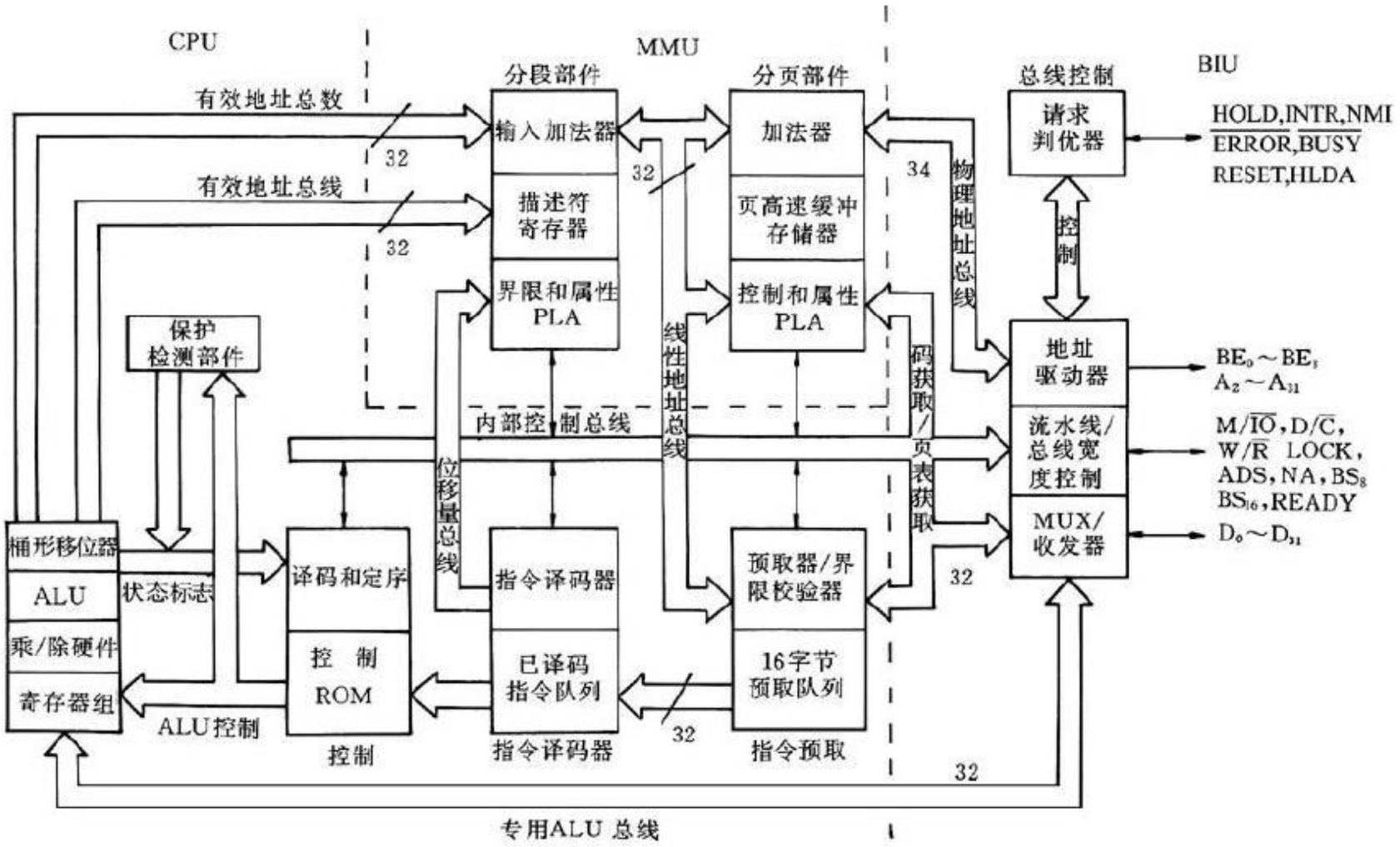
汇编语言与接口技术

微处理器管理模式

微处理器的基本结构

- CPU
- MMU
- BIU



BIU

- 总线接口部件（Bus Interface Unit，BIU）通过数据总线、地址总线、控制总线来与外部环境联系，包括从存储器中预取指令、读写数据，从I/O端口读写数据，以及其他的控制功能。
- 数据总线和地址总线都是32位的。
- 从存储器中存储数据最快也需要两个时钟周期内完成。

