北京理工大学计算机学院
School of Computer Science and Technology,BIT

第二章

微处理器管理模式

提纲

- 微处理器基本结构
- CPU工作模式
- 寄存器
- 内存管理
- 任务
- 保护

内存管理

- 80368以上的CPU允许使用虚拟存储器，除了有一个速度较快地、容量较小的主存储器（内存），还有一个速度较慢，但容量很大的外部存储器。二者通过存贮器管理机制，有机地、灵活地结合在一起。
- 存储体系
  - CPU/Cache/内存/虚拟内存/外存
- 实模式下的存储器分段管理

保护模式下存储器分段管理

- CS:EIP为例
  - CS中存放了一个16位的段选择符
  - EIP是32位偏移量
  - 16位段选择符加上32位偏移量，总共是48位，其中段选择符的2位RPL与虚拟地址的转换无关，因此可以认为虚拟地址是46位的，段选择符的Index和TI占14位，偏移量为32位，
  - 虚拟地址空间为2^{46}B=64TB，

保护模式下存储器分段管理

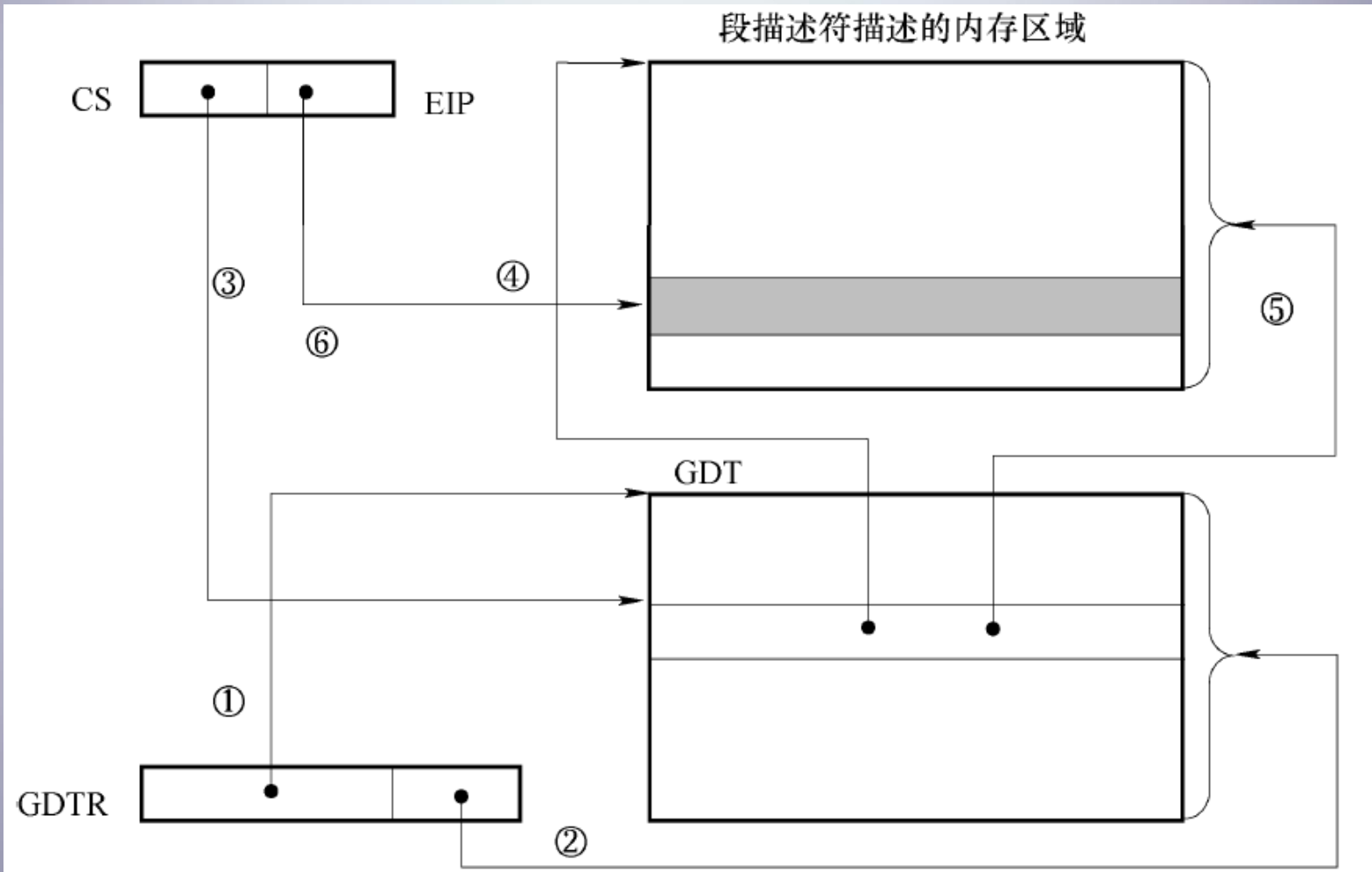


图 2－14　全局存储空间中寻址示例

段描述符

- 段描述符
  - 段描述符用于描述代码段、数据段和堆栈段。
  - 段限长指出了一个段的最后一个字节的偏移地址。
  - 段描述符位于GDT或LDT中，占8字节（64位），由以下几个部分组成：段基址（32位）、限长（20位）、访问权限（8位）和属性（4位）。

段描述符

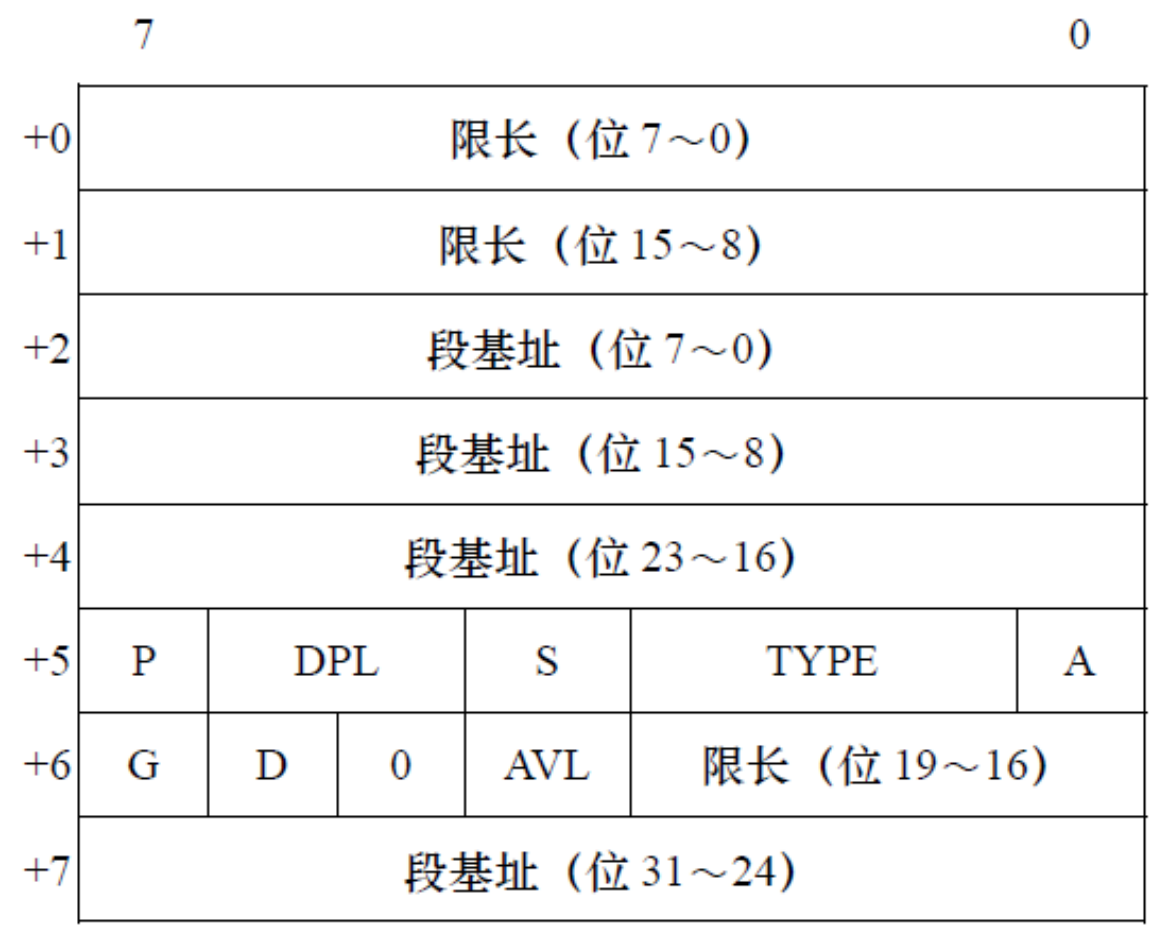


图 2－15　段描述符的格式

段描述符

- P（Present）存在位。=1时表示该段已装入内存；=0时表示该段没有在内存中，访问这个段会产生段异常。操作系统将某个段从物理内存中交换到磁盘时，设置此位为0。
- DPL（Descriptor Privilege Level）描述符特权级。说明这个段的特权级，取值为0～3。
- S（System）描述符类型位。=1时，这个段为代码段、数据段或堆栈段；=0时，为系统段描述符。
- E（Executable）可执行位，用来区分代码段和数据段。S=1且E=1时，这是一个代码段，可执行。S=1且E=0时，这是一个数据段或堆栈段，不可执行。E=0时，后面的两位为ED和W；若E=1时，后面的两位为C和R。

S=1 且 E=0 时:	P	DPL	1	E	ED	W	A
S=1 且 E=1 时:	P	DPL	1	E	C	R	A

图 2-16　访问权限字节的格式

权限字节

- ED（Expansion Direction）扩展方向位（对数据段或堆栈段）。=0时，段向上扩展（从低地址向高地址扩展），偏移量小于等于限长。=1时，段向下扩展（从高地址向低地址扩展），偏移量必须大于限长。
- W（Writeable）写允许位（对数据段或堆栈段）。=0时，不允许对这个数据段写入；=1时，允许对这个数据段写入。对数据段进行读操作总是被允许的。
- C（Conforming）一致位（对代码段）。=0时，这个段不是一致代码段；=1时，这个段是一致代码段。
- R（Readable）读允许位（对代码段）。=0时，不允许读这个段的内容；=1时，允许读这个段的内容。对于代码段进行写操作总是被禁止的。
- A（Accessed）访问位。=1表示段已被访问（使用）过；=0表示段未被访问过。

属性位

- G（Granularity）粒度位。G＝1时，限长以页为单位；G＝0时，限长以字节为单位。
- D（Default Operation Size）默认操作数宽度。D＝1时，为32位数据操作段；D＝0时，为16位数据操作段。
- AVL（Available field）可用位。这一位保留给操作系统或应用程序来使用。

- 例2.5 设段基址为0CD310000H，段界限为001FFH。G位分别为0和为1时，求段的起始地址和段的结束地址。

描述符示例

- 例2.6 段描述符的实际应用。

段的属性

- 假设EBX指向一个有效的数据单元，但下面指令执行时会产生异常：
  - MOV　CS:[EBX], EAX
  - 因为001BH所代表的段是一个代码段（S=0，E=1），不允许写入。

