CPU的实模式和保护模式（一）

> 文章来源：

总结：实模式和保护模式：这包括**16**位模式即实模式，**32**位模式即保护模式，以及32位模式下的16位兼容模式（首先需要明确的是保护模式有着不同的内存访问机制，简单地说就是传统的涉及段的几个寄存器如CS,DS,ES,SS等被解释为不同的内容——实模式解释为段寄存器，保护模式解释为段选择子，可以先理解为换一种叫法，但要注意它们保存的内容也会有着不同的处理方式）

> 实模式和保护模式：实模式是在*x86*的*16*位系统时称呼的，此时要寻址*1M*的物理内存地址，就用 段基址:段偏移量表示，此时的模式就称为真实物理地址模式，实模式；而对于后续发展到*x86*的*32*位计算机时，为了向后兼容 *16*位计算机的表示方式，此时段寄存器被称为段选择子，存储的内存也不再是段基址，而是*GDT*的索引值、*RPL*（请求特权级）和访问的是*GDT*还是*LDT*，此时*32*为计算机称为保护模式。

实模式和保护模式都是CPU的工作模式，而CPU的工作模式是指CPU的寻址方式、寄存器大小等用来反应CPU在该环境下如何工作的概念。

1.实模式工作原理

实模式出现于早期8088CPU时期。当时由于CPU的性能有限，一共只有20位地址线（所以地址空间只有1MB），以及8个16位的通用寄存器，以及4个16位的段寄存器。所以为了能够通过这些16位的寄存器去构成20位的主存地址，必须采取一种特殊的方式。当某个指令想要访问某个内存地址时，它通常需要用下面的这种格式来表示：

- (段基址：段偏移量)

其中第一个字段是段基址，它的值是由段寄存器提供的(一般来说，段寄存器有6种，分别为cs，ds，ss，es，fs，gs，这几种段寄存器都有自己的特殊意义，这里不做介绍)。

第二字段是段内偏移量，代表你要访问的这个内存地址距离这个段基址的偏移。它的值就是由通用寄存器来提供的，所以也是16位。那么两个16位的值如何组合成一个20位的地址呢？CPU采用的方式是把段寄存器所提供的段基址先向左移4位。这样就变成了一个20位的值，然后再与段偏移量相加。

即：

- 物理地址 = 段基址<<4 + 段内偏移
- 所以假设段寄存器中的值是0xff00，段偏移量为0x0110。则这个地址对应的真实物理地址是 0xff00<<4 + 0x0110 = 0xff110。

- 通过讲述段寄存器在加法器中左移四位，从而使得段地址寄存器的高4位变成了20位物理地址的高四位。示例如下：
  段地址：0000-1000-0100-1000
  偏移地址：0100-0010-0010-0001
  段地址左移四位得到：**1000**-0100-1000-0000-0000
  偏移地址与段地址相加：**1000**-1000-1010-0010-0001，此时就得到了20位的物理地址。

由上面的介绍可见，实模式的"实"更多地体现在其地址是真实的物理地址。

2.保护模式工作原理

随着CPU的发展，CPU的地址线的个数也从原来的20根变为现在的32根，所以可以访问的内存空间也从1MB变为现在4GB，寄存器的位数也变为32位。所以实模式下的内存地址计算方式就已经不再适合了。所以就引入了现在的保护模式，实现更大空间的，更灵活也更安全的内存访问。

在保护模式下，CPU的32条地址线全部有效，可寻址高达4G字节的物理地址空间;但是我们的内存寻址方式还是得兼容老办法(这也是没办法的，有时候是为了方便，有时候是一种无奈)，即(段基址：段偏移量)的表示方式。当然此时CPU中的通用寄存器都要换成32位寄存器(除了段寄存器，原因后面再说)来保证寄存器能访问所有的4GB空间。

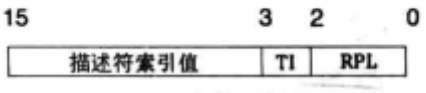
我们的偏移值和实模式下是一样的，就是变成了32位而已，而段值仍旧是存放在原来16位的段寄存器中，但是这些段寄存器存放的却不再是段基址了，毕竟之前说过实模式下寻址方式不安全，我们在保护模式下需要加一些限制，而这些限制可不是一个寄存器能够容纳的，于是我们把这些关于内存段的限制信息放在一个叫做全局描述符表**(GDT)**的结构里。全局描述符表中含有一个个表项，每一个表项称为段描述符。而段寄存器在保护模式下存放的便是相当于一个数组索引的东西，通过这个索引，可以找到对应的表项。段描述符存放了段基址、段界限、内存段类型属性(比如是数据段还是代码段,注意一个段描述符只能用来定义一个内存段)等许多属性,具体信息见下图：