CPU

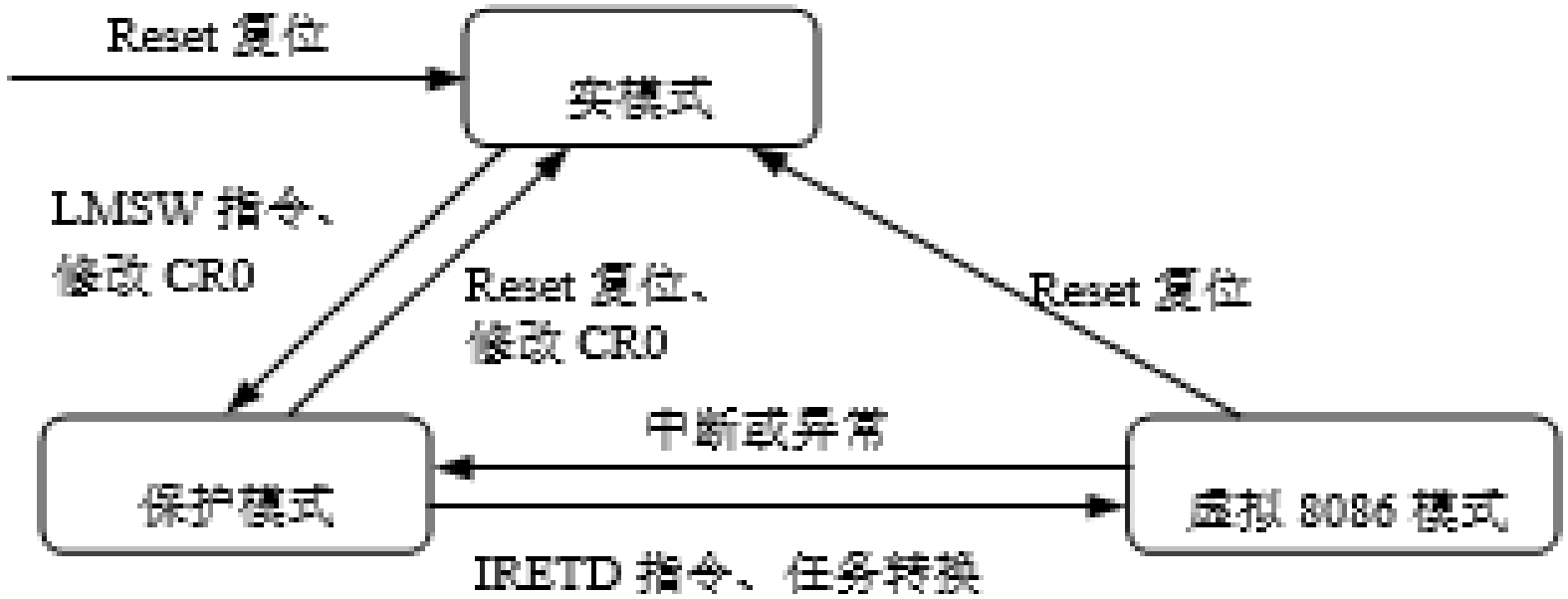
- 中央处理部件（Central Process Unit，CPU）由指令部件和执行部件组成。指令部件包括指令预取单元（Instruction Prefetch Unit，IUP）和指令译码单元（Instruction Decode Unit，IDU）两部分
- 指令预取负责从存储器取出指令，放到一个16字节的FIFO指令队列中，这个队列叫做指令预取队列。
- 指令译码单元从指令预取队列中取出指令，进行译码，并将译码后的可执行指令放入已译码指令队列中。

- 执行部件(Execution Unit，EU)执行从已译码指令队列中取出的指令。它包含8个32位通用寄存器、32位的算术运算单元ALU、1个64位的移位器和乘/除硬件。
- 如果是算术、逻辑或者移位指令，则交给ALU处理，若指令执行时需要段或者页单元产生操作数地址（寻址），则交给分段或者分页单元进行处理。

MMU

- 存储器管理单元由分段部件和分页部件组成，提供存储器管理和保护服务，实现从逻辑地址到物理地址的转换，既支持段式存储管理、页式存储管理，也支持段页存储管理。

CPU工作模式


Reset 复位
实模式
LMSW 指令、
修改 CR0
Reset 复位、
修改 CR0
Reset 复位
中断或异常
保护模式
虚拟 8086 模式
IRETD 指令、任务转换

实模式

- CPU被复位（加电）时，自动进入实模式。在实模式下，这些CPU就相当于高性能的8086，使用1MB地址空间以及16位的“段首址:偏移”的地址格式。
- 全部权限

保护模式

- CPU支持内存分页机制，提供段式和页式内存管理功能，协助操作系统高效地实现虚拟内存，支持多任务和特权级等。
- 保护模式下CPU执行JMP/CALL/IRET等指令，就可实现任务切换。
- 分为特权级0、1、2、3

虚拟8086模式

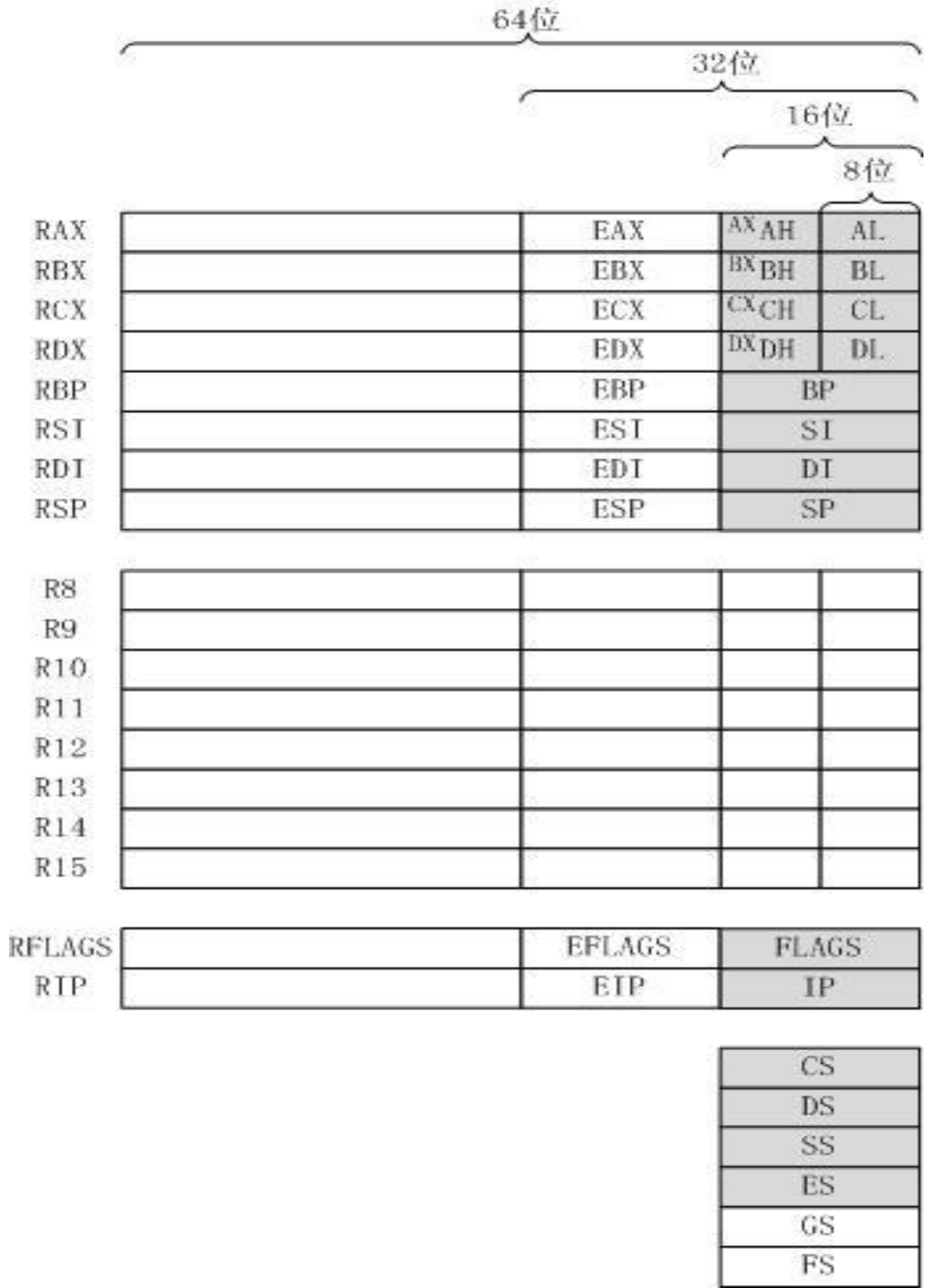
- 兼容以前的DOS及应用程序（即8086程序），从80386开始增加了虚拟8086模式（V86模式）
- 86模式是以任务形式在保护模式上执行的，每个任务都有自己的任务状态段，各个V86任务所拥有的1MB地址空间相互独立
- 特权指令，如屏蔽中断指令CLI、中断指令INT、中断返回指令IRET等，在DOS程序中是合法的。如果不让这些指令在V86模式中执行，DOS程序就无法工作。为了解决这个问题，V86管理程序采用模拟的方法来完成这些指令。

