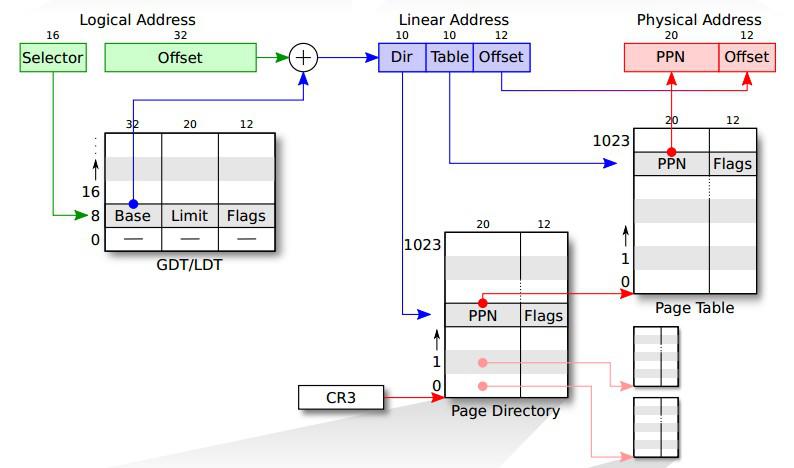
Logical Address ， Linear Address ， Physical Address

Linear Address 是转换过程中出现的中间态而已。虚拟内存中和代码上能看到的内存地址一般都是Logiccal Address （带冒号的除外）。

这几个地址是如何转换的：

首先拿到一个Logical Address，然后去查GDT表，根据选择子找到Base，再把base 和 Offset加一下就是Linear Address，最主要的部分就是如何从Linear Adress 找到 对应的Physical Address。如下图所示把Linear Address 分成三段，前10为时dir，中间10位是Table，最后12位是offset。根据dir去查找一级页表，即页表目录，页表的基值放在CR3（页表基值寄存器）寄存器中，找到之后就可以读取出PDE，通过PDE的前20位VPN找到二级页表，以Table作为偏移找出PTE，将PTE中的前20位PPN加上Offset就找到了对应的物理地址。



![](../../image/address\-translation/L\-L\-P\-Address.jpg)

操作系统是一个复杂的系统，

操作的系统的演变应该从8086芯片讲述到80386芯片可能才算是入门吧

8086

首先是8086的寄存器8086芯片中一共14个寄存器，且都是16位

其中这些寄存器又可分为

通用寄存器

控制寄存器

段寄存器

通用寄存器：

AX SP

BX BP

CX SI

DX DI

段寄存器

CS(Code):代码段

DS(Code)：数据段

SS(Stack)：堆栈段

ES(Extra)：附加段

控制寄存器：

IP：指令指针寄存器

FLAG：标志寄存器

1. 地址线宽于数据的矛盾

8086芯片中为了能够利用上尽量大的内存设计了20位的地址总线，而数据总线是16位的因此为了能够利用20为的地址总线采用了段：偏移的寻址方式；计算方式为段\*16+偏移地址由于段与偏移量都是16位的因此最大的可寻址RAM大小为0-0x10ffef（1M零64KB）

（2）8086CPU中没有保护模式

（3）8086芯片中内存中的相关信息

从地址为0的地方开始，这里存放着，CPU和外设的相关信息然后是终端向量表，之后一直到640KB为止是用户数据和操作系统代码和数据。之后的640KB到1M地址的位置，被保留硬件保留。

