第二部分　深入探讨

第十一讲　CPU 工作模式与物理地址形成

一、CPU 及其工作模式

Intel 8086：1978 年，也称 16 位 CPU。

（1）内部存储单元（寄存器）为 16 位；

（2）数据总线 16 位；

（3）地址总线为 20 位，主存寻址范围 1MB；

（4）8086 的 16 位指令系统成为后来广泛应用的其他 80X86 的基本指令集。

Intel 80386：1985 年，32 位 CPU

（1）内部存储单元（寄存器）为 32 位；

（2）数据总线 32 位；

（3）地址总线为 32 位，主存寻址范围 4GB；

（4）32 位指令系统的结构，被 Intel 公司称为英特尔结构(IA)，作为后续 80X86 微处理器的标准。

（5）80386 提供了三种工作方式：

（A）实方式（实际地址）的操作相当于一个可进行 32 位快速运算的 8086（内部 32 位、外部总线 16 位数据、20 位地址，对应的程序为 16 位段的程序）

（B）保护方式（虚地址），是 80386 设计目标全部达到的工作方式，通过对程序使用的存储区采用分段、分页的存储管理机制，达到分级使用互不干扰的保护目的。能为每个任务提供一台虚拟处理器，使每个任务单独执行，快速切换。对应的程序是 32 位段程序。

（C）在保护方式下所提供的虚拟 8086 工作方式能同时模拟多个 8086 处理器。

1989 年 80486

1993 年 Pentium（586）。

在其后的几年里，相继推出了 **Pentium**，及多核 **CPU**，它们都继承了 **80386** 的 **32** 位指令系统，同时又新增了若干专用指令、浮点指令、整数和浮点多媒体指令，极大丰富了 **80X86** 的指令系统。