寄存器

- 微处理器内部的寄存器可以分为:
- 程序可见的寄存器（Program Visible Register）
- 程序不可见的寄存器（Program Invisible Register）
- 80386以上CPU才包含程序不可见寄存器组

程序可见寄存器

- Intel 8086~Core2（包括64位扩展）



通用型寄存器

寄存器	常用功能	64位	32位	16位	8位
RAX	累加器，乘法、除法运算等指令的专用寄存器。	RAX	EAX	AX	AH，AL
RBX	保存数据，可用作基址寄存器。	RBX	EBX	BX	BH，BL
RCX	保存数据，计数值（用于循环，串指令等），80386以上CPU也可用于访问存储器的偏移地址。	RCX	ECX	CX	CH，CL
RDX	保存数据，乘法、除法运算指令的专用寄存器，80386以上CPU也可用于访问存储器的偏移地址。	RDX	EDX	DX	DH，DL
RBP	保存访问存储单元时的偏移地址。	RBP	EBP	BP	无
RDI	用于寻址串指令的目的操作数。	RDI	EDI	DI	无
RSI	用于寻址串指令的源的操作数。	RSI	ESI	SI	无

- 通用寄存器功能中RAX，RBX，RCX和RDX可以作为64位，32位，16位和8位寄存器使用，微处理器访问不同长度数据时，可以直接利用表中的命名。
- 如利用MOV指令修改DX的值：MOV DX，0010h。指令给DX寄存器赋值0010h，指令执行结果只会改变寄存器低16位，RDX的其他部分保持不变。又如EAX的值为6BC30E9FH，那么，AX=0E9FH（AH=0EH，AL=9FH）。

64位中新增（非扩展）寄存器

- R8~R15共8个64位寄存器。这些寄存器可以按照字节、字、双字或者四字方式寻址。

访问位数	控制字	寄存器位置	示例
8	B	7~0	MOV R8B，R9B
16	W	15~0	MOV R9W，AX
32	D	31~0	MOV R10D，EAX
64	无	63~0	MOV R11，RAX

段寄存器

- 80286以前的CPU有4个段寄存器，分别称为代码段寄存器CS（Code Segment），数据段寄存器DS（Data Segment），堆栈段寄存器SS（Stack Segment），附加数据段寄存器ES（Extra Segment）。自80386 CPU开始，增加了FS和GS两个段寄存器。
- 分段：基于逻辑
- 分页：基于管理
- 实模式：段地址+段内偏移
- 保护模式：段选择符

- 这些段寄存器都是16位的，对于16位CPU段长度限制为2^{16}B=64KB，对于32位CPU段长度限制为2^{32}B=4GB，
- 实模式和V86模式下它们的用法兼容16位CPU，即段寄存器保存20位段首址的高16位，段首址的低4位为0。
- 在保护模式下，不直接存放段基址，而是存放一个索引，称之为段选择符（Segment Selector）。由段选择符从全局描述符表或局部描述符表中找到8个字节长的段描述符，从而确定关于这个段的全部描述信息。

- RPL（Requestor Privilege Level）：请求特权级，表示将要访问的段的特权级。取值范围为0～3。
- TI（Table Indicator）：表指示符。为0时，从全局描述符表（GDT）中选择描述符；为1时，从局部描述符表（LDT）中选择描述符。