+------------------+ <- 0xFFFFFFFF (4GB)

| 32-bit |

| memory mapped |

| devices |

| |

/\/\/\/\/\/\/\/\/\/\

/\/\/\/\/\/\/\/\/\/\

| |

| Unused |

| |

+------------------+ <- depends on amount of RAM

| |

| |

| Extended Memory |

| |

| |

+------------------+ <- 0x00100000 (1MB)

| BIOS ROM |

+------------------+ <- 0x000F0000 (960KB)

| 16-bit devices, |

| expansion ROMs |

+------------------+ <- 0x000C0000 (768KB)

| VGA Display |

+------------------+ <- 0x000A0000 (640KB)

| |

| Low Memory |

| |

+------------------+ <- 0x00000000



8086中之后CPU的实模式而不存在保护模式

而且在1M的RAM中0-640KB才是整个系统和用户能使用的空间之上一直到1M的空间范围内是VGA映射，各种硬件的映射和BIOS的硬连线的地址

80286

1.我只是个配角

对于 80286 CPU 来说，其地址总线数目达到了 24 根，从而最大寻址能力为 224 即 16MB，

由于支持更多的物理内存寻址，因此 80286 便开始成为了多任务，多用户系统的核心。

80386

而后来，Intel  又推出了 80386 ，80386 为 32 位微处理器，Intel 80x86 家族的 32 位微处理器始于 80386；

同时 80386 也完全兼容先前的 8086/8088，80186，80286，并且 80386 全面支持 32 位数据类型和 32 位操作，

并且 80386 的数据总线根数和地址总线根数均达到了 32 根，从而可以最大物理寻址为 232即 4GB 。

而之后的 80486 也是 32 位微处理器，而后又出来了 Pentium 和 Pentium Pro 等等第五代微处理器，

这些处理器虽然也是 32 位微处理器，但是他们的数据总线和地址总线都有所扩展，

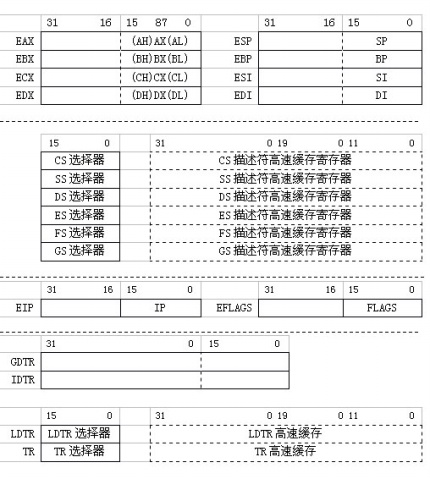
比如 Pentium 的数据总线达到 64 位，而 Pentium Pro 的地址总线位数达到了 36 位 。

在经过80286之后x86CPU结构在进入80386芯片之后基本上算是定型。相比与8086芯片的巨大进步在于：

1. 实现了多任务
2. 实现了保护模式

以上两点构成了今天所有现代操作系统的基石，因为这两个解决了计算机效率低下和程序健壮性这两个问题，从此计算机的发展越来越迅速。

寄存器：



另外增加的还有：

标志和控制寄存器：

机器状态字：CR0

Intel预留：CR1

页面故障地址：CR2

页目录地址：CR3

调试和测试寄存器：

TR6 32位测试命令寄存器

TR7 32为数据寄存器保存转换旁路缓冲器测试数据

##系统地址寄存器的相关解释

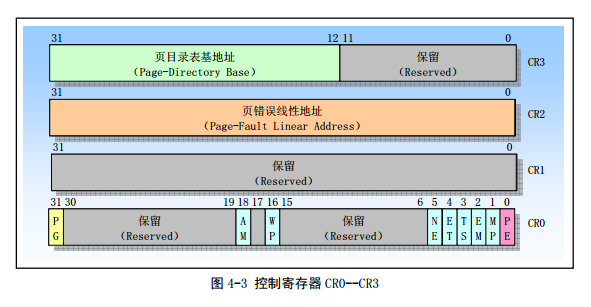
全局描述符表寄存器GDTR（Global Descriptor Table Register）是48位寄存器用来保存全局描述符表GDT的32位基地址和GDT的大小（16位）

• 中断描述符表寄存器IDTR（Interrupt Descriptor Table Register），是48 位寄存器，用来保存中断描述符表（IDT）的32 位基地址和IDT 的大小（16位）。