二、CPU 如何给出主存单元的物理地址 PA，以便得到指令或数据

分段管理的方法：

段寄存器保存起始首地址 + 段内偏移 EA 的总体策略

（如 CS：IP，二维逻辑地址）

1、实方式下 CPU 计算 PA 的方法

如何用 16 位信息给出 20 位 PA？

段寄存器都是 16 位的，如何表示 20 位的首地址？

例：10、 20、 30、 40、 50、 …、90

记住高位的数值即可（去掉后面公共的 0）：

1、 2、 3、 4、 5、 …、9

表示具体数值时，将高位数值后补零后加上个位：

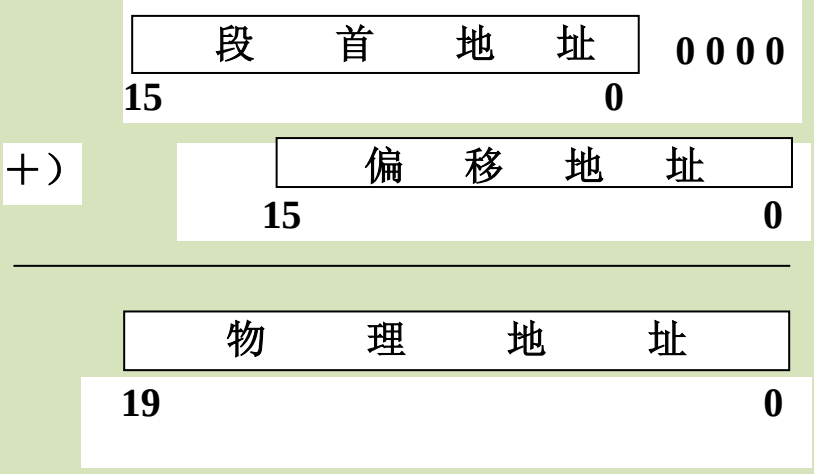
数值 34 = 高位 3 后补 0 后加上 4

= 高位 * 10 + 4

这样，16 位段寄存器只需保存 20 位首地址的高 16 位，低 4 位认为是去掉了 4 个二进制的 0，因此，具体存储单元的物理地址 PA

PA = 段寄存器保存的 16 位二进制数后补 4 个 0 + EA

（EA 只有 16 位，因此相加时，高 4 位为 0）



PA = （段寄存器）* 16 + EA

问：如果不改变（段寄存器），只改变 EA，则 PA 的范围？

问：设 PA=10015H，请问对应的段地址和偏移地址为多少？

知道了段寄存器的内容 和 EA 的值，就等于知道了实际的 PA。

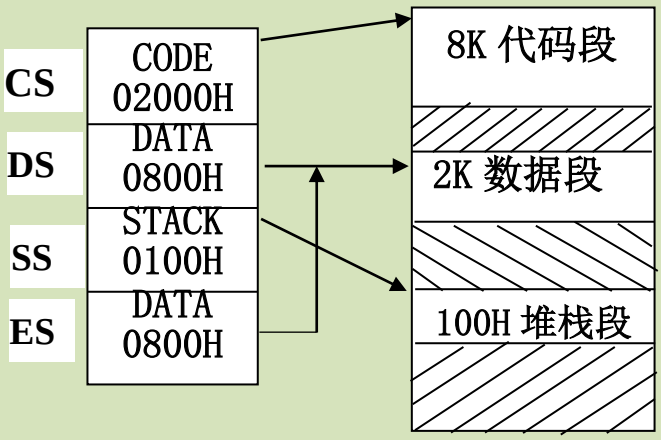
实际地址 》 实方式的真实含义。

分段的实际作用：

（1）实现了 16 位表示 20 位的地址；

（2）当程序和数据的大小<64KB 时，编制的程序可只关心 EA，而不用管它的起始地址在哪（便于程序在主存中任何位置运行）。（只要确定了车内的相互关系，不管车开到哪了，车内关系不变）。