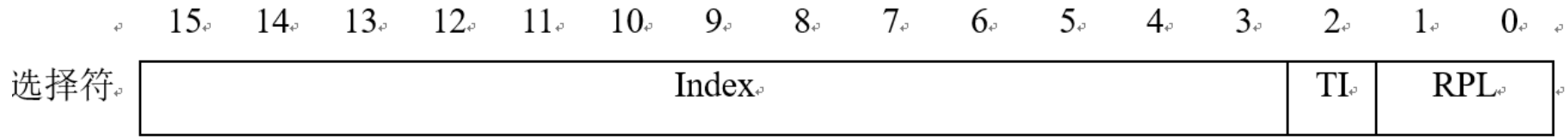
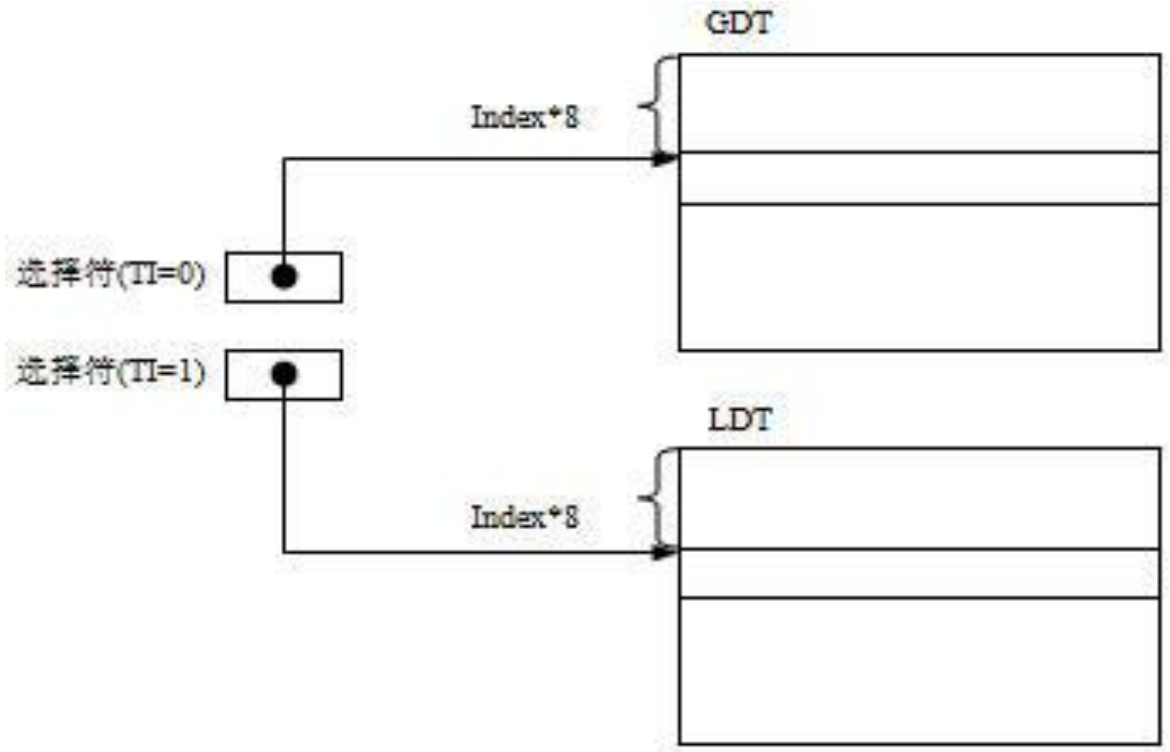


图 3-4 选择符的格式

- Index：索引。指出要访问描述符在段描述符表中的顺序号，Index占13位。因此，顺序号的范围是0～8191。每个段描述符表（GDT或LDT）中最多有8192=2^{13}个描述符。



- DS=0023H=0000 0000 0010 0011b，可知：Index=0 0000 0000 0100b=4。TI=0，RPL=11b=3。因为TI=0，DS的段描述符在GDT中。Index×8=4×8=0020H，该描述符在GDT表中的位置是0020H～0027H，占8个字节。RPL=3，请求特权级为3。

专用寄存器

- Intel 8086~Core2专用型寄存器包括：指令指针寄存器、堆栈指针寄存器和标志寄存器等。

指令指针寄存器IP/EIP/RIP

- 指令指针指向程序的下一条指令。
- 当微处理器为8086/8088，80286或者工作在实模式下时，这个寄存器取16位IP；80386以及更高型号的微处理工作于保护模式下时这个寄存器取32位EIP。在64位模式中，RIP包含40位地址总线。
- 指令指针可以由转移指令或者调用指令修改（间接）。

堆栈指针寄存器SP/ESP/RSP

- 堆栈指针，指向栈顶单元。这个寄存器作为16位寄存器时使用SP；作为32位寄存器时，使用ESP；64位使用RSP。

标志寄存器FLAGS/EFLAGS /RFLAGS

- 用来指示微处理器状态并控制它的操作。
- 从8086开始直到Pentium微处理器向上兼容。
- 8086/8088和80286使用16位FLAGS寄存器。80386以及更高微处理器使用32位EFLAGS寄存器。64位版本中的RFLAGS包含EFLAGS和FLAGS寄存器
- 状态位+控制位

进位标志CF（Carry Flag）

- 当结果的最高位（字节操作时的第7位或字操作时的第15位）产生一个进位或借位，CF = 1，否则CF = 0。
- 在移位或循环移位指令中，会把操作数的最高位（左移时）或最低位（右移时）移入CF中。

奇偶标志PF（Parity Flag）

- Intel微处理器中采用奇校验，当执行结果的低8位中二进制1的个数为奇数时，PF为0，否则为1。
- 不考虑其他位
- 6DH+6DH=DAH
- DAH=1101 1010B，结果中有5个1，因此PF为0。

辅助进位标志AF（Auxiliary Carry Flag）

- 在字节操作时若低半字节（一个字节的低4位）向高半字节有进位或借位；在字操作时若低位字节向高位字节有进位或借位，则AF = 1，否则AF = 0。
- 这个标志用于十进制算术运算指令中，即通过二进制数运算调整为十进制数表示的结果。(BCD)

