寻址模式

从80286模型开始，Intel微处理器以两种不同的方式执行地址转换，这两种方式分别是实模式和保护模式。

段寄存器存放段选择符，一共有6个段寄存器：

cs 代码段寄存器

ss 栈段寄存器

ds 数据段寄存器

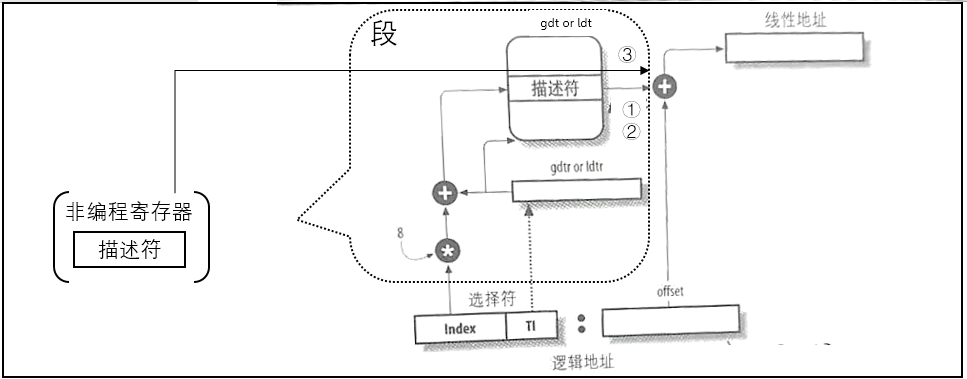
es

fs

gs

逻辑地址：

16位段选择符+32位偏移量组成



逻辑地址-> 线性地址

可以省略掉①②、直接由③进行

每个段由一个8字节的段描述符，它描述了段的特征。

段描述符放在全局描述符表（GDT）或者局部描述符表(LDT),

通常只定义一个GDT，GDT在主存的地址和大小放在gdtr控制寄存器中。每个进程如果还需要创建附加的段，可以放在自己的LDT，当前正在被使用的LDT地址和大小放在ldtr控制寄存器中。

分段可以给每一个进程分配不同的线性地址空间。分页可以把同一线性地址空间映射到不同的物理空间。

4个主要的linux段描述符字段

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 段 | 宏 | Base | Limit | S | Type | DPL |
| 用户代码段 | \_\_USER\_CS | 0x0000 0000 | 0xfffff（1Mb） | 1 |  | 3 |
| 用户数据段 | \_\_USER\_DS | 0x0000 0000 |  | 1 |  | 3 |
| 内核代码段 | \_\_KERNEL\_CS | 0x0000 0000 |  | 1 |  | 0（内核态CPL=0才能访问） |
| 内核数据段 | \_\_KERNEL\_DS | 0x0000 0000 |  | 1 |  | 0 |