段描述符高速缓存

- Cache中，段选择符不变则描述符不变

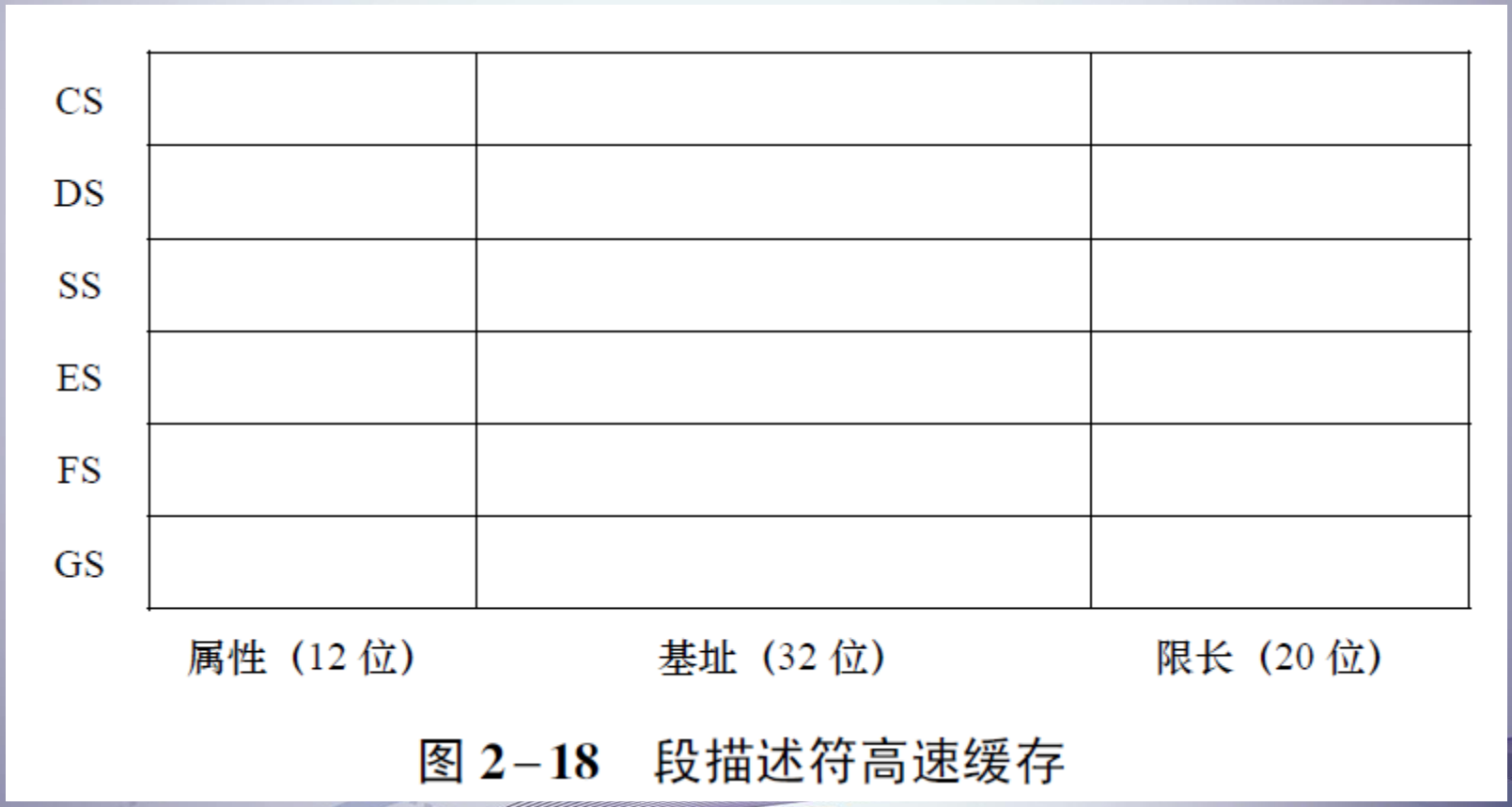
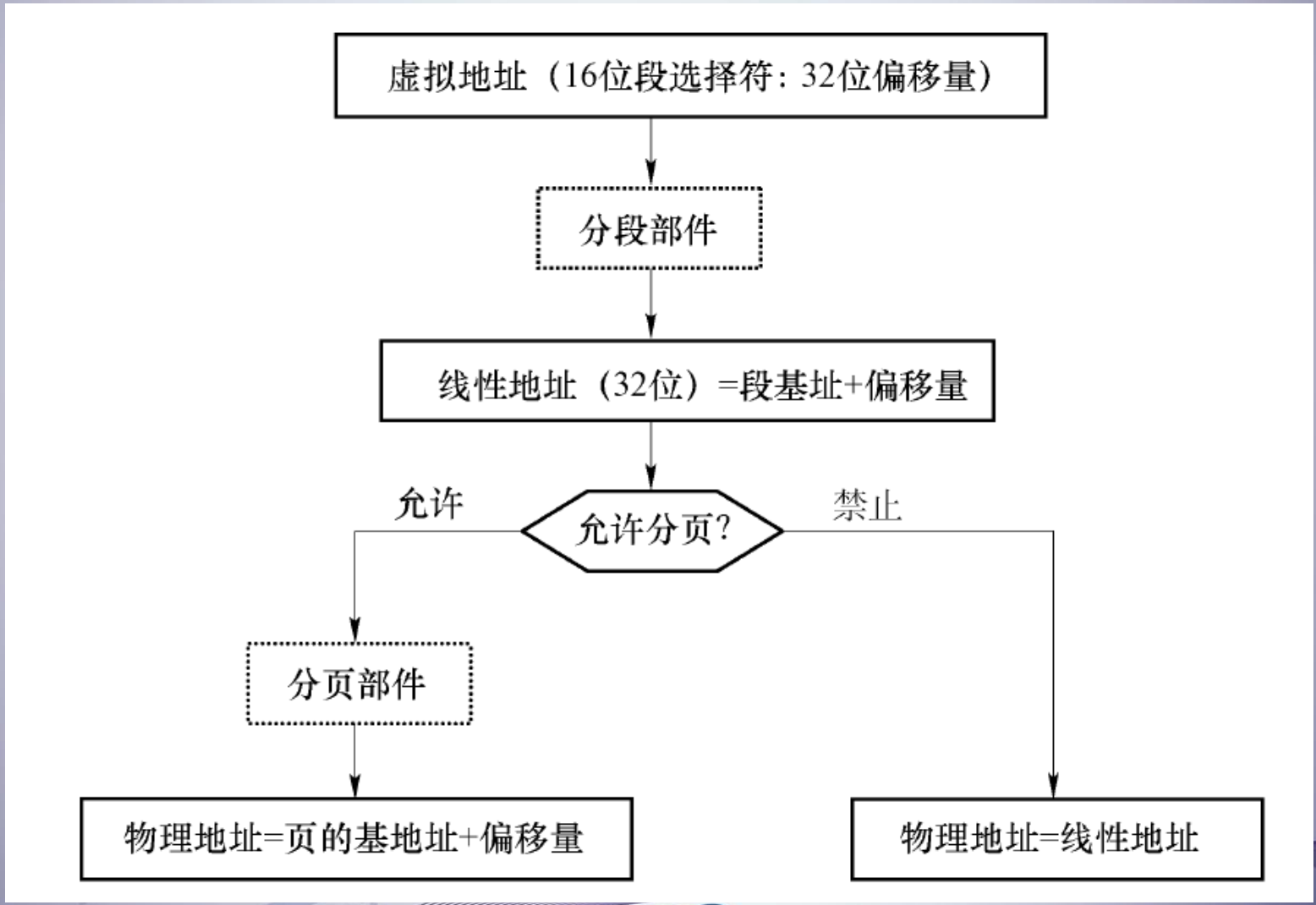


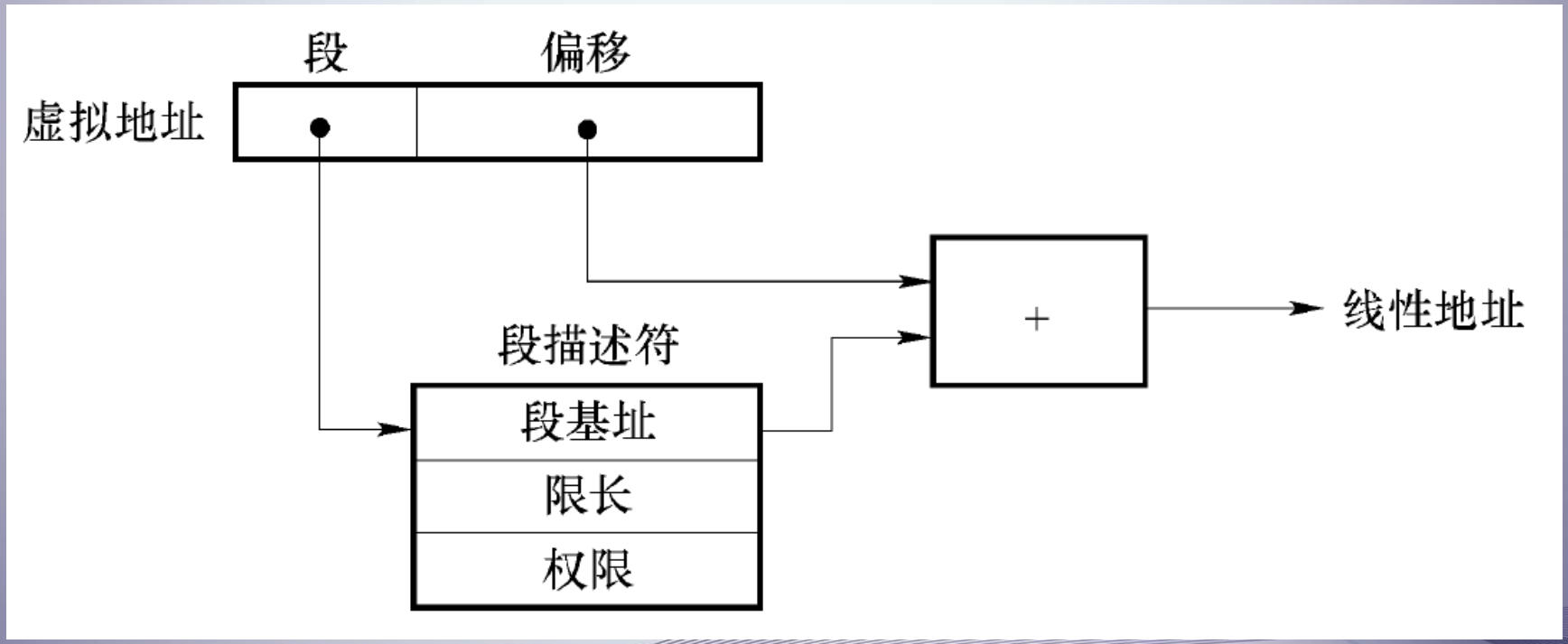
图 2-18　段描述符高速缓存

虚拟地址到物理地址


虚拟地址（16位段选择符：32位偏移量）
分段部件
线性地址（32位）=段基址+偏移量
允许
允许分页?
禁止
分页部件
物理地址=页的基地址+偏移量
物理地址=线性地址

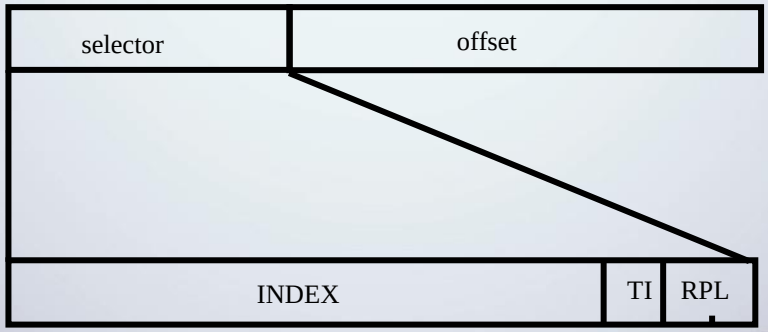
段式地址转换

- 如不支持分页，则线性地址==物理地址



分段内存管理

- 虚拟地址、线性地址、物理地址
  - 虚拟地址就是程序指令中使用的地址（也称逻辑地址），它由段和偏移两个部分组成。段选择符的Index和TI占14位，偏移量为32位。如DS:[EBX]就是一个虚拟地址。