局部描述符表寄存器LDTR（Local Descriptor Table Register ），是16 位寄存器，保存局部描述符表LDT 段的选择符。（由于都是段描述符所以只要指向段描述符就能达到效果）

• 任务状态寄存器TR（Task State Register）是16 位寄存器，用于保存任务状态段TSS 段的16 位选择符。

<https://www.cnblogs.com/alantu2018/p/8471955.html>



寻址

3.一切都是兼容惹的祸

寻址的过程：

（1）逻辑地址 线性地址（物理地址）

以上状态是CPU之开启了分段而没有开启分页所有当我们得到线性地址之后直接把线性地址当作物理地址来进行内存的访问。

（2）逻辑地址 线性地址 物理地址

逻辑地址经过分段之后转化为线性地址，之后线性地址经过分页机制转化为物理地址之后，在进行特定地址的访问。

2．分段机制

80386之中比较重要的相关寄存器都升级为32位的寄存器，而只有C D S E F G段仍然是16位的寄存器，这样保持了向上的兼容，同时在16位段寄存器旁边设置了程序员不可见的描述符高速缓冲寄存器，而且16位的段寄存器也不再盛放地址，而是盛放段选择符。

由于保护模式的需要，从此以后段的定义就发生了根本的改变。

段描述符：

大小：8字节64位，每个段对应一个描述符

分类：（大类），代码数据段描述符类型，系统描述符类型（由标志位S的标志确定）

内容：定义了段的基址（段的起始地址用于形成线性地址）

，段的限长（说明该段的长度用于储存空间的保护《保护模式的基础》）

，访问类型（说明该段的访问权限，该段在当前内存中的存在性，段所在的特权级）等

系统描述符类型的分类：

当段描述符中的S标志（描述符类型）是复位状态（0）的话，那么该描述符是一个系统描述符。处理器能够识别以下一些类型的系统段描述符：

1. 局部任务段描述符表（LDT）的段描述符
2. 任务状态段（TSS）描述符
3. 中断门描述符
4. 陷阱门描述符
5. 任务门描述符

这些描述符类型又可以分为两大类：系统段描述符fdf和门描述符。系统段描述符指向系统段（如LDT和TSS段），门描述符就是一个门。对于调用、中断或陷阱门。其中含有代码段的选择符和段中程序入口点的额指针；对于任务们，其中含有TSS的段选择符。下表给出了系统段描述符和门描述符类型字段的编码。