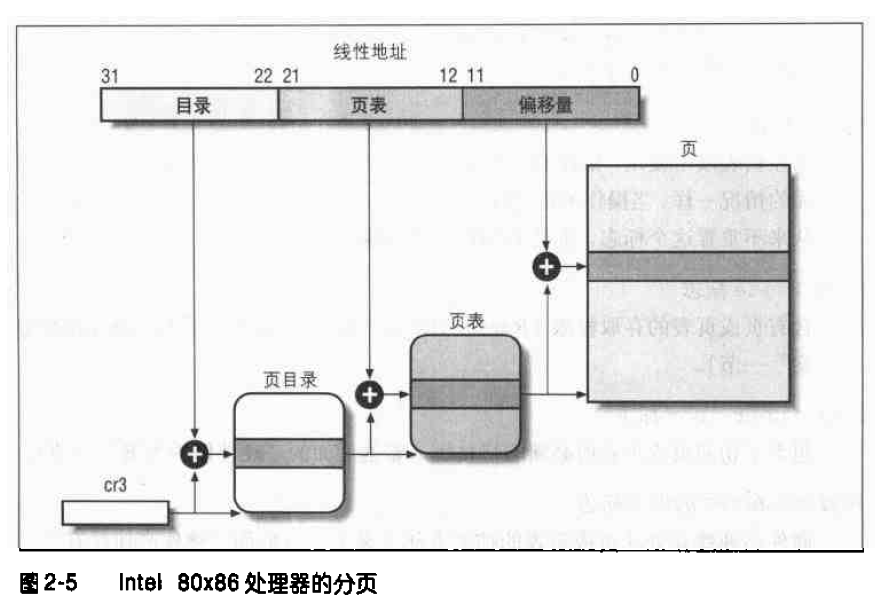
所有段都从0x0000 0000开始，

分页

从80386起，intel的分页单元处理4KB的页

cr0中pg标志。pg=0，线性地址就被解释成物理地址

每个活动进程必须分配给它一个页目录，正在使用的页目录的物理地址存放在控制寄存器cr3中。



页目录项存放页表地址。页表项存放页地址，最后加上偏移量得到。

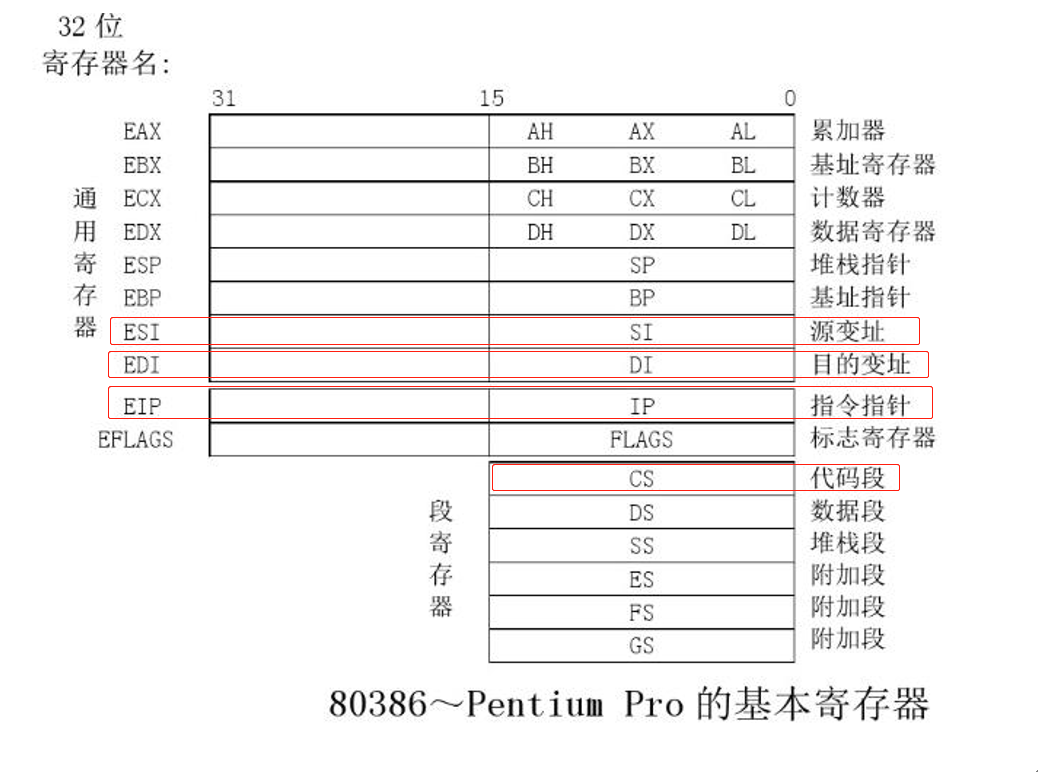
不同的进程，页目录的物理地址（cr3）不同，即使使用相同的线性地址，得到的物理地址也不同。则每个进程有自己的页

实模式(real mode)：

在实模式下，内存寻址方式和8086相同，由16位段寄存器的内容乘以16（10H）当做段基地址，加上16位[偏移地址](https://baike.baidu.com/item/%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80/3108819" \t "_blank)形成20位的物理地址，最大寻址空间1MB，最大分段64KB。可以使用32位指令。32位的x86 CPU用做高速的8086。在实模式下，所有的段都是可以读、写和可执行的。

对应【linux理解】

逻辑地址：把程序分成若干段，每一个逻辑地址都由一个段和偏移量组成，偏移量指明了从段开始的地方到实际地址之间的距离（局部偏移）



**实模式的寻址方式与工作机理**

8086的段寄存器和IP寄存器都是16位的，如果按照原先的方式，把段寄存器的内容和偏移地址直接相加来形成物理地址的话，也只能得到 16 位的物理地址。8086却提供了20根地址线。

