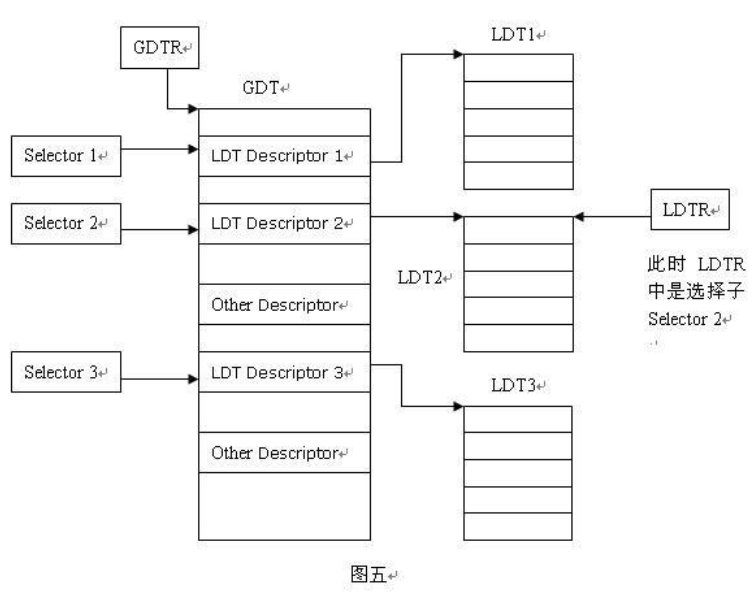
补充内容：

LDT属于程序，GDT属于系统，同一台计算机上的所有程序共享一个GDT。  
LDT描述局部于每个程序的段，包括其代码、数据、堆栈等。

GDT描述系统段，包括操作系统本身、TSS、LDT1、LDT2。。。。。



**https://blog.csdn.net/chen1540524015/article/details/74075252**

**1 任务状态段（TSS）**   
任务状态段（Task-State Segment(TSS)），保存任务状态信息的系统段为任务状态段。其内容即是trapframe

**2 TSS 描述符**

就像其它段描述符描述段的一些性质一样，TSS描述符用来描述TSS的某些性质。TSS描述符仅可能存放在GDT中，不能存放在LDT或IDT中。32位模式下，TSS描述符格式如下：

当G标志是0时，界限字段的值大于或等于67H（103字节），比TSS（104字节）的大小少一个字节。如果切换到任务的TSS界限字段小于67H会产生无效TSS异常（#TS）。

**3 任务寄存器（task register）**

任务寄存器拥有当前任务的TSS的段选择符和段描述符（32位基地址，16位段界限和描述符参数）。任务寄存器具有可见部分和不可见部分（类似段寄存器）。任务寄存器指向GDT中TSS描述符。 当执行任务切换时，处理器会把新任务的TSS的段选择符和段描述符自动加载进任务寄存器TR中。