线性地址

CPU的分段部件将虚拟地址转换为线性地址。

对于一个虚拟地址，线性地址是段的基地址再加上偏移量。段的基地址保存在段描述符中。

段基址和偏移量都是32位的，所以线性地址是32位。线性地址空间的范围是2^{32}=4GB，即00000000_{16}-$FFFFFFFF_{16}$。

物理地址

物理地址是微处理器引脚输出的地址信号。

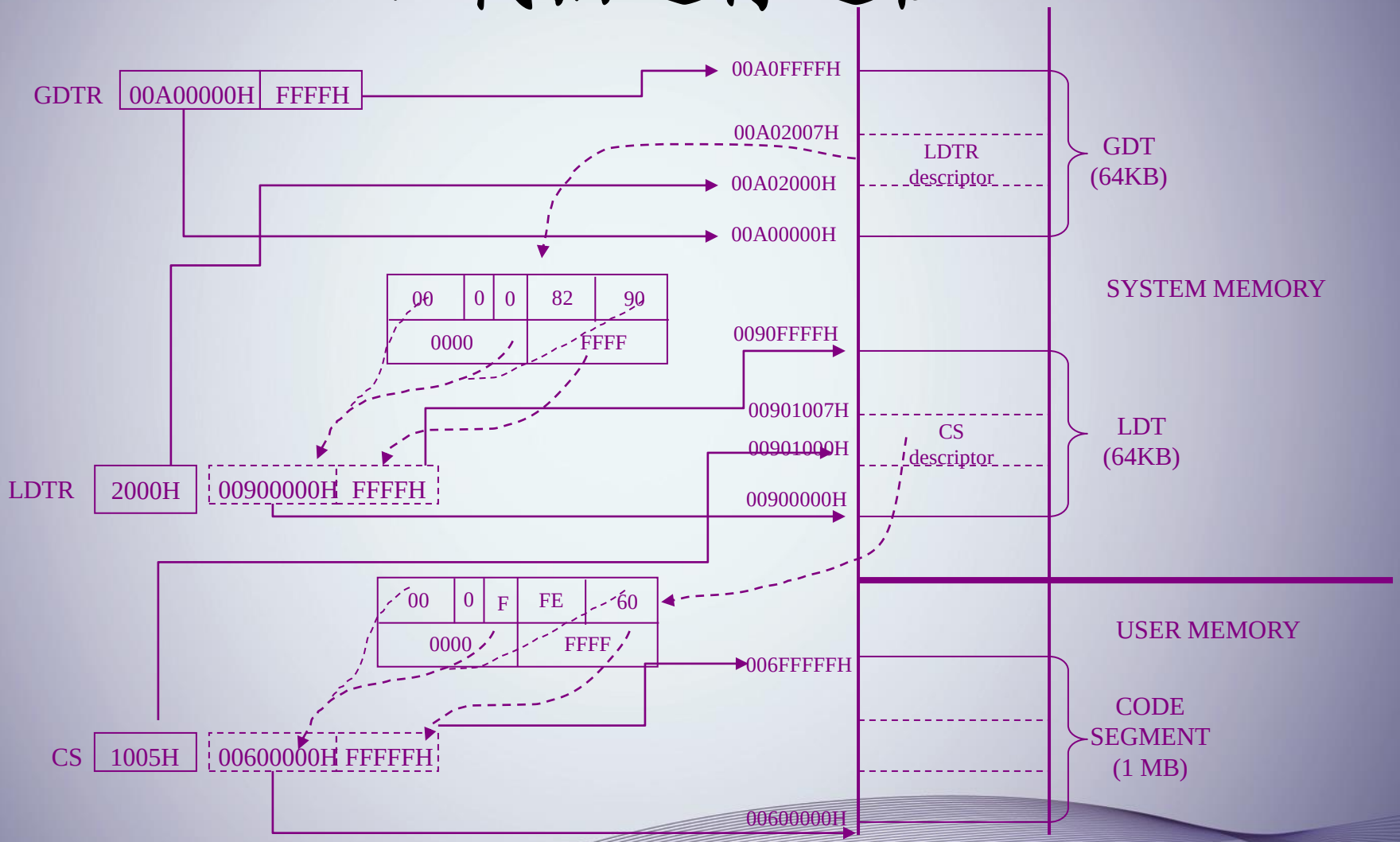
同一个线性地址得到的物理地址可以不同。

CPU的分页部件将线性地址转换为物理地址。如果禁止CPU的分页功能，线性地址就直接作为物理地址。

地址

- 同一个虚拟地址可能得到不同的线形地址
  - 任务1和任务2对同一个虚拟地址0047H:00002000H得到的线性地址可能是不同的。
  - 对于同一个局部段选择符,LDT可对不同的任务设置不同的段基址。

加载描述符过程



CS=1003H, 其过程?

分页内存管理

- 保护模式下的CPU支持分页机制，并且分页管理是在分段管理机制的基础上工作，它将分段管理机制得到的线性地址转换为物理地址。
- 使用分页机制的好处在于，它可以把每个活动任务当前所必需的少量页面放在内存中，而不必将整个段调入内存，从而提高了内存的使用效率。

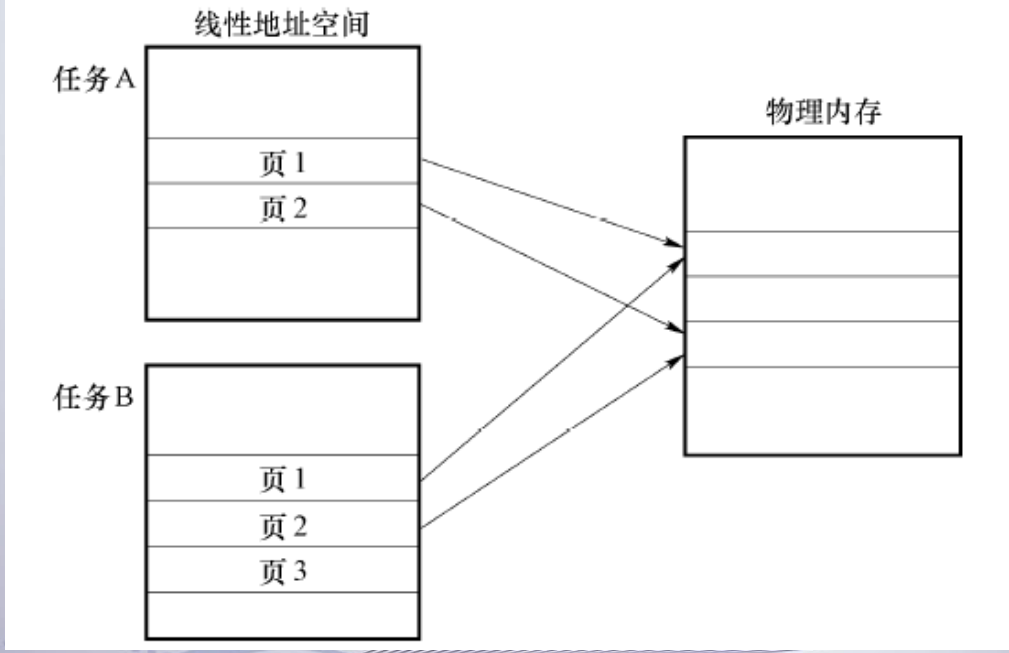
分页内存管理

- 分页
  - 所有页的长度固定为4KB，页与页之间也没有重叠。CPU将4GB的线性地址空间划分成2^{20}页。

  - 在Windows/Linux操作系统中，将段式内存管理和页式内存管理结合起来，主要是依赖页式内存管理来调度内存。

分页内存管理

- 线性地址到物理地址的映射
  - 线性地址按页（4KB）为单位映射到物理地址。
  - 每一个线性页面都映射到一个物理页面上。



分页内存管理

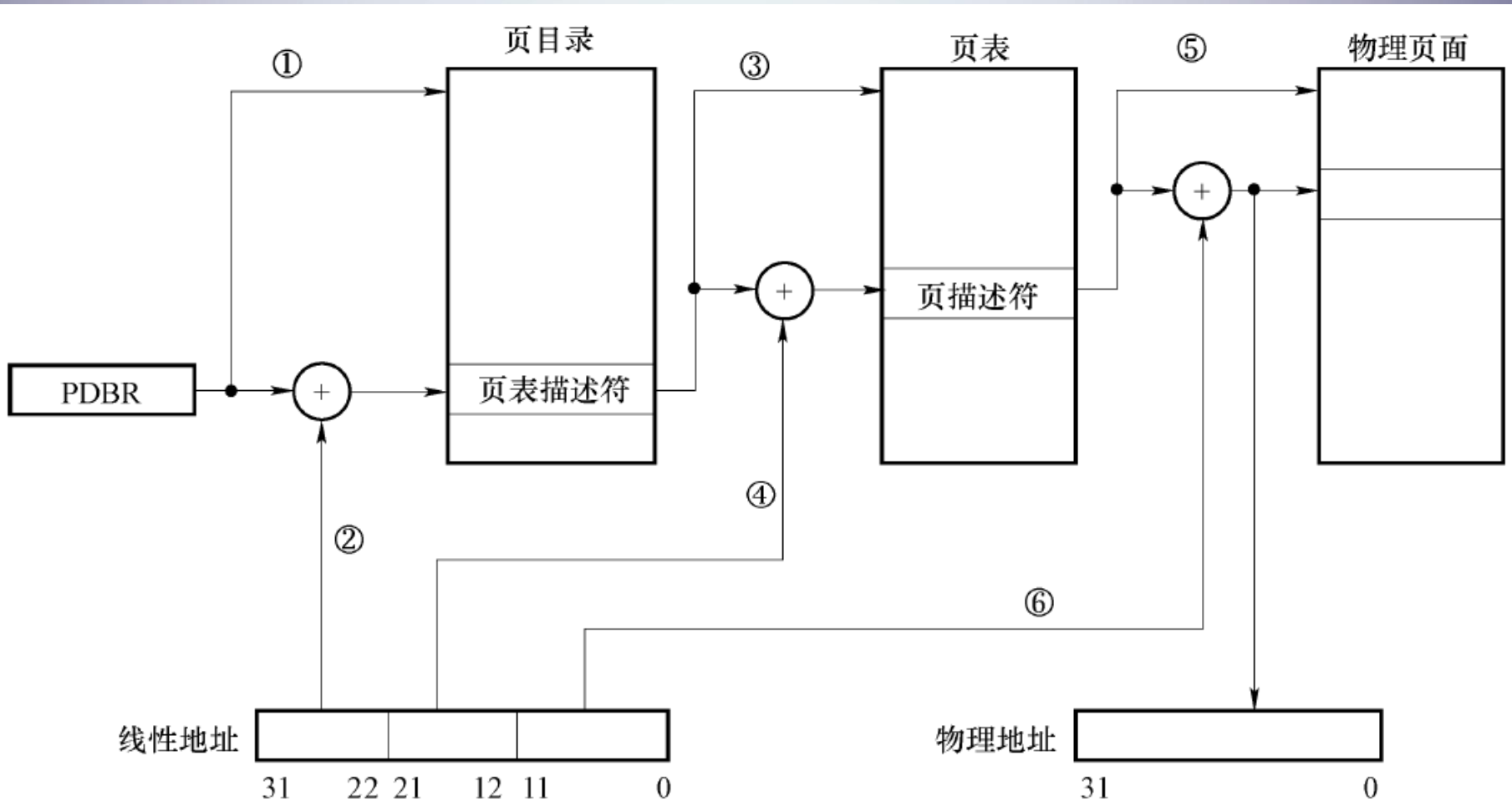
32位线性地址被划分为3个部分:

页目录索引　页表索引　　页面索引

其中第1项是对页目录（Page Directory）的索引，第2项是对页表（Page Tables）的索引，第3项是线性地址在页面内的偏移。

页目录表　　页表　　页面/页帧

分页内存管理


页目录
①
③
页表
⑤
物理页面
PDBR
页表描述符
页描述符
④
②
⑥
线性地址
31
22
21
12
11
0
物理地址
31
0

分页内存管理

- 页目录的地址由CR3的最高20位决定，CR3又被称做页目录基址寄存器PDBR（Page Directory Base Register），CR3的低12位＝000H。页目录大小为4KB，由1024个页表描述符组成，每个页表描述符占4个字节。
- 线性地址中高10位为页目录索引，页目录基地址加上页目录索引乘以4获得的地址指向页目录表中一个页表描述符。
- 页表描述符的高20位给出了页表的基地址。页表同样占4KB，由1024个页描述符组成，每个页描述符占4字节。
- 线性地址中的页表索引（10位），指示了被访问的页在页表中的序号。根据页表基地址加上10位页表索引乘以4指向页表中的一个页描述符。
- 页描述符的高20位给出了物理页面的基地址的高位20位。
- 物理页面的基地址再加上线性地址中12位字节的页内偏移量，得到物理地址。

分页内存管理

- 片内转换检测缓冲器TLB
  - 每次内存操作都需要将线性地址转换为物理地址，转换过程中需要访问页目录表和页表来取得页表描述符和页描述符。
  - 为了提高转换效率，CPU内部设置了片内转换检测缓冲器TLB（Translation Lookaside Buffer），其中保存了32个页描述符，它们都是最近使用过的。

分页内存管理

- 页表项
  - 页表项就是在分页转换时用到的页表描述符和页描述符，都是32位，页表项格式如图2-21。

31 ~ 12	11 ~ 9	8	7	6	5	4	3	2	1	0
高 20 位基地址	AVL	0	0	D	A	0	0	U/S	R/W	P

图 2-24　页表项的格式

分页内存管理

- 页目录、页表和物理页的基地址的低12位全部为0，定位在页的边界上。
- 页表项的低12位提供保护功能和统计信息。U/S位、R/W位、P位实现页保护机制。A、D、AVL访问控制。

表 2-4　U/S 位和 R/W 位对页面的保护

页表项权限	用户程序	系统程序	用途
U/S=0, R/W=0	不可读写	可读写	系统页面
U/S=0, R/W=1	不可读写	可读写	系统页面
U/S=1, R/W=0	只能读	可读写	代码页面
U/S=1, R/W=1	可读写	可读写	数据/堆栈页面

任务

多任务环境，是指其硬件允许软件系统中存在多个任务，并能够以分时的方式使各程序轮流执行。

当运行一个应用程序后，操作系统就为这个程序创建一个任务。

在保护模式下，在任何时刻都有一个当前任务，当前任务由TR寄存器指定，CPU在这个任务的环境下执行。因此，系统中至少存在一个任务。

TR

- 任务寄存器TR
  - 任务寄存器TR在保护模式的任务切换机制中使用。
  - TR是16位的选择符，其内容为索引值，它选中的是TSS描述符。
  - TR的初值由软件装入，当执行任务切换指令时TR的内容自动修改。

TSS

- 在多任务环境下，每个任务都有属于自己的任务状态段TSS，TSS中包含启动任务所必需的信息。
- 任务状态段TSS在存储器的基地址和限长（大小）由TSS描述符指出。
- TSS描述符放在全局描述符表GDT中，TR内容为选择符，它指出TSS描述符在GDT中的顺序号。
- 由描述符说明各TSS的位置和限长。

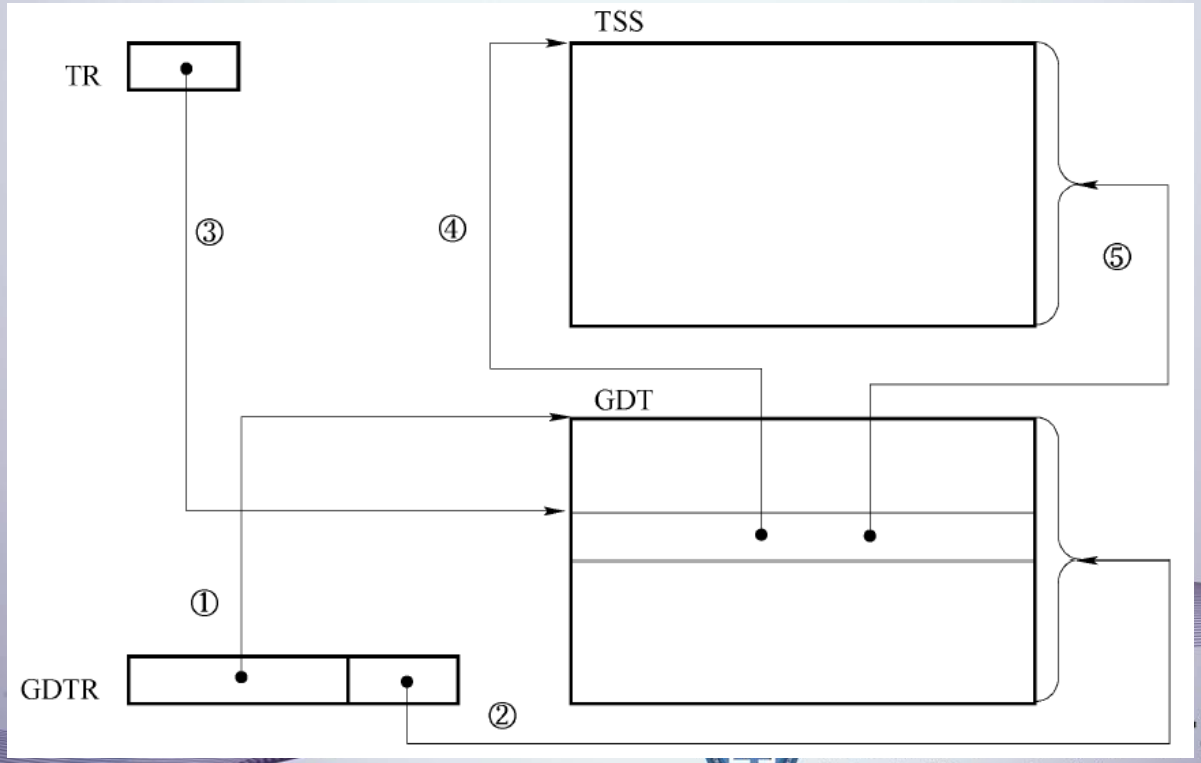
由任务寄存器TR取得TSS的过程

①和②由GDTR确定了GDT表在存储器中的位置和限长。

TR选择符中包含了TSS描述符在GDT中的索引。

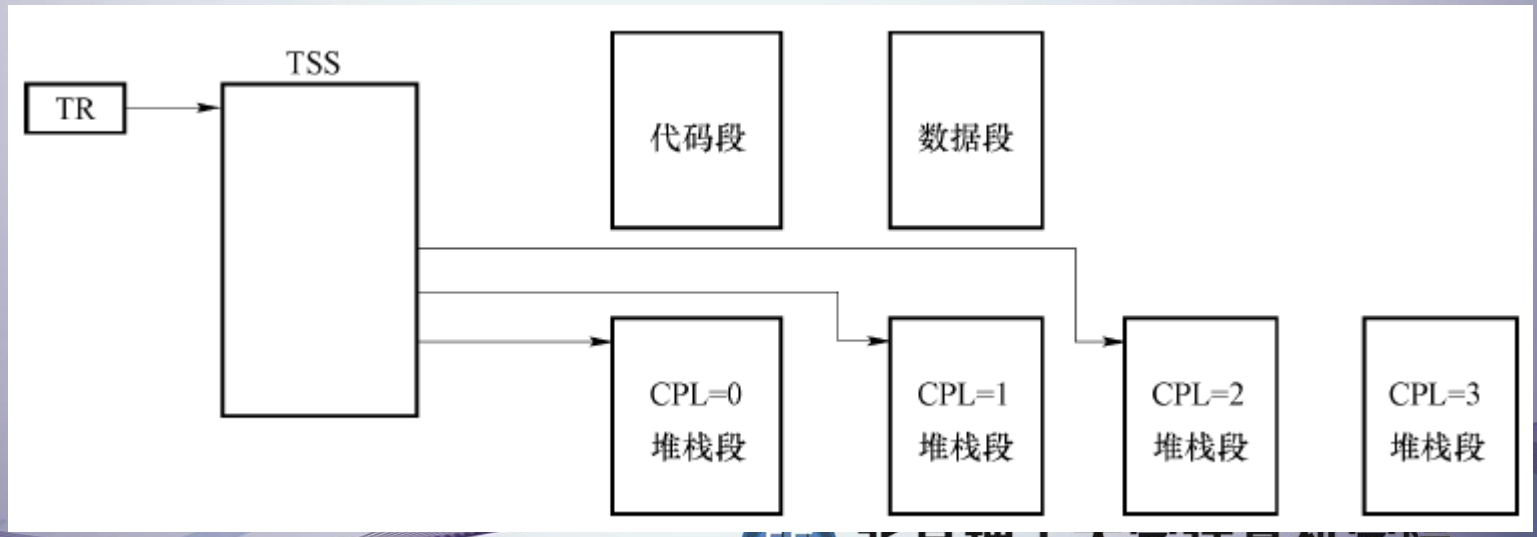
③步依据TR在GDT中取出TSS描述符。

④和⑤中，在TSS描述符中取得TSS的基址和限长。



任务执行环境

每个任务都由两个部分组成：任务执行环境TES（Task Executation Space）和任务状态段TSS（Task State Segment）。任务执行环境包括一个代码段、堆栈段和数据段等，任务在每一个特权级上执行时都有一个堆栈段。



任务状态段

- 104个字节

31 16	15 0	偏移
0000 0000 0000 0000	链接字段	+00
ESP0		+04
0000 0000 0000 0000	SS0	+08
ESP1		+0C
0000 0000 0000 0000	SS1	+10
ESP2		+14
0000 0000 0000 0000	SS2	+18
CR3		+1C
EIP		+20
EFLAGS		+24
EAX		+28
ECX		+2C
EDX		+30
EBX		+34
ESP		+38
EBP		+3C
ESI		+40
EDI		+44
0000 0000 0000 0000	ES	+48
0000 0000 0000 0000	CS	+4C
0000 0000 0000 0000	SS	+50
0000 0000 0000 0000	DS	+54
0000 0000 0000 0000	FS	+58
0000 0000 0000 0000	GS	+5C
0000 0000 0000 0000	LDTR	+60
I/O 许可位图偏移	T	+64

任务状态段

- 任务状态段
- TSS中保存了任务的各种状态信息。任务状态段描述符（即TSS描述符）描述某个任务状态段，规定了任务状态段的基地址和任务状态段的大小等信息。

任务状态段

- 内存堆栈指针
- 4个堆栈对应于4个特权级，0/1/2/3

- 早期CPU通过自动保存原来的硬件上下文，装入新的硬件上下文来执行硬件上下文切换。
- 软件执行进程切换，执行效率高。进程硬件上下文的一部分存放在TSS段，而剩余部分存放在内核态堆栈中。

门（Gate）

- 一种转换机构，可以实现不同特权级别之间的控制传送。
- 调用门、任务门、中断门、陷阱门。
- 其中调用门用于控制传送，改变任务或者程序的特权级别；任务门像个开关一样，用来执行任务切换；中断门和陷阱门用来指出中断服务程序的入口地址。

门描述符

- 系统描述符
- 表2.5
  - 描述符类型
- 段选择符

	7							0
+0	偏移（位 7～0）							
+1	偏移（位 15～8）							
+2	段选择符（位 7～0）							
+3	段选择符（位 15～8）							
+4	0	0	0	参数计数值（5 位）				
+5	P	DPL			S＝0	TYPE（4 位）		
+6	偏移（位 23～16）							
+7	偏移（位 31～24）							