其中，段界限表示段边界的扩张最值，即最大扩展多少或最小扩展多少，用20位来表示，它的单位可以是字节，也可以是4KB，这是由G位决定的(G为1时表示单位为4KB)。

实际段界限边界值=(描述符中的段界限+1)*（段界限的单位大小(即字节或4KB))-1，如果偏移地址超过了段界限，CPU会抛出异常。

全局描述符表位于内存中，需要用专门的寄存器指向它后， CPU 才知道它在哪里。这个专门的寄存器便是**GDTR**(一个48位的寄存器),专门用来存储 GDT 的内存地址及大小。

最后我们再介绍一下一个新的概念：段的选择子。段寄存器 CS、 DS、 ES、 FS、 GS、 SS，在实模式下时，段中存储的是段基地址，即内存段的起始地址。 而在保护模式下，由于段基址已经存入了段描述符中，所以段寄存器中再存放段基址是没有意义的，在段寄 存器中存入的是一个叫作选择子的东西。选择子“基本上”是个索引值，虽然它还有其他内容，不过作为初学者暂时忽略也没太大关系。由于段寄存器是 **16** 位，所以选择子也是 **16** 位，在其低 **2** 位即第 **0**～**1** 位， 用来存储 **RPL**，即请求特权级(有兴趣的可以了解一下，不想了解的忽略即可，跟用户态和内核态相关的)，可以表示 0、 1、 2、 3 四种特权级。在选择子的第 **2** 位是 **TI** 位，即 Table Indicator，用来指示选择子是在 GDT 中，还是 LDT 中索引描述符。 TI 为 **0** 表示在 **GDT** 中索引描述符， **TI** 为 **1** 表示在 **LDT** 中索引描述符。选择子的高 **13** 位，即第 **3**～**15** 位是 描述符的索引值，用此值在 **GDT** 中索引描述符。前面说过 GDT 相当于一个描述符数组，所以此选择子中的索引值就是 **GDT** 中的下标。选择子结构如下：



此外， 扩充的存储器分段管理机制和可选的存储器分页管理机制，不仅为存储器共享和保护提供了硬件支持，而且为实现虚拟存储器提供了硬件支持; 支持多任务，能够快速地进行任务切换(switch)和保护任务环境(context); 4个特权级和完善的特权检查机制，既能实现资源共享又能保证代码和数据的安全和保密及任务的隔离; 支持虚拟8086方式，便于执行8086程序。

对实模式/保护模式的三种访问内存机制的理解

假设x86处理器执行以下的指令

```
1 mov ds, cx    ;其中假设 (cx) = 0x0010
```

主要讨论实模式和保护模式：这包括16位模式即实模式，32位模式即保护模式，以及32位模式下的16位兼容模式（首先需要明确的是保护模式有着不同的内存访问机制，简单地说就是传统的涉及段的几个寄存器如CS,DS,ES,SS等被解释为不同的内容——实模式解释为段寄存器，保护模式解释为段选择子，可以先理解为换一种叫法，但要注意它们保存的内容也会有着不同的处理方式）

（在讨论之前我想先说说下面我用红字标出来的内容是相对于其他模式寻址机制不同的地方，其次我会用“斜体的引用”注解新引入的概念）

1、16位模式/实模式：

执行 mov ds, cx 时，DS解释为段寄存器，0x0010作为段基址

先直接将0x0010赋给DS

然后当访问涉及内存的指令如 mov [0x00], al 时，处理器自动完成段基址0x0010左移4位再加上偏移地址（这个例子是0x00）

最后访问该过程得到的20位物理地址（这个例子是0x00100）便是该模式访问内存的过程

2、32位下的（兼容）实模式——比如保护模式下运行**8086**程序，或者说**32**位模式下运行**16**位程序

执行 mov ds, cx 时，DS仍然解释为段寄存器，0x0010也仍然作为段基址

先直接将0x0010赋给DS

然后当访问涉及内存的指令如 mov [0x00], al 时，处理器也是自动完成段基址0x0010左移4位，不过之后处理器会把左移4位后产生的地址传送到描述符高速缓存器（此时描述符高速缓存器的内容是0x100），再加上偏移地址（该例子中是0x00）