- 零标志ZF（Zero Flag），当运算结果为零时，ZF为1；否则为0。
- 符号标志SF（Sign Flag），它与运算结果的最高位相同。对字节操作（8位运算）来说，是结果的第7位；对字操作（16位运算）来说，是结果的第15位。当SF＝0时，结果为正数或0；当SF＝1时，结果为负数。

- 单步标志TF（Trap Enable Flag），当TF=1时，CPU进入单步方式，在每条指令执行以后产生一个内部中断（单步中断）。当TF＝0时，CPU执行指令后不产生单步中断。
- 中断允许标志IF（Interrupt Enable Flag），当IF=1时，允许CPU接收外部中断请求，此时为“开中断”状态。当IF=0时，屏蔽外部中断请求，此时为“关中断”状态。

- 方向标志DF（Direction Flag），在字符串操作指令中，当DF=0时，串操作为自动增址；当DF=1时，串操作为自动减址。STD指令置位DF，CLD指令清除DF。
- 溢出标志OF（Overflow Flag），带符号数运算时，当其运算结果超出了8位或16位带符号数所能表达的范围时，将产生溢出，置OF为1，否则OF清0。溢出OF与进位CF是两个不同性质的标志。

- IOPL（I/O Privilege Level），表示I/O特权级，IOPL占2位，取值为0、1、2、3对应四个特权级。只有特权级高于IOPL的程序才能够执行I/O指令，否则会产生异常，并将任务挂起。
- NT（Nest Task），嵌套任务位，此位只用于保护模式，如果保护模式下当前的任务嵌套在其他任务中，此位为1，否则为0。IRET指令会检测NT的值。若NT=0，则执行中断的正常返回；若NT＝1，则执行任务切换操作。

- VM（Virtual 8086 Mode），V86模式位。VM=1时，表示当前CPU正工作在V86模式下；VM=0，表示当前CPU工作在实模式或保护模式下。VM位只能在保护方式中由IRET指令置位（如果当前特权级为0）或在任意特权级上通过任务切换而置位。

486以上新增标志位：ID、VIP，VIF和AC

- AC（Alignment Check），地址对齐检查位。寻址一个字或者双字时，当地址不在字或者双字的边界上，此时AC=1，否则AC=0。
- VIF（Virtual Interrupt Flag），虚拟中断标志，与VIP一起使用，在虚拟方式下提供中断允许标志位IF的副本。
- VIP（Virtual Interrupt Pending），虚拟中断挂起标志，为Pentium~Pentium 4处理器提供有关虚拟模式中断的信息，它用于多任务环境下，为操作系统提供虚拟中断标志和中断挂起信息。
- ID（Identification）微处理器标识标志，用来指示Pentium~Pentium 4处理器，支持CPUID指令。CPUID指令是Intel IA-32架构下获得CPU信息的汇编指令，可以得到CPU类型、型号、制造商、商标、序列号、缓存等一系列CPU相关的信息。

控制寄存器

- CR0 ~ CR3（Control Register）。
- CR0的低5位是系统控制标志，被称为机器状态字（Machine Status Word，MSW）
- 分页机制中用到CR3、CR2和CR0。

CRx	31	30 ~12	11~5	4	3	2	1	0
CR0	PG	00000000000000000000	0000000	ET	TS	EM	MP	PE
CR1	保留未用							
CR2	页故障线性地址							
CR3	页目录基址		000000000000					

- PE（Protection Mode Enable）：保护模式允许标志。PE=0为实模式，CPU复位时自动进入实模式；PE=1为保护模式。可以通过软件设置PE进入或退出保护模式。
- MP（Monitor Coprocessor Extension）：运算协处理器存在位，MP=1表示系统中有协处理器。
- EM（Emulate Processor Extension）：仿真位。设置该位可以使每条ESC指令引起7号中断（ESCape指令用来对80387协处理器指令编码）。EM=1时，可以利用7号中断，用软件来仿真协处理器的功能；EM=0时用硬件控制浮点指令。

- TS（Task Switched）：任务切换标志。TS=1时表明任务已经切换，在保护模式下，TR的内容改变将自动设置此位为1。
- ET（Extension Type）：协处理器选择标志，早期80386里面没有80387协处理器，因此设置此位。当处理器复位时，ET位被初始化以指示系统中数字协处理器的类型。如果系统中存在 80387协处理器，那么ET位置1；如果系统中存在80287协处理器或者不存在协处理器，那么ET位清0。80386以后的系统中ET位被置为1表示系统中存在协处理器。

- PG（Paging Enable）：分页标志。PG=1时，存储器管理单元允许分页，线性地址通过页表转换获得物理地址。PG=0时，分页功能被关闭，线性地址等于物理地址。当PG=0时，CR2和CR3寄存器无效，PG=1时二者用于分页管理机制。

分页-页面

- 每页大小为4KB。CR3也被称做页目录基址寄存器PDBR（Page Directory Base Register），它的高20位用于保存页目录表的起始物理地址的高20位。
- 向CR3中装入一个新值时，低12位必须为0，这是由于页目录是页对齐的，所以仅高20位有效，低12位保留未用；从CR3中取值时，低12位被忽略。

分页-页面

- CR2也被称做页面故障线性地址寄存器（Page Fault Linear Address Register），用于发生页异常时报告出错信息。如在访问某个线性地址时，该地址的所在页没有在内存中，则发生页异常，处理器把引起页异常的线性地址保存在CR2中。

全局描述符表寄存器

- 全局描述符表（Global Descriptor Table，GDT）是用来定义全局存储器中的段，KB）。
- 全局描述符表寄存器（的位置和大小。

- GDTR是48位的寄存器。其最低16位是限长，给出GDT的字节大小（其值比GDT的长度少1），其高32位是基址，指出GDT在物理存储器中存放的基地址。

- 已知GDTR=0E003F0003FFH，则全局描述符表的基址是多少？这个全局描述符表有多大，里面有多少个描述符？
- GDT的地址为0E003F00H
- 长度为3FFH+1=400H。
- 可容纳400H/8=80H个段描述符。