**4 任务门描述符（Task-Gate Descriptor）**

略

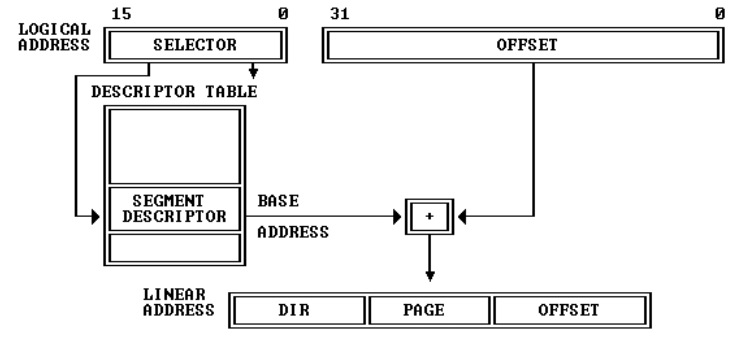
正文：

<https://pdos.csail.mit.edu/6.828/2018/readings/i386/s05_01.htm>

### 5.1 段翻译

图5.2显示了cpu怎样将逻辑地址转换为线性地址.（CS:IP就是逻辑地址，线性地址就是当前进程运行在自己的地址空间的地址，线性地址用分页机制转化为物理地址）

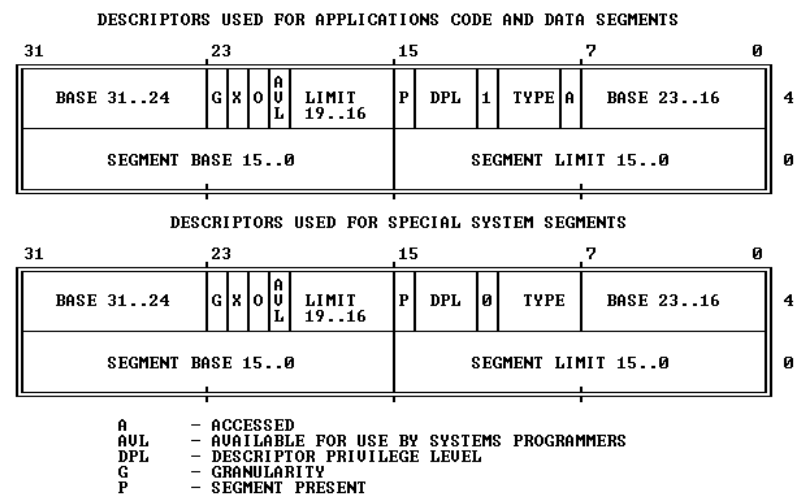
为了完成这一转换，cpu使用了下面的数据结构



1. 描述符
2. 描述符表
3. 选择器
4. 段寄存器

段描述器给cpu提供需要的数据，将逻辑地址转化为线性地址，

描述器有编译器，连接器，loader或者操作系统创建。图5.3介绍了两个描述器格式。所有类型的描述器都是这两个格式中的一个。



描述器=base + limit + granularity bit + type + DPL + segment present bit

**Base:**定义在4G空间内segment的地址，cpu基地址的这三个部分合在一起，生成32位值。

**Limit:**定义segment的大小，当cpu把limit的两个部分合在一起，生成一个20的数。Cpu以两种方式解释limit field,者取决于**granularity**的大小。

1如果granularity是1bit，定义limit最多1M

2如果是4byte,limit最多4G，limit在加载时左移12位，然后插入低位的1bit

**Type:**区别两种类型的描述器

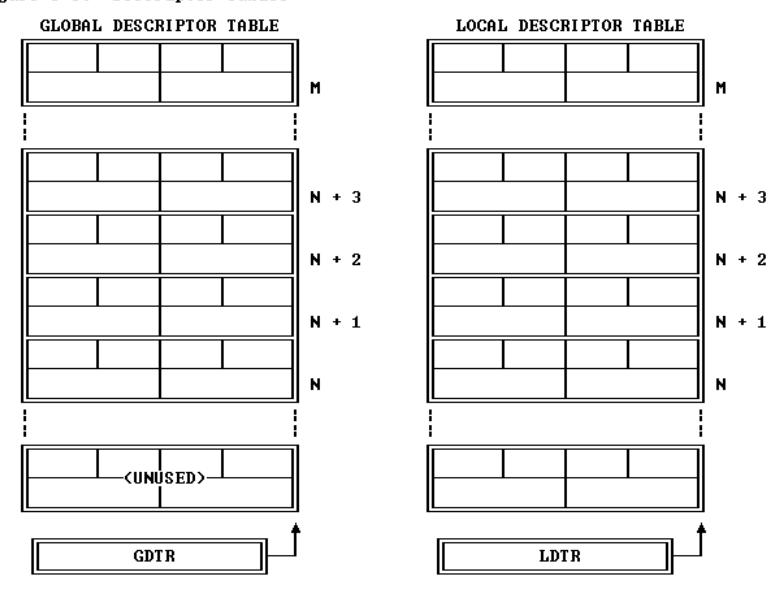
创建和维护描述器是系统软件的任务，经常需要和编译器，程序加载器结合在一起使用。

### 5.1.2描述器表

描述器存在GDT或者LDT中

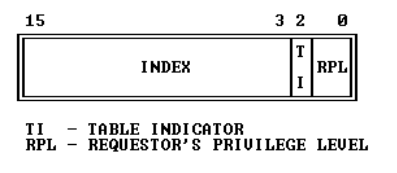
如图所示，描述表是包含描述器的8位entries，一般不适用第一个

Cpu通过LDTR和GDTR寄存器定位GDT和当前LDT，这些寄存器存储这些表在线性空间的基地址，并存储segment limit，指令LGDT和SGDT来过去GDTR，LLDT和SLDT获取LDTR。



### 5.1.3 选择器

逻辑地址的选择部分通过声明描述器表和索引描述器表中的描述器来定位描述器。选择器作为一个带有指针的field对应用可见，但是选择器的值通过由链接器和loader赋值，下面是选择的形式：



**Index:**选择某个描述器，cpu将index乘以8，然后把结果加到描述器表的基地址上，来获取表中合适的描述器。

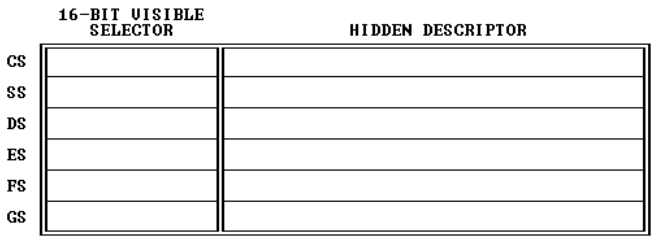
**Table indicator:**声明selector指向哪一个描述器表。零=GDT,1=LDT

由于CPU不使用表中第一个描述器，index=0的选择器和table indicator=0可以作为NULL selector，当段寄存器加载一个NULL selector，cpu不产生exception。但是当段寄存器用来获取内存时，cpu会产生exception，此功能对于初始化未使用的段寄存器以捕获意外引用非常有用。

### 5.1.4 段寄存器

Cpu将描述器的信息存入段寄存器。

每个段寄存器有一个可视部分和不可视部分，如图所示



**可视部分由程序操作，就好像他们是一个简单的16位寄存器一样。**

不可视部分由cpu操作

加载这些寄存器的命令如下：

直接加载：[MOV](https://pdos.csail.mit.edu/6.828/2018/readings/i386/MOV.htm), [POP](https://pdos.csail.mit.edu/6.828/2018/readings/i386/POP.htm), [LDS](https://pdos.csail.mit.edu/6.828/2018/readings/i386/LGS.htm), [LSS](https://pdos.csail.mit.edu/6.828/2018/readings/i386/LGS.htm), [LGS](https://pdos.csail.mit.edu/6.828/2018/readings/i386/LGS.htm), [LFS](https://pdos.csail.mit.edu/6.828/2018/readings/i386/LGS.htm).

隐式加载：[CALL](https://pdos.csail.mit.edu/6.828/2018/readings/i386/CALL.htm) and [JMP](https://pdos.csail.mit.edu/6.828/2018/readings/i386/JMP.htm)

使用这些命令，应用程序可以加载可是部分的16位

**Cpu自动获取及地址，limt，type等等，并把它们加载到不可视部分。**

由于大多数指令是指段中的数据，而段中的选择器已经加载到段寄存器中，因此处理器可以将指令提供的段相对偏移量添加到段基地址，而无需额外的开销。