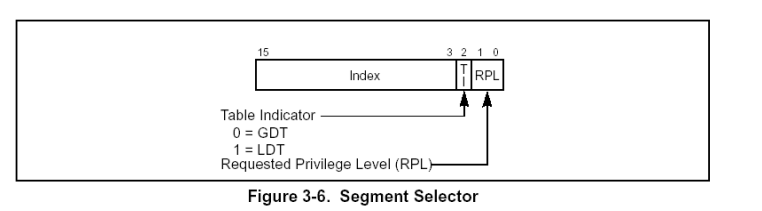
段选择子是一个16位的段标识符，它指向定义该段的段描述符；它包括以下项目：

**Index(位3~15)**，选中GDT或LDT中8192个描述符中的某个描述符。处理器将索引值乘以8，再加上GDT或LDT的基地址

**TI（位2）**，确定使用哪个描述符表，标记为0，表示用GDT。标记为1，表示用LDT

**RPL（位0和1）**，请求的特权级，确定该选择符的特权级，特权级从0到3,0为最高特权级

GDT第一项是不用的，指向GDT中第一项的段选择符被视为空段选择符；当段寄存器被赋值为空选择符时，处理器不产生异常；但当使用空选择符的寄存器来访问内存时，处理器会产生异常，空选择符可以用来初始化未使用的段寄存器



### Segment Registers

处理器提供了6个段寄存器来保存段选择符：**代码段寄存器（CS）、数据段寄存器（DS）、堆栈段寄存器（SS），还提供了另外三个数据段寄存器（ES、FS、GS）供进程使用**；只有六个段是可以被直接使用的，其他段只有在他们的段选择符被置入这些寄存器中时才可以被使用；

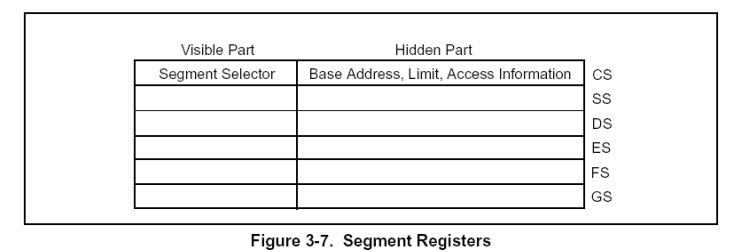
每个段寄存器由可见部分和不可见部分组成，当段选择符被加载到一个段寄存器的可见部分时，处理器也通过段选择符所指向的段描述符获取了这个段寄存器的不可见信息：段基址，段限长和访问权限，进行地址转换时，直接使用。当描述符表被改变后，软件应该重新载入段寄存器。

载入指令：

直接载入指令：MOV,POP,LDS,LES,LSS,LGS,LFS等，这些指令明确指定了相应的寄存器

隐含载入指令：CALL,JMP,RET,SYSENTER,SYSEXIT,IRET,INTnINTO,INT3等指令，他们改变了CS寄存器的内容，有时也会改变其他段寄存器的内容

MOV指令也可以将一个段寄存器的可见部分保存到一个通用寄存器中



### Segment Descriptor

段描述符是GDT或LDT中的一个数据结构，它为处理器提供诸如段基址，段大小，访问权限及状态等信息。段描述符主要是由编译器，连接器，装载器或者操作系统构造的；

段描述符中的标志和字段如下：

**段限长字段**：指定了段的大小，处理器将这两个段限长域组合成一个20位的段限长值，根据G的不同，处理器按两种方式处理段限长；G为0，则该段大小可以从1字节到1M字节，段长增量单位为字节G为1，则该段大小可以从4K字节到4G字节，段长增量单位为4K字节

**基地址域**：确定该段的第0字节在4GB线性空间中的位置，处理器将3个基地址域组合在一起，构成了一个32位的地址值。

**类型域**：指明段或者门的类型，确定段的范围权限和增长方向，如何解释这个域，取决于该描述符是应用描述符还是系统描述符，由描述符类型标志S确定。代码段。数据段和系统段对类型域有不同的意义

**S标志**：s为0表示系统描述符，s为1表示代码，数据段描述符

**DPL域**：指明该段的特权级，从0~3，0为最高特权级

**P标志**：指出该段当前是否在内存中，1表示在，0表示不在

**D/B标志**：根据这个段描述符所指向的是一个可执行代码段，一个向下扩展的数据段还是一个堆栈段，这个标志完成不同的功能。

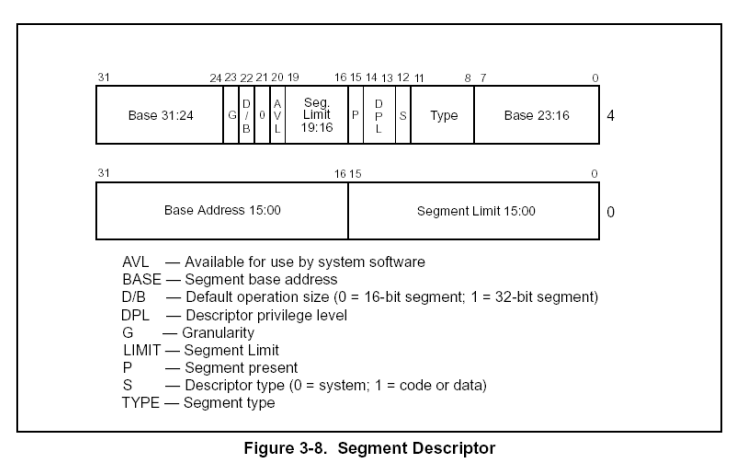
**可执行代码段**：D标志，指明了该段中的指令所涉及的有效地址值的缺省位位数和操作符的缺省位位数。标志为1，缺省为32位的地址，32位或者8位的操作符；标志为0，缺省为16位的地址，16位或者8位的操作符