中断描述符表寄存器

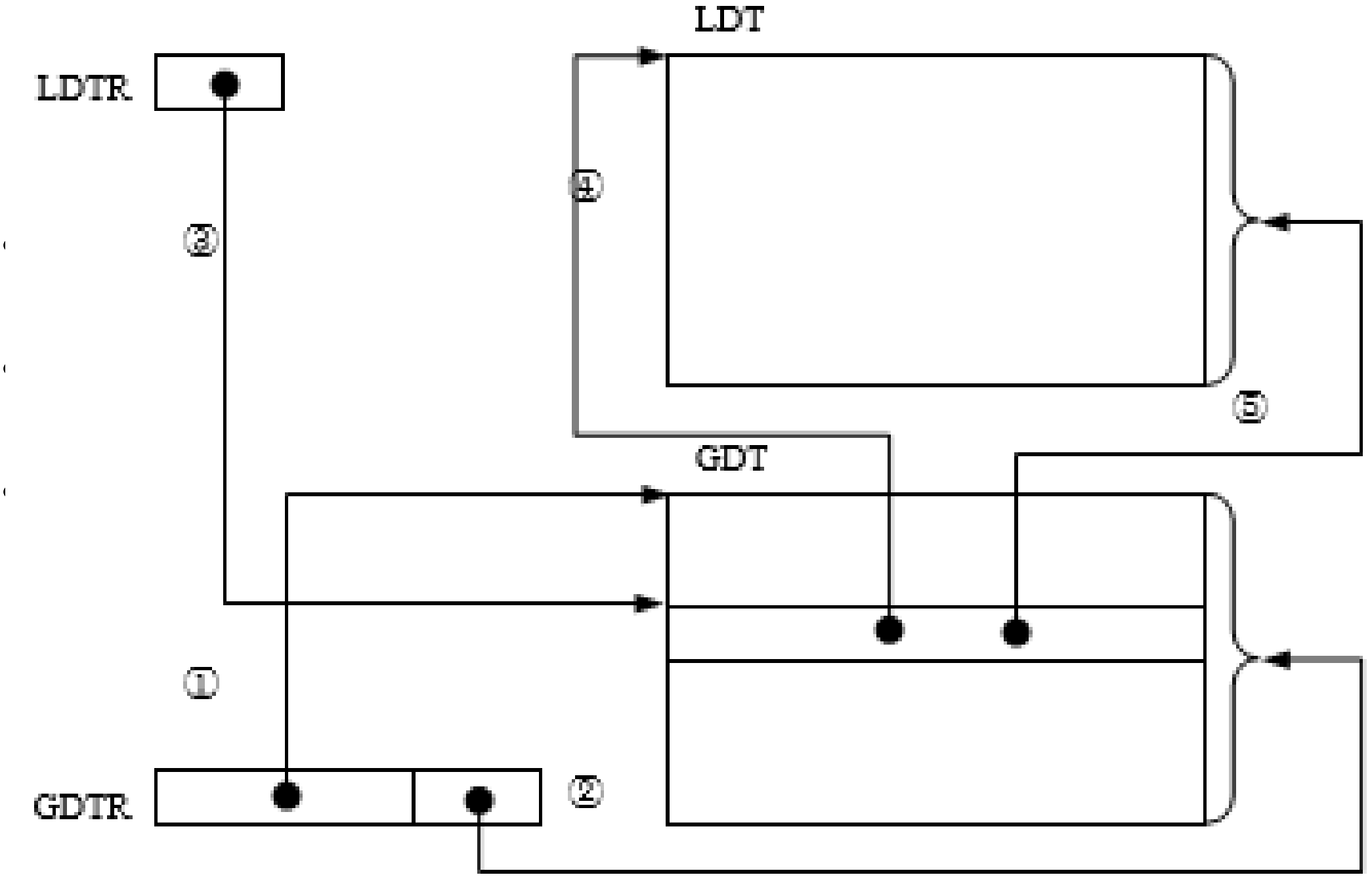
- 中断描述符表寄存器（Interrupt Descriptor Table Register，IDTR）也在存储器中定义了一个表，该表称为中断描述符表IDT。
- IDT中保存的不是段描述符，而是中断门描述符。每个门描述符也包含8字节
- IDT最多包含256个门描述符，因为CPU最多支持256个中断。中断门指向的是中断服务程序的入口。
- IDTR是48位的寄存器。其最低16位是限长，给出IDT的字节大小（其值比IDT的长度少1），其高32位是线性地址，经过分页部件转换为物理地址，指出IDT在物理存储器中存放的基地址。

- 已知IDTR=0E003F40007FFH，则中断描述符表的基址是多少？这个中断描述符表有多大，里面有多少个描述符？
- IDT的地址为0E003F400H
- 长度为7FFH+1=800H
- 可容纳800H/8=100H个描述符。

- 保护模式下的中断描述符表IDT的位置是可变的，实模式下的中断向量表的地址是固定在物理地址00000H处。
- GDTR和IDTR的值必须在进入保护模式之前装入。在实模式下执行LGDT和LIDT指令装入GDTR和IDTR。

局部描述符表寄存器

- 保护模式提供了多任务的环境，为每个任务建一个局部描述符表（Local Descriptor Table，LDT）。
- LDT只含有与系统中某一个任务相关的各个段的描述符。
- 可以使每一任务的代码段、数据段、堆栈段与系统其他部分隔离开。