(3)便于不同目的的程序或数据分开存放,使程序各部分的含义更加明确。



2、保护方式下 CPU 计算 PA 的方法

保护什么？（不谈保护问题时如何得到 PA ）

分清不同程序使用的存储区域，不允许随便使用别人的数据和代码。

必要条件：

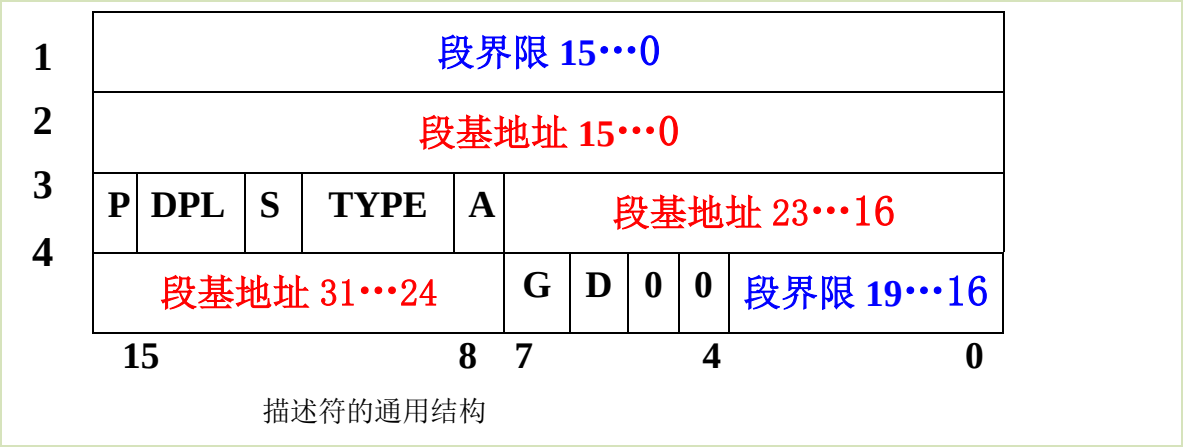
（1）要标记每段存储区的所有者或被使用的权限级别。

（2）要标记使用者是谁（权限级别）。

（3）中间环节：CPU 要去判断此次访问是否合法。

（1）如何标记存储区

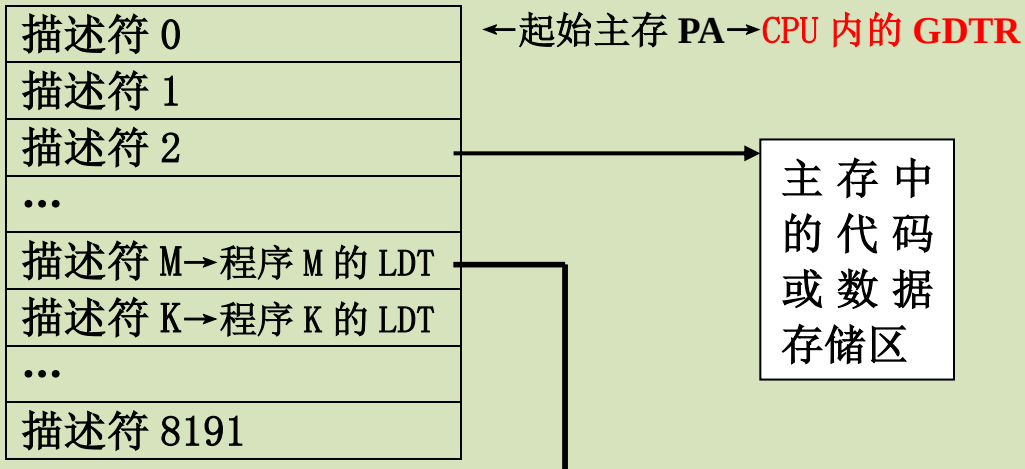
操作系统每次将某块存储区分给某程序使用时，用 8 个字节的描述符描述这段存储区的特征。



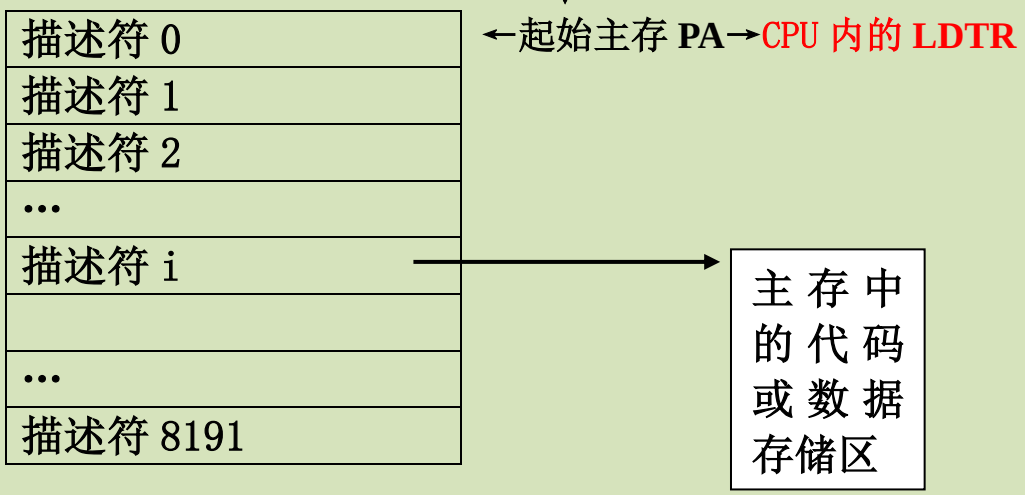
描述符的通用结构

P：该段在主存还是在磁盘；DPL：段的特权级；S：系统段还是用户程序段；TYPE：段的类型，只读只执行还是可读可写等；A：是否被访问过；G：段的计量单位是字节还是 4KB；D：16 位段还是 32 位段。