调用门

- 类型为4、C
- 由低特权级到高特权级的间接控制转移
- 参数计数值
  - 传递的参数个数
  - 从主程序堆栈复制到子程序堆栈
  - 大小：个数*4（32位环境）

调用门

- X：选择符，Y：不起作用

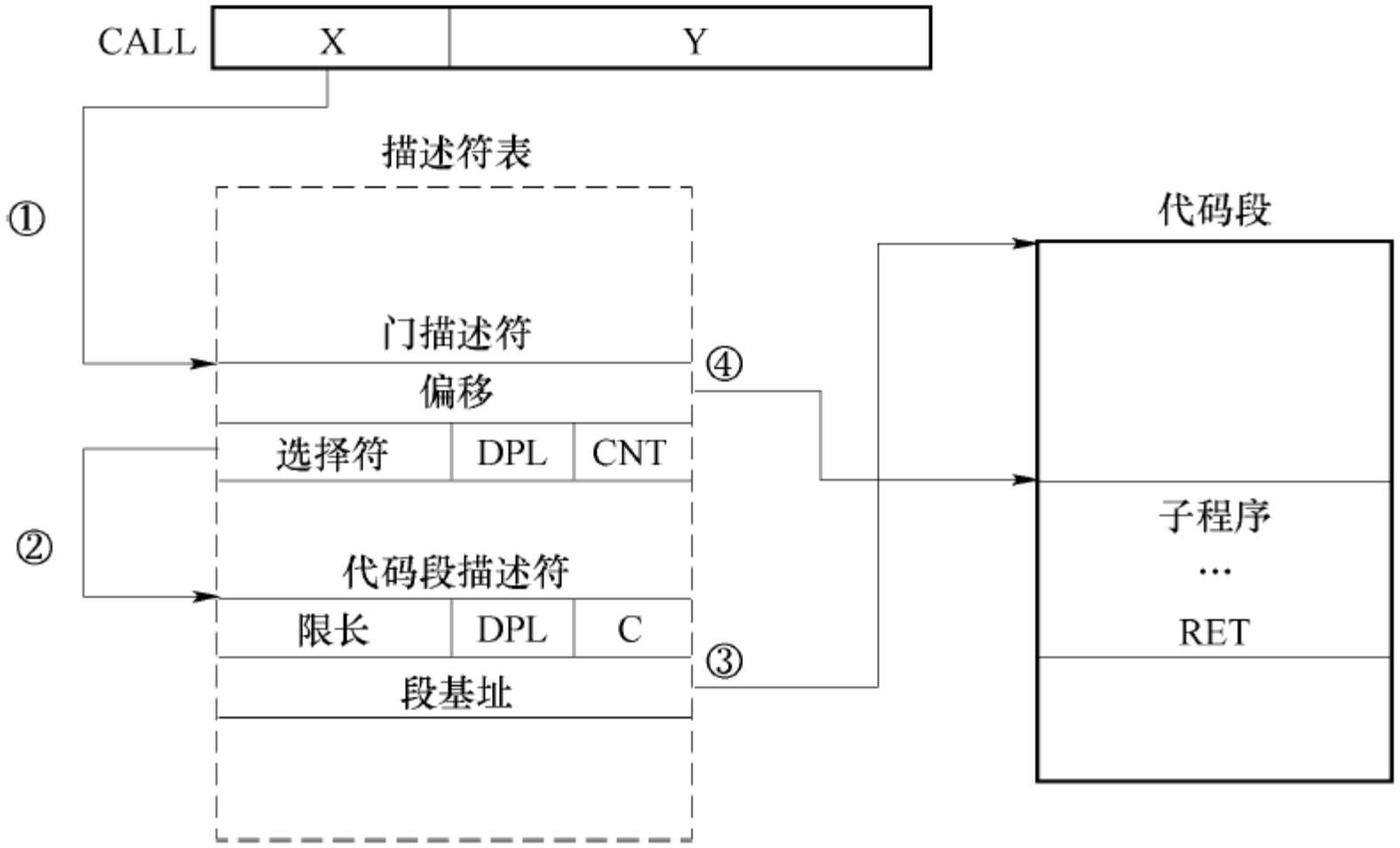


图 2－33　通过调用门转移到更高特权级

任务门

- 选择符指示TSS

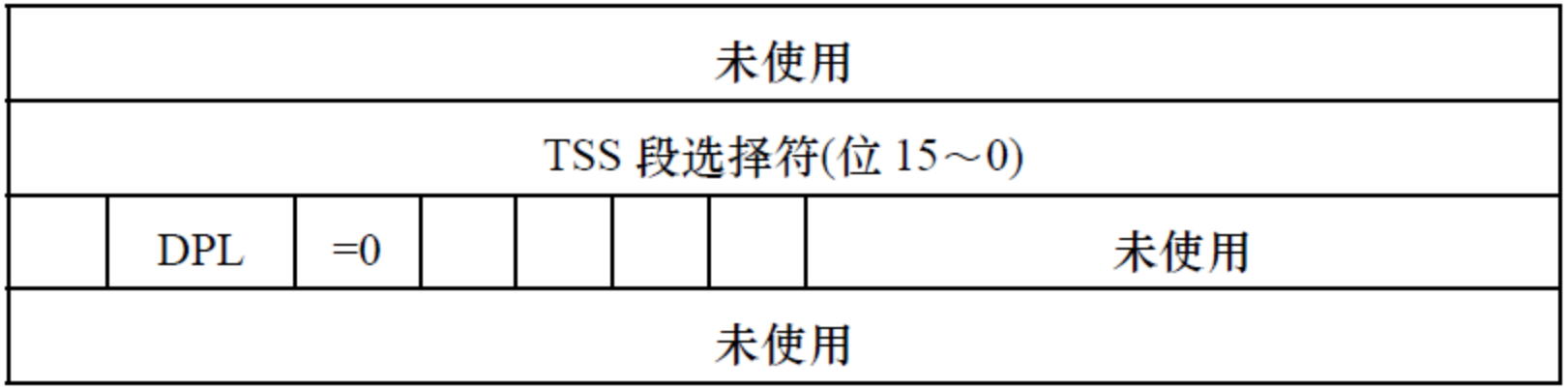


图 2－34　任务门描述符的格式

任务切换

- 四种情况
  - 执行远程JMP 或者CALL 指令，选择了GDT 中的TSS 描述符。
  - 执行远程JMP 或者CALL 指令，从GDT 或者LDT 中选择了任务门。
  - 发生了中断或异常，中断向量选择了IDT 中的任务门。
  - 当FLAGS 中的NT=1 时，执行IRET 指令，目的任务选择符在执行IRET 任务的TSS链接域中。