LDT
LDTR
④
②
⑤
GDT
①
GDTR
②

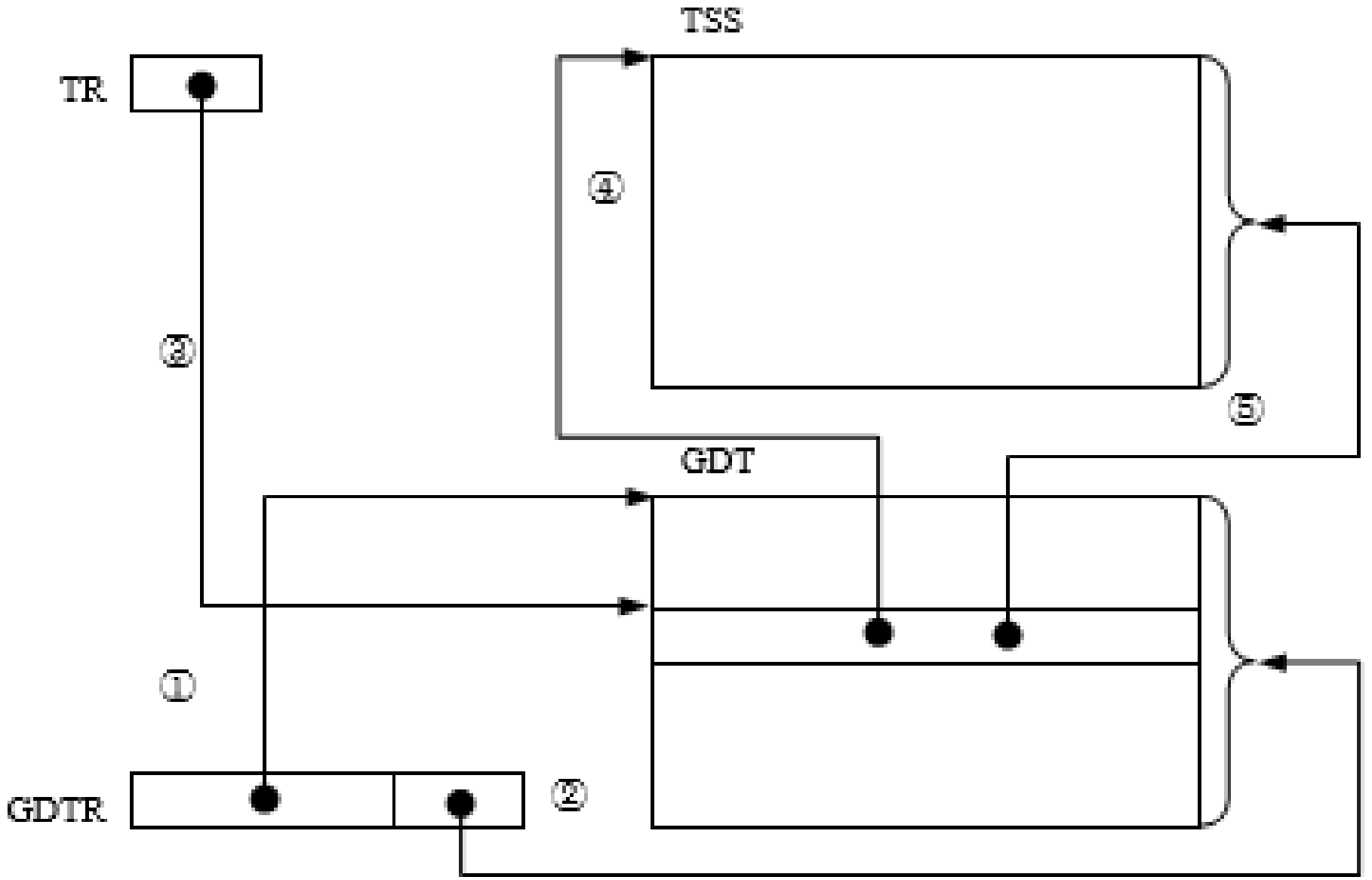
- ①和②步由GDTR确定了GDT表在存储器中的位置和限长。LDTR中是一个选择符，它包含了LDT描述符在GDT中的索引。③步是依据LDTR在GDT中取出LDT描述符的过程。在LDT描述符中，包含由LDT的位置和限长，即④和⑤步

任务寄存器

- 任务寄存器（Task Register，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。

- 在多任务环境下，每个任务都有属于自己的任务状态段（Task Status Segment，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，它指出了TSS描述符在GDT中的顺序号。

- 假定全局描述符表的基址为00011000H，TR为2108H，问TSS描述符的起止范围是多少？
- TSS起始地址=00011000H+2108H=00013108H
- TSS终止位置=00013108H+7H=0001310FH