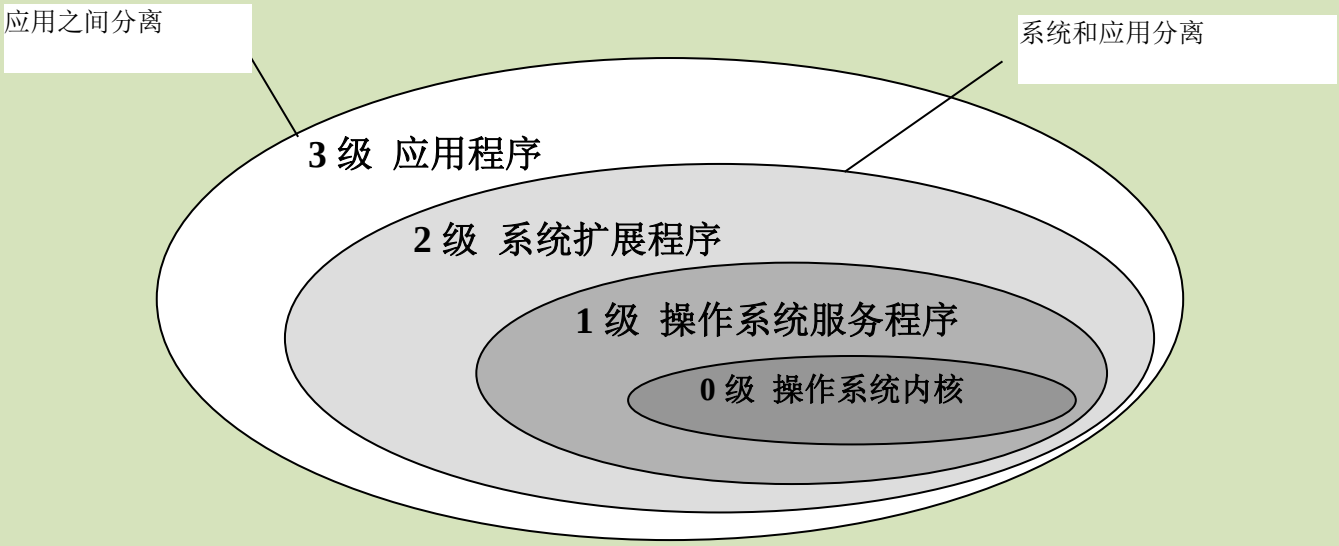
全局描述符表 GDT



局部描述符表 LDT



（2）如何标记使用者



程序的权限由程序所在的段决定，由当前程序正在使用的段寄存器内的段选择符反映。

段选择符→段寄存器中

描述符索引	TI	特权级
15 4	3	1 0

TI =0 从全局描述符表中找。
=1 从局部描述符表中找。

描述符索引 13 位。2^{13} = 8192

问：已知段寄存器中内容，如何得到描述符？如何得到段的实际段首址？

（3）中间执行环节：CPU 去判断

例如，执行：CS:00110054H　　MOV　AL，DS：[100H]

设：（CS）= 0000000000001 1 11 B，

（DS）= 0000000000002 0 01 B，

（a）程序权限与被访问数据段的权限的关系；

（b）（DS）指向的描述表中寻找描述符；　　（若 CS 从 11 改成 01 权限）

（c）对应描述符描述的段是否可以被访问；（类别、权限防止改 DS 权限的欺骗）

（d）从描述符 2 中提取段的首地址（32 位）；

（e）段的首地址（32 位）+32 位 EA（100H）= 线性地址，不分页时即为 PA

（f）送出 PA，选中存储单元，将内容读入 AL 中。

保护方式的实际含义：

程序知道的“段首址”实际上是一个代号，是一个虚拟的起始地址，CPU 通过查表转换才能获得真正的段首地址。

在这个由 CPU 完成的查表转换过程中，实现了访问权限的判断，达到了保护的目的。

地址表在系统软件设计中占有重要地位,其好处是调度和管理方便,对应的代码或数据存取区可以动态改变,有利于增减改，有利于实现特定的功能（如检查与保护）。缺点是容易被攻击。