任务切换过程

直接任务切换

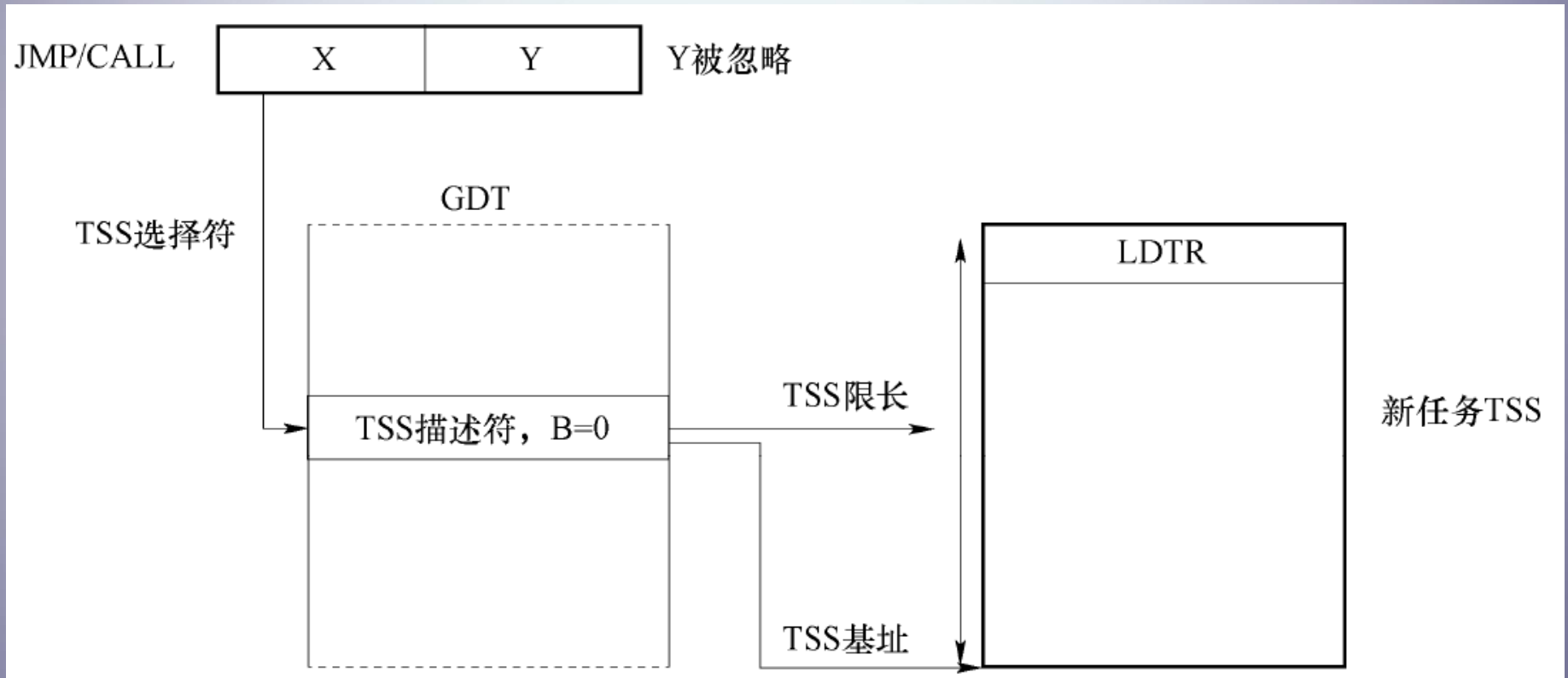


图 2－35　直接任务切换

任务切换过程

间接任务切换

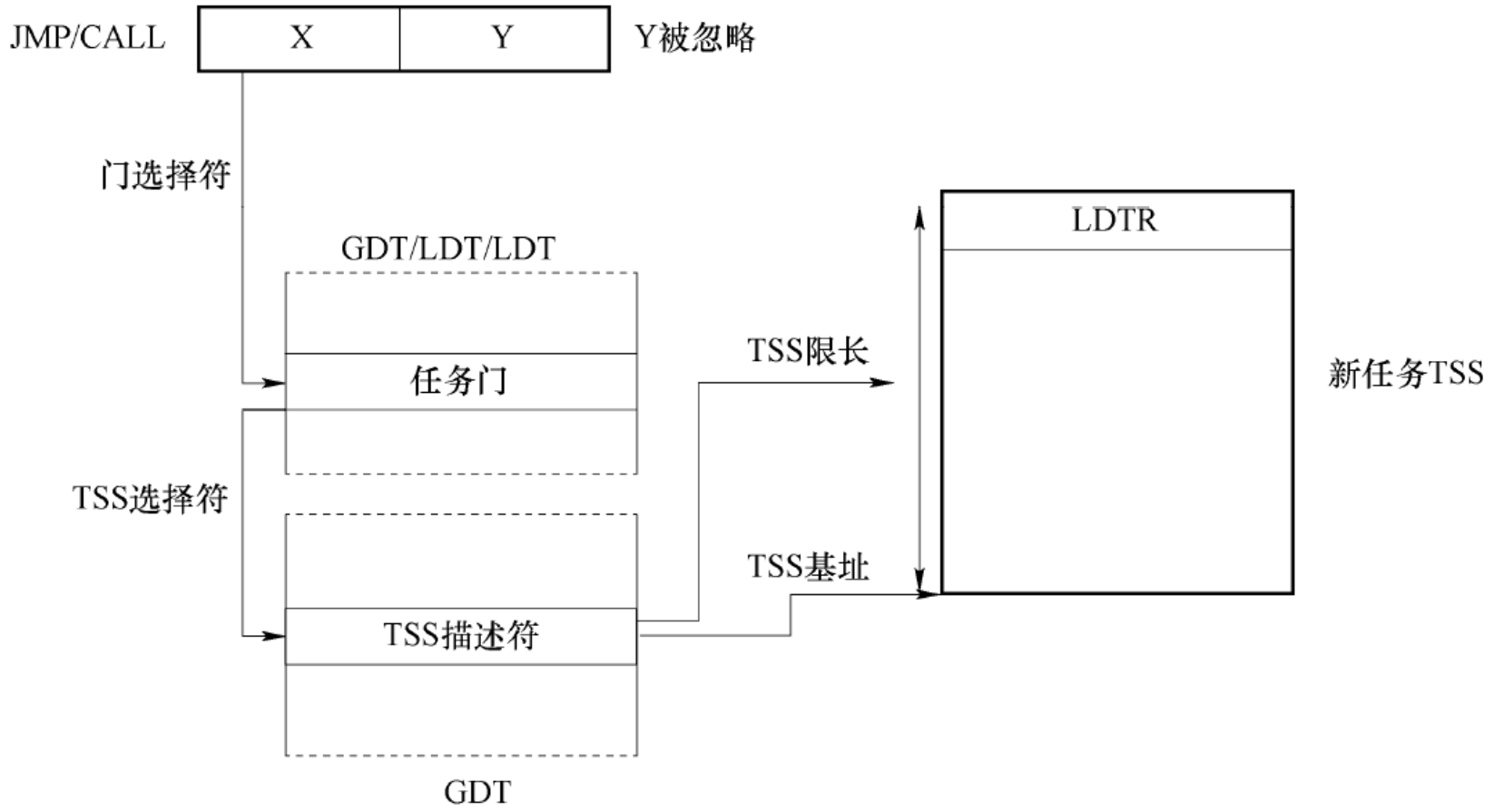


图 2－36　间接任务切换

任务切换过程

直接任务切换

- $DPL_{TSS} \geqslant MAX(CPL, RPL)$【值比较】

间接任务切换

- 当$CPL > DPL_{TSS}$时，就不能采用任务的直接切换，必须通过任务门进行任务的切换。
- 此时，指令中包含的是任务门选择符。选择符指向的是任务门描述符，门中的TSS选择符选中新任务的TSS描述符，激活新的TSS，启动新的任务。

任务切换过程

- 任务切换的步骤
  - 把寄存器现场保存到当前任务的TSS（任务A的TSS）。
  - 把指示目标任务（任务B）TSS的选择符装入TR寄存器中，同时把对应TSS的描述符装入TR的高速缓冲寄存器中。
  - 恢复当前任务（任务B）的寄存器现场。
  - 进行链接处理。

任务切换过程

- 把CR0中的TS标志置为1，这表示已发生过任务切换，在当前任务使用协处理器指令时，产生故障（向量号为7）。
- 把TSS中的CS选择符的RPL作为当前任务特权级，设置为CPL。
- 装载LDTR寄存器。
- 装载代码段寄存器CS、堆栈段寄存器SS和各数据段寄存器及其高速缓冲寄存器。

保护

保护模式下设置了0、1、2、3共4个特权级，以区分操作系统程序和应用程序。这种特权级机制阻止了用户程序对操作系统的非法访问，保证高特权级的代码或数据不被低特权级的程序所破坏。

对数据访问的保护

- 为段寄存器赋值时的检查
  - 程序要访问某一个段时，在将选择符赋给DS之前，CPU依据选择符的TI位在GDT或LDT中读入段描述符，并验证：
    - P位，如果P位为0，CPU会引发一个异常，由操作系统程序进行处理
    - 段类型是否与目标段选择符类型（CS、DS、SS等）一致，即类型检查
  - 限长检查
  - 其他属性检查

限长和类型检查

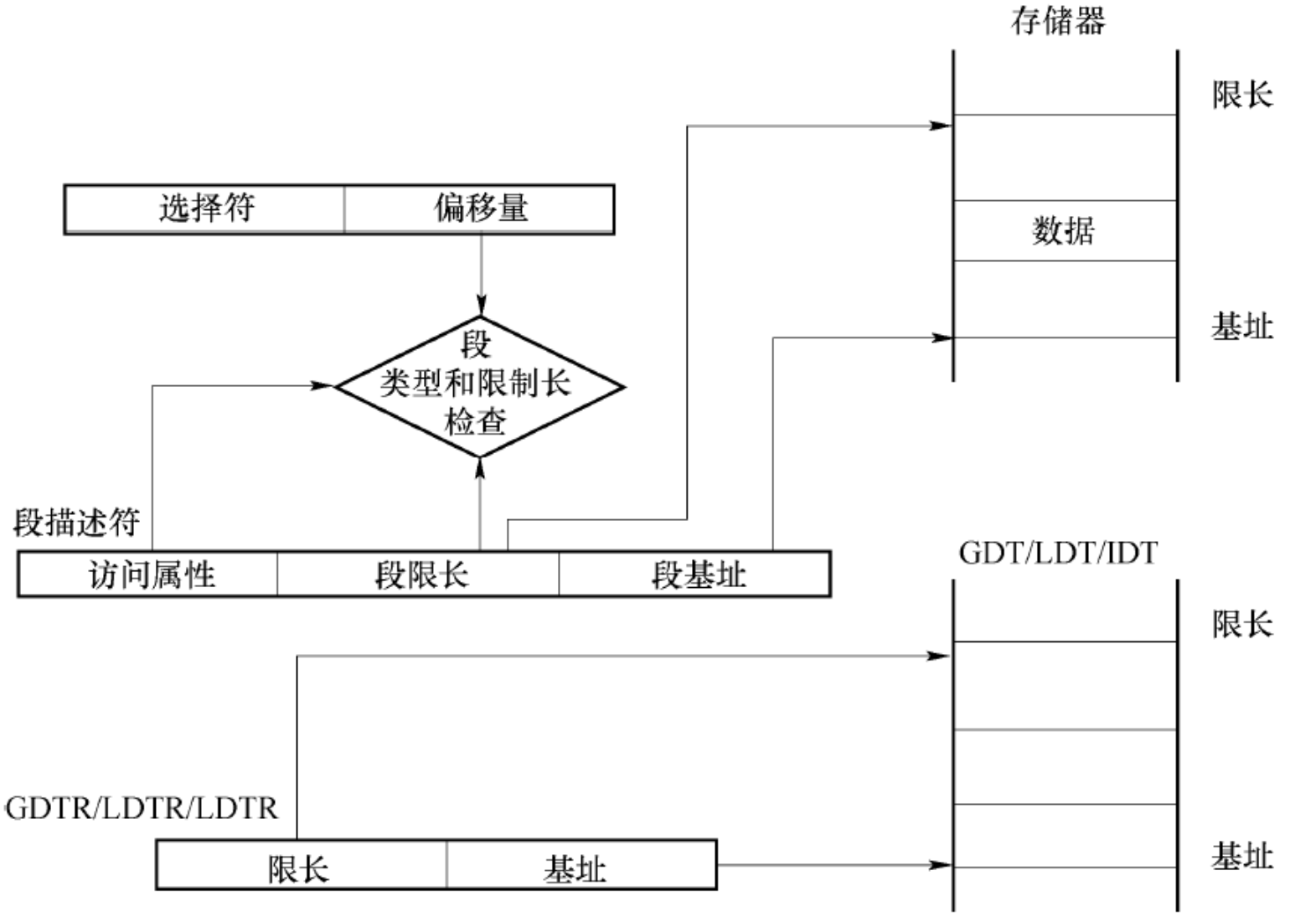


图 2－38　类型和限长检查

特权级检查

- 依据CPL、DPL、RPL来判断特权级是否满足要求：DPL ≥ MAX(CPL, RPL).
  - CPL是当前正在运行的程序的特权级.CS寄存器的最低两位
  - DPL是描述符特权级，位于段描述符中，它表明了什么样的特权级程序可以使用这个段。
  - RPL是请求特权级。如果RPL大于CPL，当前程序降级和RPL匹配。

注：数字越大特权级越低

特权级检查

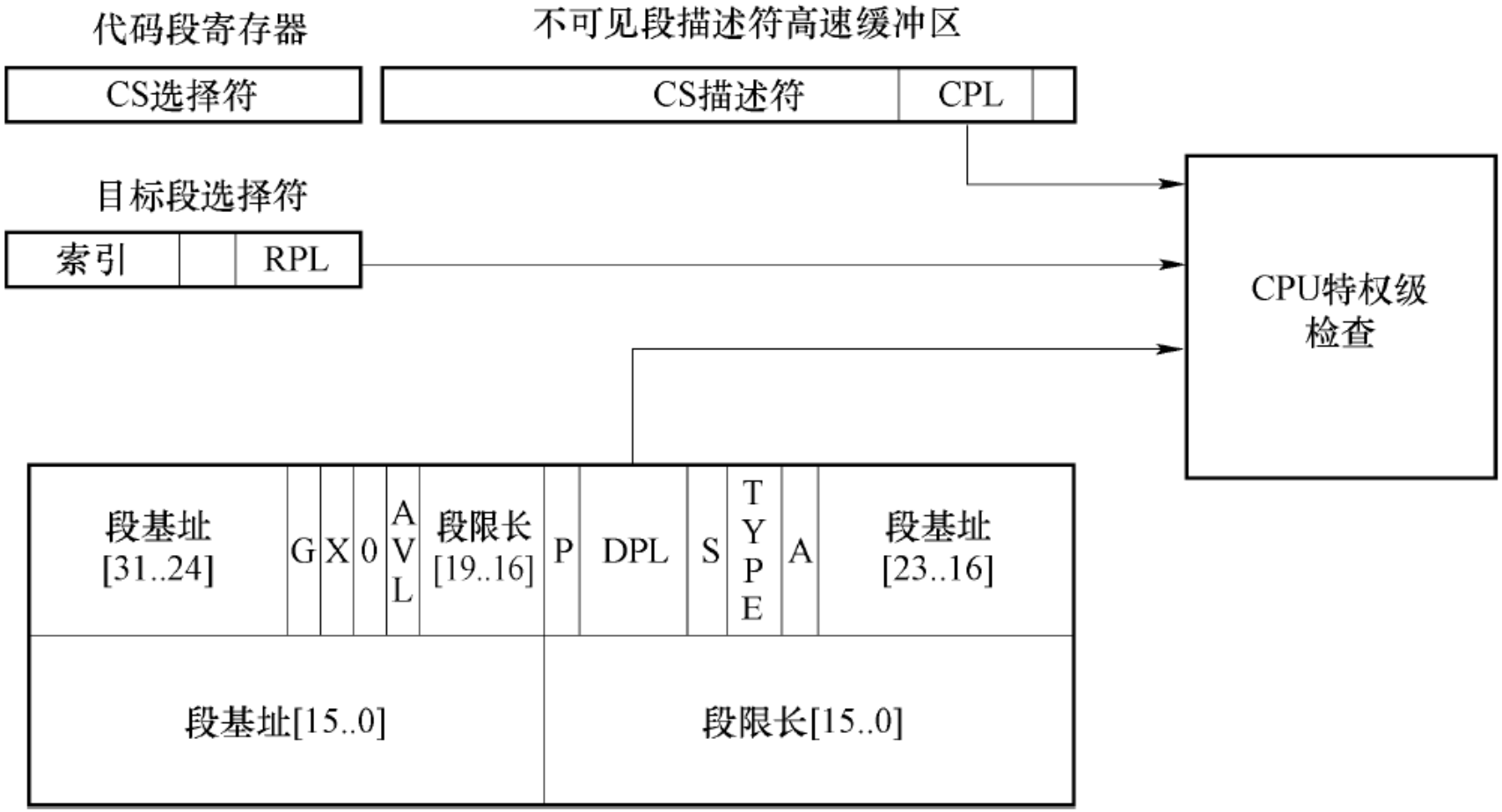


图 2－39　段访问时的特权级检查

数据段访问示例

- DPL ≥ MAX(CPL, RPL)