**堆栈段**：B标志，它为隐含的栈操作确定栈指针的位数，标志为1，使用32位的栈指针；标志为0，使用16位的栈指针

**向下扩展的数据段**：B标志，确定了该段的地址上界，标志为1，上界为FFFFFFFFH；标志为0，上界为FFFFH。

**G标志**：确定段限长扩展的增量，G为0，段限长以字节为单位，G为1，段限长以4KB为单位

**可用及保留的位s**：段描述符的第二个双子的20位可以被系统软件使用，21位被保留，并且应该设置为0

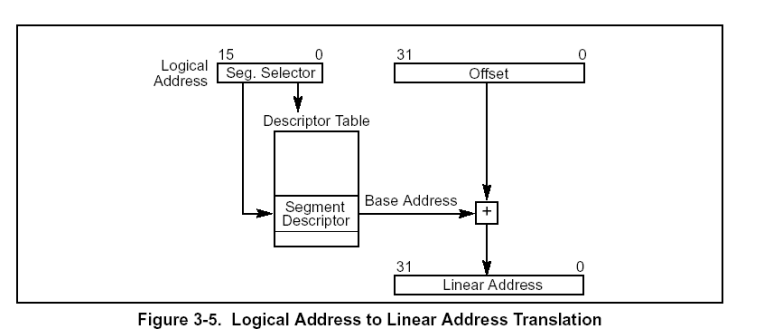


逻辑地址转换成线性地址：

（1）通过段选择符中的偏移量，在GDT或者LDT中定位该段的段描述符

（2）检查段描述符中的访问权限和段的地址范围以确保该段是可访问的，偏移量是在段限长范围内的

（3）将段描述符中的段基址和偏移量相加以构成线性地址



## 描述符的分类

### 代码、数据段描述符：

当段描述符中的S标志为1时，该描述符为代码段描述符或者数据段描述符，类型域的最高位为0：数据段描述符；为1：代码段描述符；

**数据段**：描述符类型域的低三位被解释为访问控制，是否可写，扩展方向

堆栈段必须是可读写的数据段。将一个不可写的数据段选择符置入SS寄存器会导致通用保护异常(GP)。如果堆栈段的大小需要动态变化，可以将其置为向下扩展数据段（扩展方

向标志为1)。这里，动态改变段限长将导致栈空间朝着栈底部空间扩展。如果段的长度保

持不变，堆栈段可以是向上扩展的，也可以是向下扩展的。

访问位(access位)表示访问位自最后一次被操作系统清零后，该段是否被访问过。每当

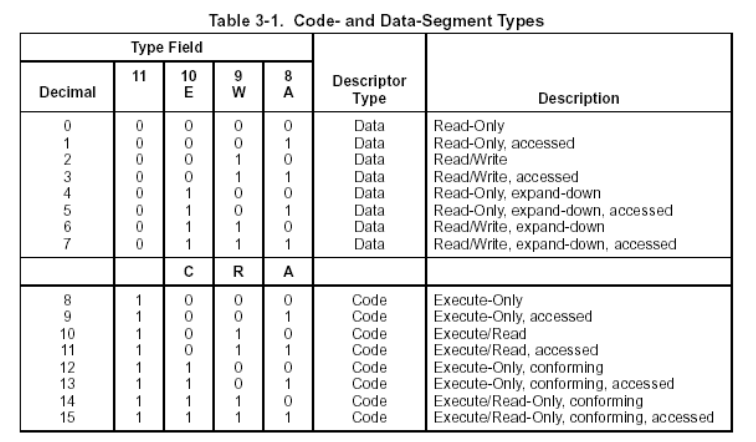
处理器将该段的段选择符置入某个段寄存器时，就将访问位置为1。该位一直保持为1直到

被显式清零。该位可以用于虚拟内存管理和debug。

**代码段**：类型域的低三位被解释为访问位，可读位，一致位，根据可读位的设置，代码段可以为“只执行”或者“可执行可读”。当有常量或者其他静态数据与指令代码一起在ROM中时，必须使用可执行可读的段，要从代码段读取数据，可以通过带有CS前缀的指令或者将代码段选择符置入数据段寄存器，在保护模式中，代码段是不可写的；

代码段可以是一致的，也可以是不一致的进程的执行转入一个具有更高特权级的一致段可以使代码在当前特权级继续运行。除非使用了调用门或者任务门，进程将转入一个不同特

权级的非一致段使处理器产生一个“一般保护异常”(GP),不访问受保护的程序和某类异常处理程序（比如除法错或者溢出）的系统程序可以被载入一致的代码段。不能被更低特权级的进程访问的程序应该被载入非一致的代码段



### 系统描述符：

当段描述符的S标志为0时，该描述符为系统描述符：

局部描述符表描述符 Local descriptor-table (LDT) segment descriptor

任务状态段描述符 Task-state segment (TSS) descriptor

调用门描述符 Call-gate descriptor

中断门描述符 Interrupt-gate descriptor

陷阱门描述符 Trap-gate descriptor

任务门描述符 Task-gate descriptor

这些描述符又分为两类：系统段描述符和门描述符；系统段描述符指向系统段；门描述符它们本身就是门，他们或者持有指向代码段的过程的入口点的指针，或者持有TSS的段选择符。