地址表在一般软件设计中的好处：有利于实现存在大量分支情况的程序的设计，对应的间接的转移功能有利于实现反跟踪（尤其是针对静态分析的反跟踪）。

三、程序地址表与间接转移/调用

问：DOS 系统功能调用大约有 140 个子功能，“INT 21H”被执行后具体执行什么功能的程序代码是通过判断 AH 的值来确定的。试想象一下 INT 21H 是如何来实现这一判断过程的？

段间间接调用

格式：CALL DWORD PTR OPD 或 CALL FAR PTR OPD （16 位）

CALL FWORD PTR OPD 或 CALL FAR PTR OPD （32 位）

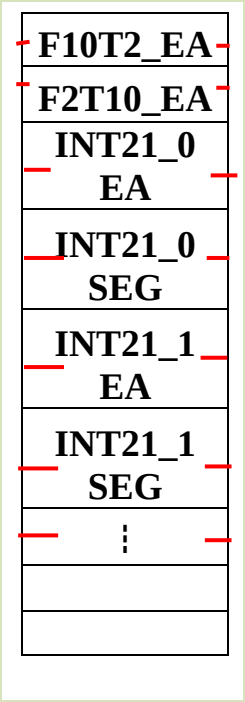
(OPD)→IP /EIP

(OPD+2/4)→CS

间接调用主要用于：当多个子程序入口地址组成地址表时，可用间接寻址方式确定转入子程序的入口地址。

间接调用入口地址表 例：

```
DATA  SEGMENT  USE16
  PROC_TAB1  DW  F10T2      ;子程序名、标号等，NEAR 类型。
             DW  F2T10
  PROC_TAB2  DD  INT21_0    ;子程序名、标号等，FAR 类型。
             DD  INT21_1    ;
             DD  INT21_2
             DD  INT21_3
             DD  INT21_4
          ..........
          ┆
    LEA   SI，PROC_TAB1
    CALL  NEAR  PTR  2[SI]  ;调用 F2T10
          ┆
    LEA   EBX，PROC_TAB2
    MOV   EDX，0
    MOV   DL，AH        ；AH=功能号 n （n=0 ,1, 2 ~）
    CALL  FAR   PTR  [4*EDX+EBX]   ; 4*n + PROC_TAB2 的 EA
或  CALL  DWORD  PTR  [4*EDX+EBX]
          ┆
       INT21_1   PROC  FAR
                 过程体 （程序段）
```



```
INT21_1   ENDP
INT21_2    PROC   FAR
                过程体 （程序段）
INT21_2   ENDP
   ⋮
F10T2     PROC   NEAR
                过程体 （程序段）
F10T2     ENDP
```

三、汇编过程中变量、段、段寄存器之间的翻译关系。

例：将 BUF1 中的字符传送到 BUF2 的字节缓冲区中。

```
DATA   SEGMENT   USE16
    BUF1     DB     'A'
    BUF2     DB     0
DATA     ENDS
CODE     SEGMENT   USE16
         ASSUME  DS：DATA，ES：DATA，CS：CODE，SS：STACK
START：MOV   AX，DATA
       MOV   DS，AX
       MOV   ES，AX
       MOV   AL，BUF1
       MOV   BUF2，AL

    ASSUME   DS：CODE
       MOV   AL，BUF1
       MOV   BUF2，AL
       MOV   AH，4CH
       INT    21H
CODE     ENDS
       END   START
```

问：操作数汇编成：

```
MOV    AL,DS:[0] / ES:[0]
MOV    DS:[1] / ES:[1],AL
```

同一语句中：当多个段寄存器对同一个段名假定关联时，变量的段关联优先次序从高到低：DS、SS、ES、CS。

不同的语句中：后面的假定关联语句覆盖前面的关联。