最后访问该过程得到的20位物理地址（这个例子是0x00100）便是该模式访问内存的过程

这里简单解释一下“描述符高速缓存器”：

> *32*位模式下寻址空间必然会增大，这就使得原来实模式（*16*位）寻址机制需要扩展，相应地段寄存器也需要扩展。简单地说就是在*32*位模式下，段寄存器除了包括原来的*CS,DS*这一类传统的段相关的寄存器，还包括一个对程序员来说不可见的部分——这个不可见部分就是描述符高速缓存器，它的作用就是缓存段基址（你可以能会疑惑为什么保护模式比实模式要多一步缓存，这个缓存的意义是什么？别急，在说完下面保护模式的寻址之后，我会在本博文最后附上我的理解）。

3、32位模式/保护模式：

执行 mov ds, cx 时，DS此时解释为段选择器，0x0010此时作为段选择子

这里也简单解释一下“段选择子”：

> 段选择子：段描述符在段描述符表中的索引号（可以先认为表项在表中的项号，但是实质上段选择子除了索引号还有其他组成部分）

那段描述符和段描述符表又是什么呢？

> 这得先从保护模式是什么讲起。一个程序在实模式下设置的数据段是任意的，也就是说可以设置在*1MB*范围内的随意一个区域——这就使得程序可以在即使不属于它自己的内存单元的访问也没有一点障碍，这是非常不安全也是不合理的。
>
> 而保护模式对实模式一个很重要的改进就是，保护模式让每个程序都先定义它自己能够访问的内存区域，这样每当一个程序想要非法地访问不属于它自己的区域时能有依据地被制止。而每个程序能够访问的内存区域实质上是其定义的数据段、代码段、栈段等，这些段的定义信息就被保存于内存的某个区域内
>
> 这个区域是一堆这样的“段定义信息”，这个区域便是段描述符表（*Global Descriptor Table, GDT*），而这一个个组成整个表的表项就是段描述符（*Segment Descriptor*）

此时处理器会先将段选择子（段选择子可以指示索引号）乘以**8**作为偏移地址，再同**GDTR**（Global Descriptor Table Register，段描述符表寄存器）中提供的GDT表的基地址相加，求得结果指向GDT表中某个表项（即某个段描述符），并加载表项中的该段的基地址到描述符高速缓存器

这里也简单解释一下为什么要“段选择子乘以*8*”，以及“*GDTR*”是什么

> 段选择子乘以*8*：在段描述符表中，每个表项长*8*字节，每个表项从*0*开始编排索引号。所以当我们访问*GDT*的时候，先读取*GDT*表头地址（也就是索引号*#0*的地址），这样每个表项的地址便可用相对*GDT*表头地址来表示——比如我们读取到*GDT*表头地址为*0xXXXX_XXX0*，则索引号*#0*地址为*0xXXXX_XXX0*，索引号*#1*地址为*0xXXXX_XXX8*（前面是索引号*#0*，长*8*字节）...——这样便可以用段选择子（段选择子可以指示索引号）乘以*8*来找出段描述符的地址，这个地址便是段描述符相对于*GDT*表头的地址
>
> *GDTR*（*Global Descriptor Table Register*，段描述符表寄存器）：主要作用是用来找到*GDT*，该寄存器保存有*GDT*在内存中的起始地址

然后当访问涉及内存的指令如 mov [0x00], al （32位模式也可用8位操作数）时，处理器将描述符高速缓存器中的段基地址加上该指令中给出的偏移地址

最后访问该过程得到的32位物理地址（这个例子是0x00000100）便是该模式访问内存的过程

最后我想注解一下第一个引用结尾提出来的问题：什么保护模式比实模式要多一步缓存，这个缓存的意义是什么？

> 经过保护模式寻址的讨论，我们可以知道保护模式的寻址最简单地可以理解为多增加了一步——查表，为什么需要查表？因为实模式无限制的寻址并不安全，所以保护模式需要先在表中定义每个段所能访问的地址空间（这是最简单的理解）。而查表会带来开销，对于指令给出一个偏移地址。
>
> - 思考实模式的寻址机制，实模式是每给一个偏移地址就进行左移*4*位再相加的动作；而到了保护模式，如果也像实模式这样每给出一个偏移地址就查表相加，这一步是存在开销的，所以此时引入的缓存机制，可以避免每次访问内存的操作时都进行一次查表操作。这便是我的粗略理解。*

补充：

- 百度百科：保护模式