段描述符表：

定义：IA-32处理器把所有的段描述符按照顺序组织成线性表，放在内存中按照顺序组织成线性表，放在内存中，称为段描述符表。

分类：全局描述表GDT

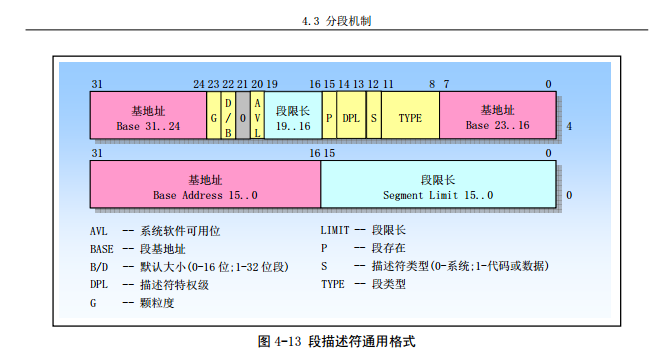
局部描述符表LDT

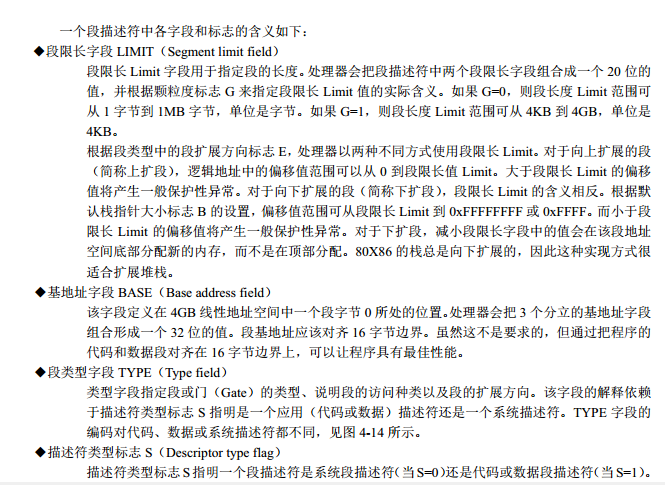
中断描述符表IDT

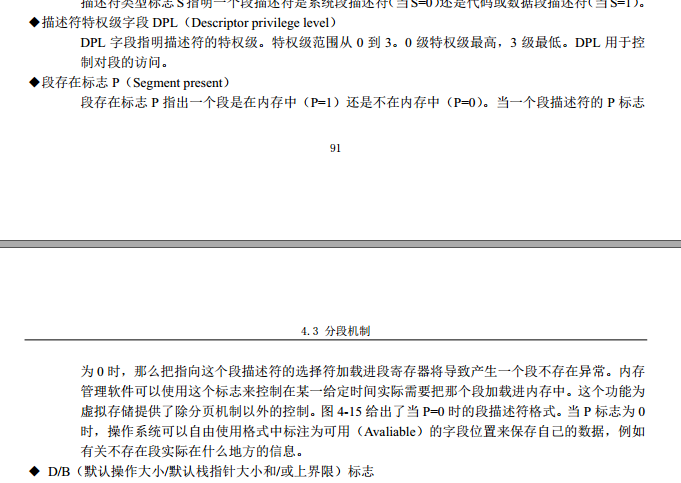
特征：GDT和IDT在整个系统中只有一张，而每个任务，都有自己的私有的一张局部描述符表LDT。用于记录本任务中的设计的各个代码段、数据段、和堆栈段、以及本任务使用的门描述符。GDT中包含系统中使用的代码段、数据段和堆栈段、特殊数据段描述符、以及所有任务局部描述符表LDT的描述符。

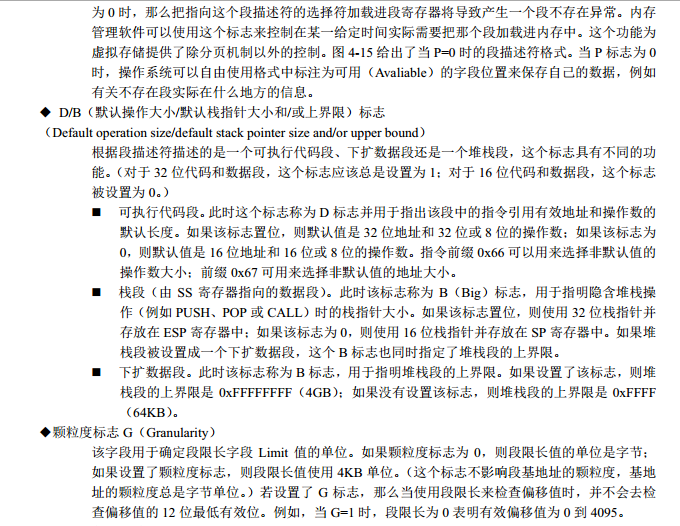
GDTR全局描述符寄存器：48位，高32位存放GDT基址，低16为存放GDT限长。  
LDTR局部描述符寄存器：16位，高13为存放LDT在GET中的索引值。