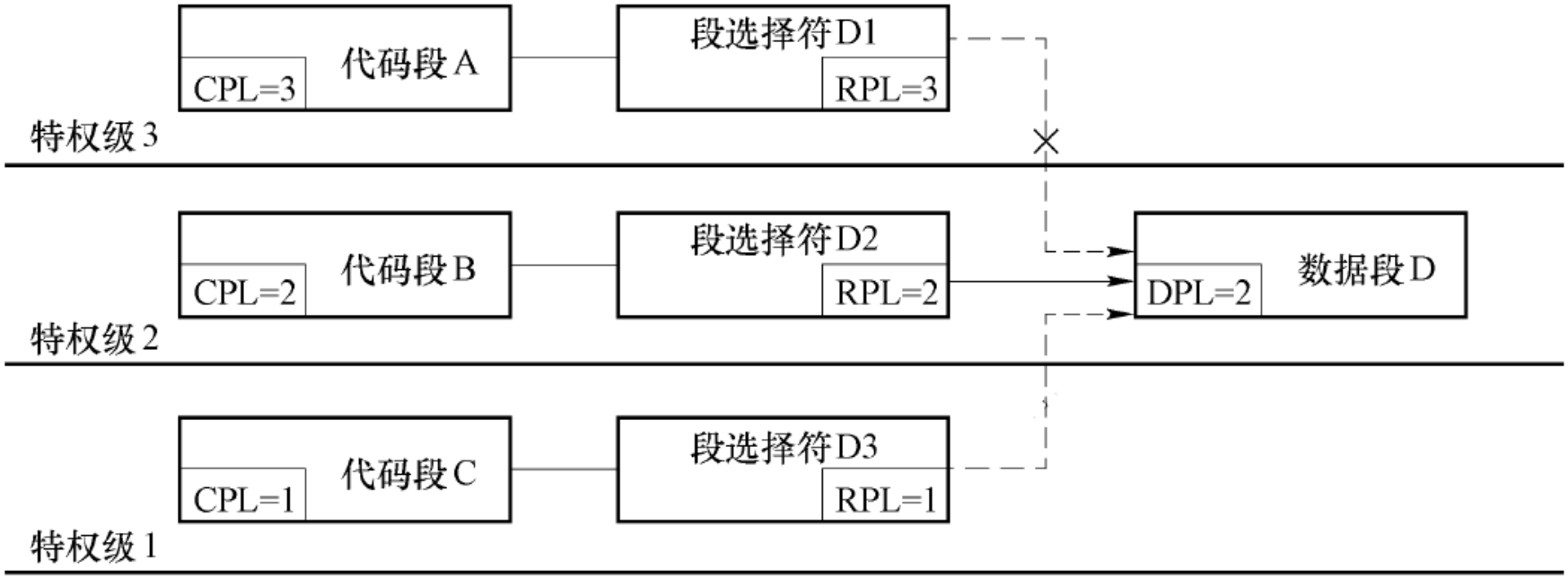


图 2－40　特权级保护数据段访问的一个例子

对程序转移的保护

- 直接转移的保护
  - 同一代码段内转移时，CS不变，特权级不发生变化，只需要检查限长。
  - 段间调用或跳转，检查限长、特权级CPL和DPL。
    - CPL=DPL，允许跳转和调用。
    - CPL<DPL，禁止。高特权级不能转移到低特权级
    - CPL>DPL，此时要检查段描述符的C位。如果C位为1，表示这是一致代码段,允许跳转和调用。

一致代码段

- 一致代码段：操作系统用于共享的代码段
  - 库、异常处理程序
  - 允许低特权级任务访问
  - 特权级高代码不可访问特权级低代码
  - 特权级不改变，还是用户态运行
- 非一致代码段：操作系统保护的系统代码
  - 只允许特权级相同的程序间访问
  - 用户访问用户态；核心访问核心态

对转移程序的保护

- C=0，非一致代码段

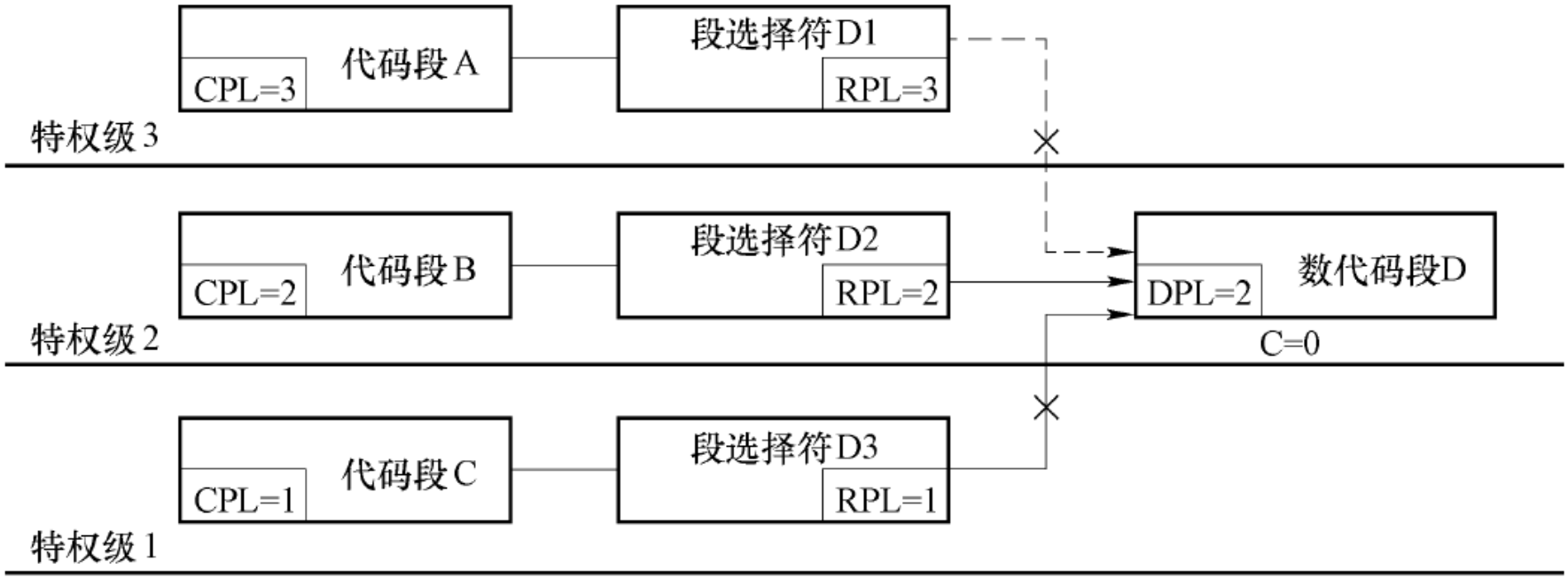


图 2－41　C=0 的代码段

对转移程序的保护

- C=1，一致代码段

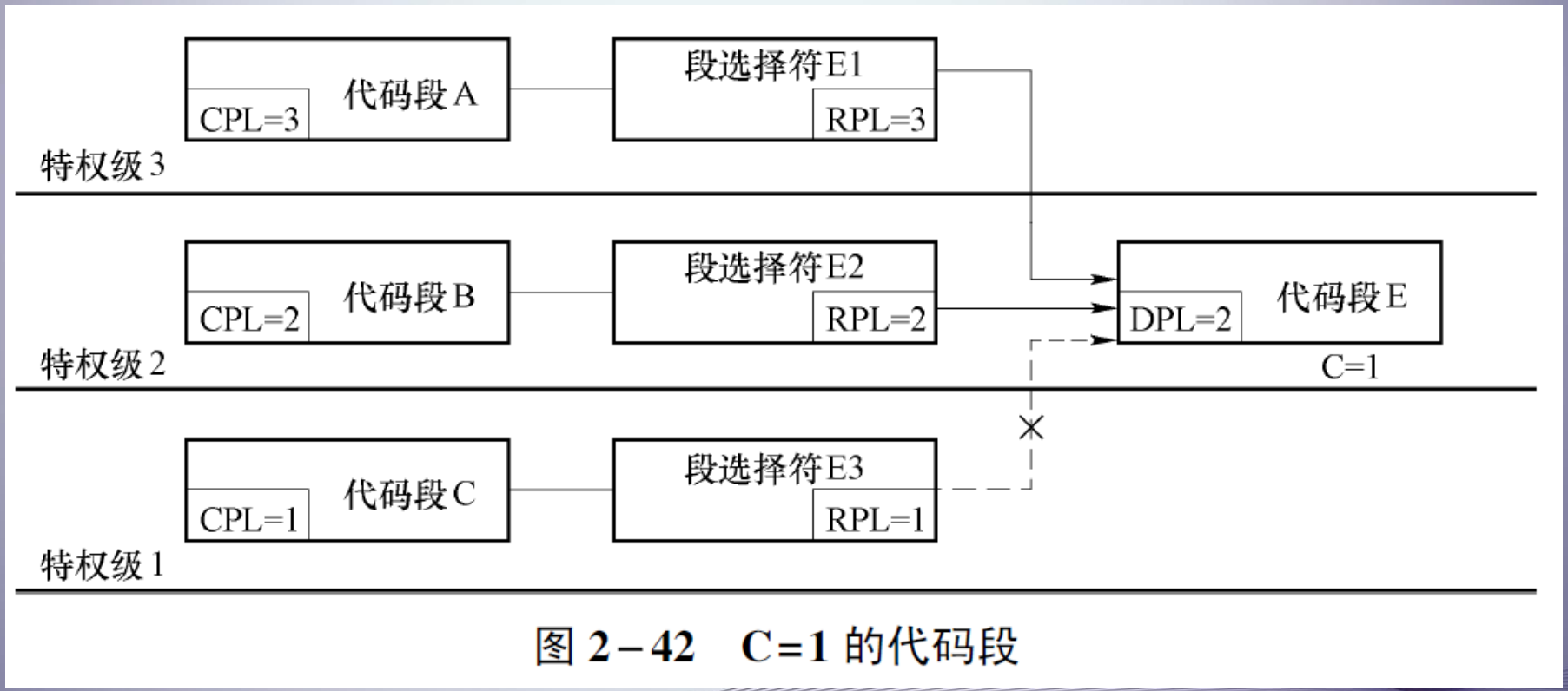


图 2－42　C＝1 的代码段

间接转移的保护

- 直接转移无需通过任务门　　JMP X:Y
- 间接转移通过任务门，涉及任务门描述符
  - 当前特权级CPL，即JMP 或CALL 指令所在的程序的特权级；
  - 请求特权级RPL，即选择符X的最低2位；
  - DPL_{GATE}，即门描述符的DPL；
  - DPL_{CODE}，即目标代码段描述符的DPL；
  - C_{CODE}，即目标代码段描述符的C位（目标代码段是否为一致代码段）。