 16 位的物理地址只能访问 64KB 的内存，地址范围是0000H～FFFFH，共 65536 个字节。

 20 位的物理地址则可以访问多达1MB的内存，地址范围从 00000H到FFFFFH。问题是，16 位的段地址和 16 位的偏移地址相加，只能形成 16 位的物理地址，怎么得到这 20位的物理地址呢？

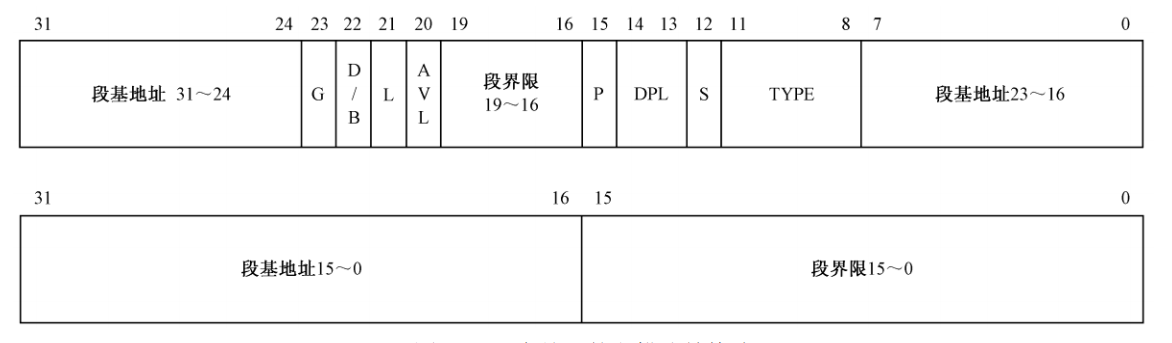
先将段寄存器的内容左移 4 位（相当于乘以十六进制的10，或者十进制的 16），形成 20 位的段地址，然后再同16位的偏移地址相加，得到20位的物理地址。比如，对于逻辑地址 F000H:052DH，处理器在形成物理地址时，将段地址F000H左移 4 位，变成 F0000H，再加上偏移地址052DH，就形成了20位的物理地址 F052DH。

**b.保护模式的寻址方式和工作原理**

　　在保护模式下的寻址相对实模式来说是比较复杂的，而且涉及到一些概念问题，下面我们先来简单介绍一下，后期操作系统的系列文章会在有相关知识点的时候进行更为详细的讲解。

　　其实在保护模式下地址的表示方式与实模式是一样的，都是：段（segment）:偏移（offset），不过保护模式下，“段”的概念发生了根本性的改变。实模式下，段值还是可以看作是地址的一部分，比如段值为xxxxh表示以xxxx0h开始的一段内存。而保护模式下，虽然段值仍然由原来的cs、ds等寄存器表示，但此时它仅仅变成了一个索引，这个索引指向了一个数据结构的一个表项，表项中详细定义了段的起始地址、界限、属性等内容。这个数据结构就是全局描述符GDT（也有可能是LDT，这里我们先介绍GDT，LDT其实跟GDT差不多，我们后期在详讲）。

　　GDT的作用是用来提供段式存储机制，这种机制是段寄存器和GDT中的描述符共同提供的。每个描述符在GDT中占8字节，也就是 2 个双字，或者说是 64 位。下图中，下面是低32位，上面是高32位。



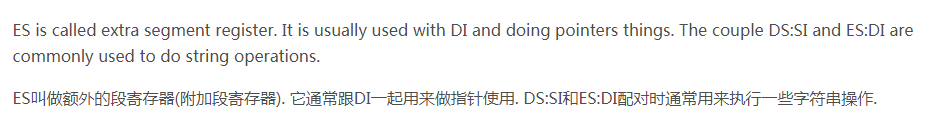
　　很明显，描述符中指定了 32 位的段起始地址，以及 20 位的段边界。在实模式下，段地址并非真实的物理地址，在计算物理地址时，还要左移 4 位（乘以 16）。和实模式不同，在 32 位保护模式下，段地址是 32 位的线性地址，如果未开启分页功能，该线性地址就是物理地址。

假设我们在ES中存入0x1000，DI中存入0xFFFF,那么

地址

ES:DI=0x1000\*0x10+0xFFFF=0x1FFFF,

这就是众所周知的“左移4位加偏移”。



进程切换

创建进程和创建轻量级进程？这两种进程有什么区别