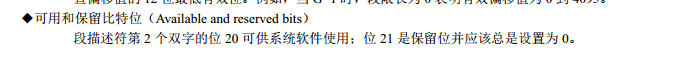
段描述符的结构图以及含义解释

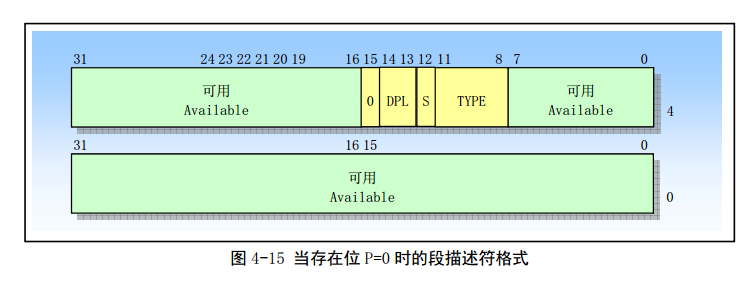
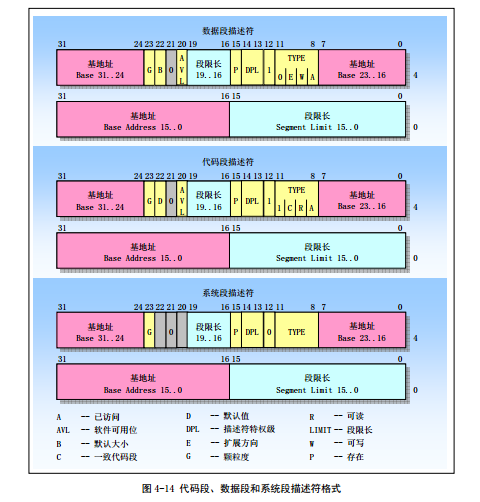






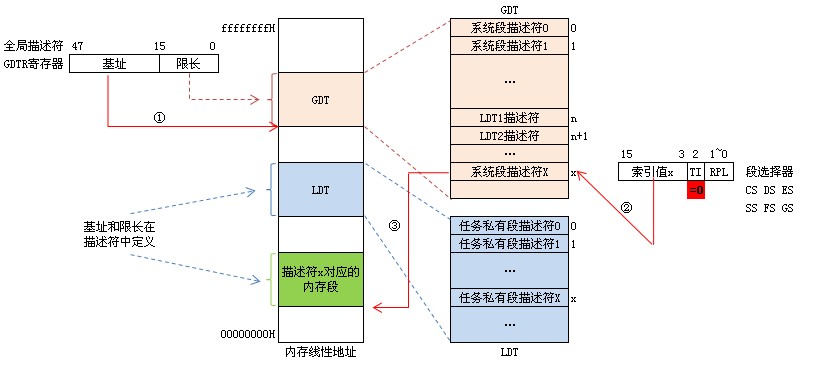






利用GDT进行访问的过程：

当 TI=0时表示段描述符在GDT中，如下图所示：① 先从GDTR寄存器中获得GDT基址。② 然后再GDT中以段选择器高13位位置索引值得到段描述符。③ 段描述符符包含段的基址、限长、优先级等各种属性，这就得到了段的起始地址（基址），再以基址加上偏移地址yyyyyyyy才得到最后的线性地址。