间接转移的保护

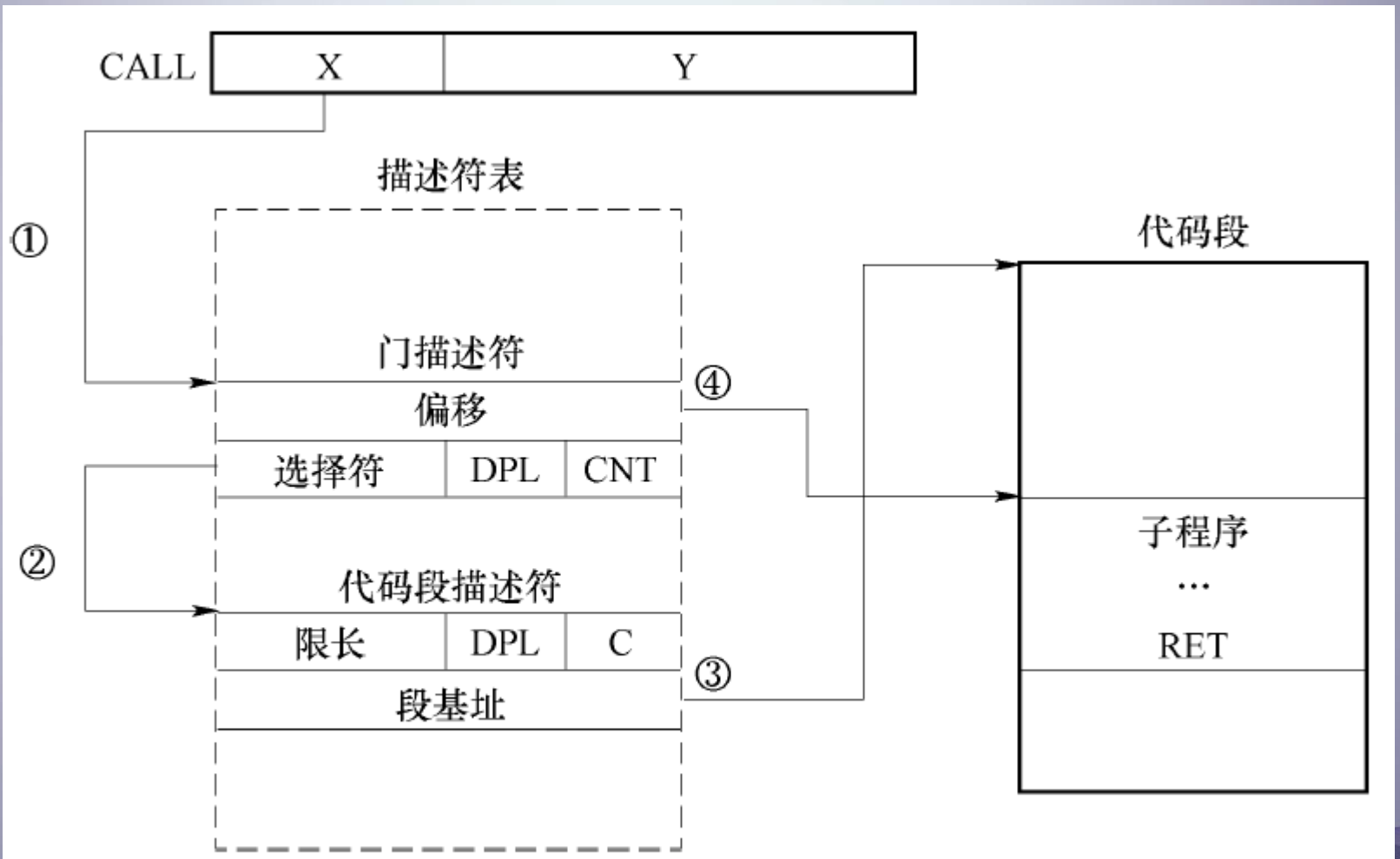


图 2-33　通过调用门转移到更高特权级

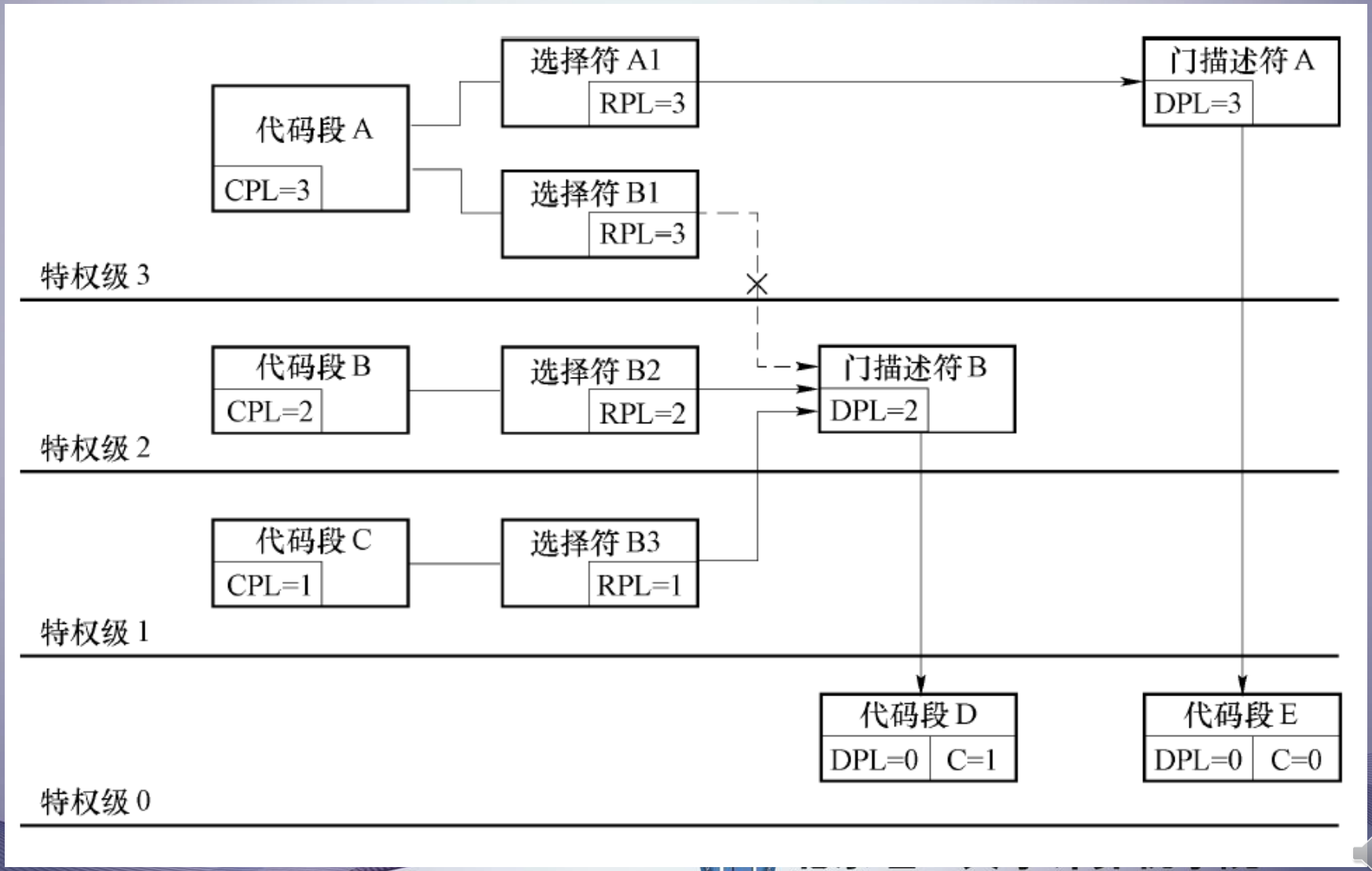
CALL指令

- 满足两个条件
  - $DPL_{GATE} \geqslant MAX(CPL, RPL)$
  - $DPL_{CODE} \leqslant CPL$
    - C=1，一致代码段，当前特权级不变
    - C=0，非一致，当前特权级提升为：
    - DPL_{CODE}（$DPL_{CODE} < CPL$）或者维持不变（$DPL_{CODE} = CPL$）
    - 防止执行调用门后特权级降低

JMP指令

- 满足两个条件
  - $DPL_{GATE} \geqslant MAX(CPL, RPL)$
  - $DPL_{CODE} \leqslant CPL$ (C=1) 或者 $DPL_{CODE} = CPL$ (C=0)

调用门特权级检查


选择符 A1
RPL=3
门描述符 A
DPL=3
代码段 A
CPL=3
选择符 B1
RPL=3
特权级 3
代码段 B
CPL=2
选择符 B2
RPL=2
门描述符 B
DPL=2
特权级 2
代码段 C
CPL=1
选择符 B3
RPL=1
特权级 1
代码段 D
DPL=0
C=1
代码段 E
DPL=0
C=0
特权级 0

输入输出保护

- IO端口对系统运行很重要
- CPU采用I/O特权级IOPL和TSS段中I/O许可位图的方法来控制输入/输出，实现输入/输出保护。
  - 在EFLAGS寄存器中，有2位是输入输出特权级IOPL。CPL≤IOPL时，可以执行I/O敏感指令。

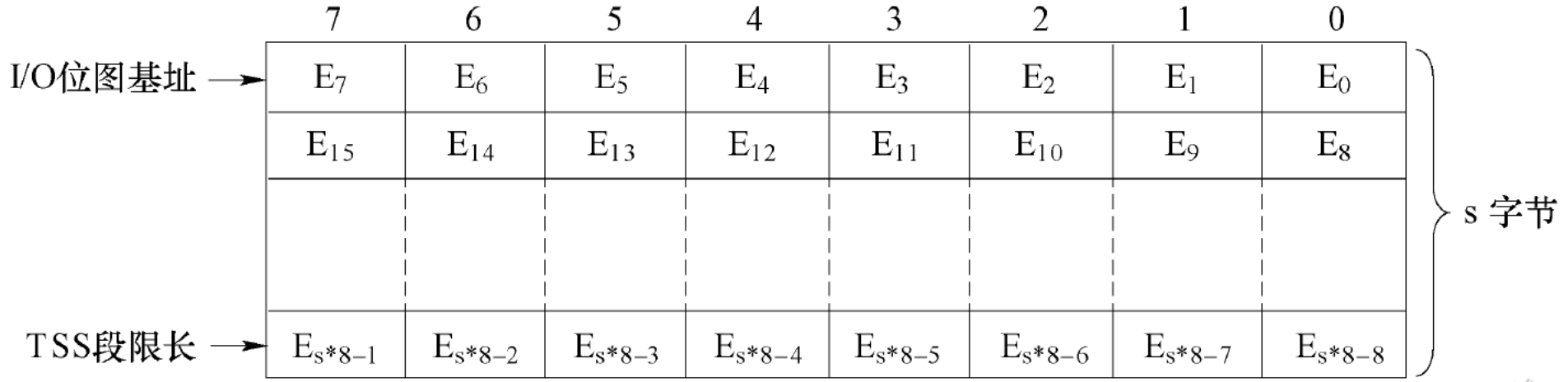
IO敏感指令

- 实模式都可执行
- 保护模式下CPL=0可执行，故针对应用程序

指令	功能	保护方式下的执行条件
CLI	置 IF 位 =0，关中断	CPL<=IOPL
STI	置 IF 位 =1，开中断	CPL<=IOPL
IN	从 I/O 地址读数据	CPL<=IOPL 或 I/O 位图许可
INS	从 I/O 地址连续读数据	CPL<=IOPL 或 I/O 位图许可
OUT	向 I/O 地址写数据	CPL<=IOPL 或 I/O 位图许可
OUTS	向 I/O 地址连续写数据	CPL<=IOPL 或 I/O 位图许可

IO许可位图

- IOPL限制导致应用程序要么访问所有IO地址，要不都不能访问，不够灵活。
- 0-不限制，1-IOPL级别或更高
- IN EAX，DX涉及4个地址



- 作业
  - 习题1　　1.9
  - 习题2　　2.15、2.21、2.23、2.24