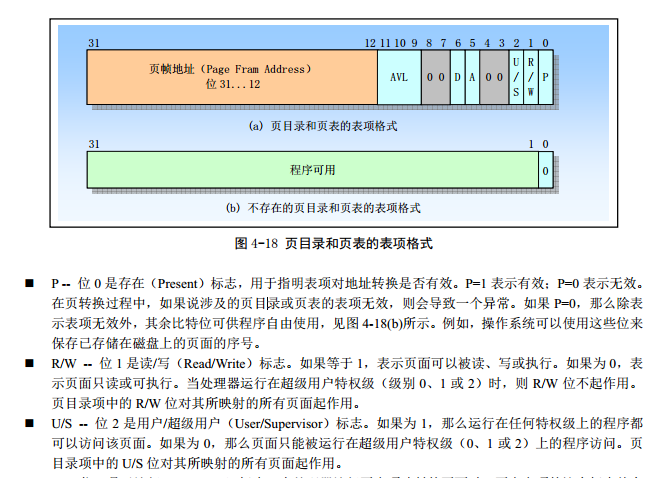


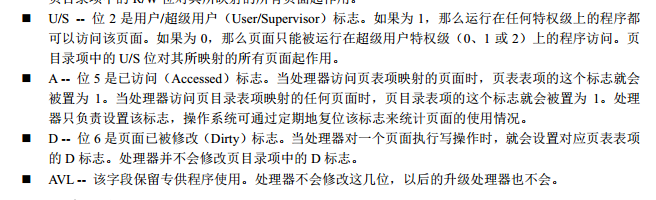
利用LDT进行访问的过程：

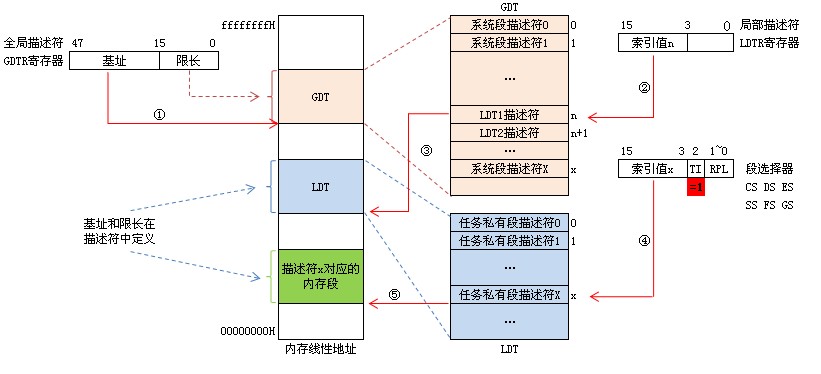
当TI=1时表示段描述符在LDT中，如下图所示：① 还是先从GDTR寄存器中获得GDT基址。② 从LDTR寄存器中获取LDT所在段的位置索引(LDTR高13位)。③ 以这个位置索引在GDT中得到LDT段描述符从而得到LDT段基址。④ 用段选择器高13位位置索引值从LDT段中得到段描述符。⑤ 段描述符符包含段的基址、限长、优先级等各种属性，这就得到了段的起始地址（基址），再以基址加上偏移地址yyyyyyyy才得到最后的线性地址。

页表项的格式：

页目录和页表的表项格式见下图，其中位31-12含有物理地址的高20位，用于定位物理地址空间中的一个页面（也成为页帧）的物理基地址。表项的低12位含有页属性信息。我们已经讨论过存在属性。这里简要说明其余属性的功能和用途。



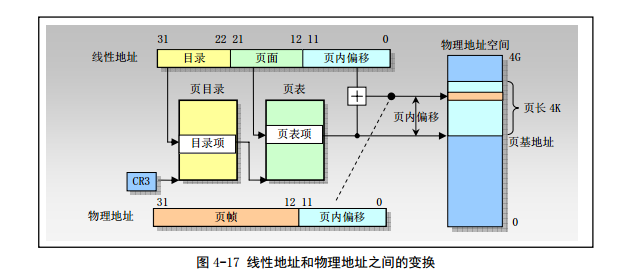




以上已经将逻辑地址转化为了线性地址，如果没有开启分页那么线性地址就是物理地址就直接进行访问了。但是如果开启了分页还需要下面的过程。

3.分页机制

我们已经得到了线性地址为了更加系统和方便的管理内存我们还启用了分页机制。



我们一共2的20次方个表项，而每个项占用4个字节所有一共占据4M对于本来内存就比较小的CPU来说基本上不可能，但是实际上我们只需要把我么需要使用的目录表页面定义下来，而没有使用的目录表不去定义；这样以来我们就只是对我们使用的内存进行分页。

而且页目录表中的每一个表象也有一个存在（Present）属性，类似于页表的表项。页目录表项的存在属性指明对应的二级页表是否存在。如果目录表项指明对应的二级页表存在。那么通过访问二级表查找过程第二步将如同上描述继续下去。如果存在位表明对应的二级表不存在，那么处理器就会产生一个异常来通知操作系统。页目录表项中的存在属性使得操作系统可以根据实际使用的线性地址范围来分配二级页表页面。