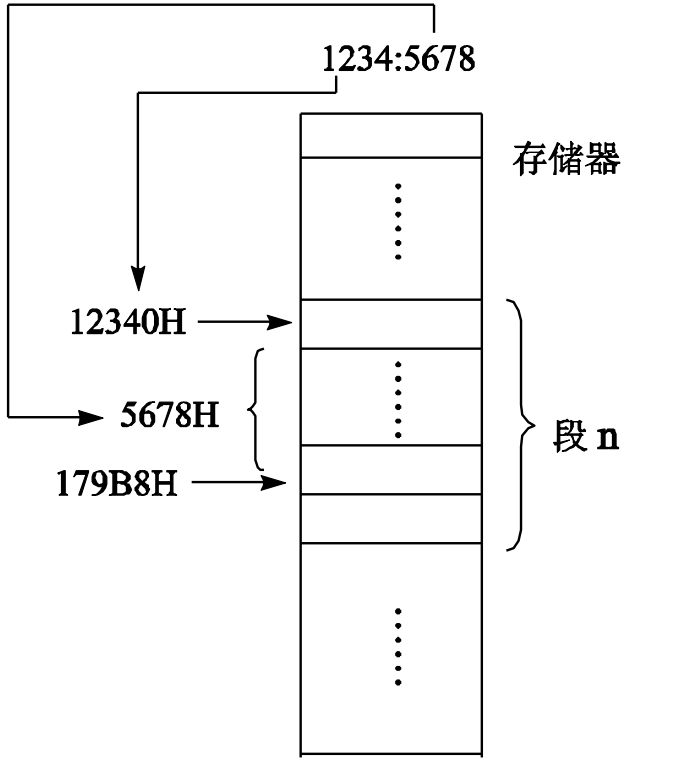

TSS
TR
④
③
⑤
GDT
①
GDTR
②

- ①和②步由GDTR确定了GDT表在存储器中的位置和限长。TR是一个选择符，这个选择符中包含了TSS描述符在GDT中的索引。③步依据TR在GDT中取出TSS描述符。在第④和⑤步中，在TSS描述符中取得TSS的基址和限长。

内存管理

- 实模式下分段管理
- 20位地址总线，最大可寻址内存空间应为2^{20}=1MB，其物理地址范围从00000H ~ FFFFFH。
- 段的大小可以变化，16位CPU中最大的段为2^{16}=64KB。段的起始地址可以在任何16的倍数上，段寄存器中存放段基址除以16得到的商（也就是20位段基址的高16位）。
- 段基址:偏移量
- 10H×段基址+偏移量

- 某内存单元的地址用十六进制数表示为1234:5678
- 其物理地址为179B8H

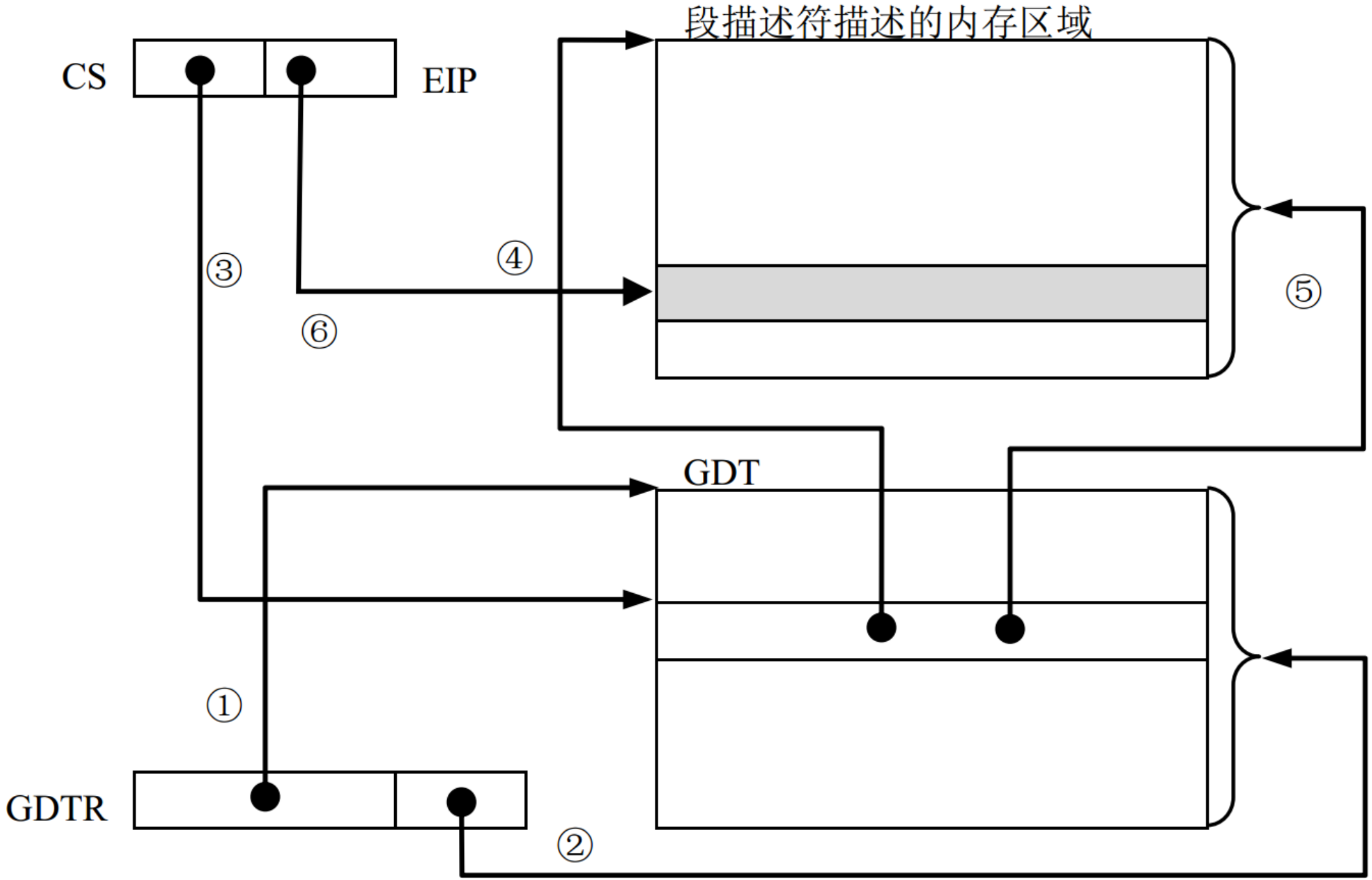


	1	2	3	4	0	(16H×段基址)
+		5	6	7	8	(偏移量)
	1	7	9	B	8	(物理地址)

- 计算实模式下1000:1F00、11F0:0000、1080:1700的物理地址。
- 1000:1F00=10H×1000+1F00=11F00H;
- 11F0:0000=10H×11F0+0000=11F00H;
- 1080:1700=10H×1080+1700=11F00H。

保护模式下分段管理

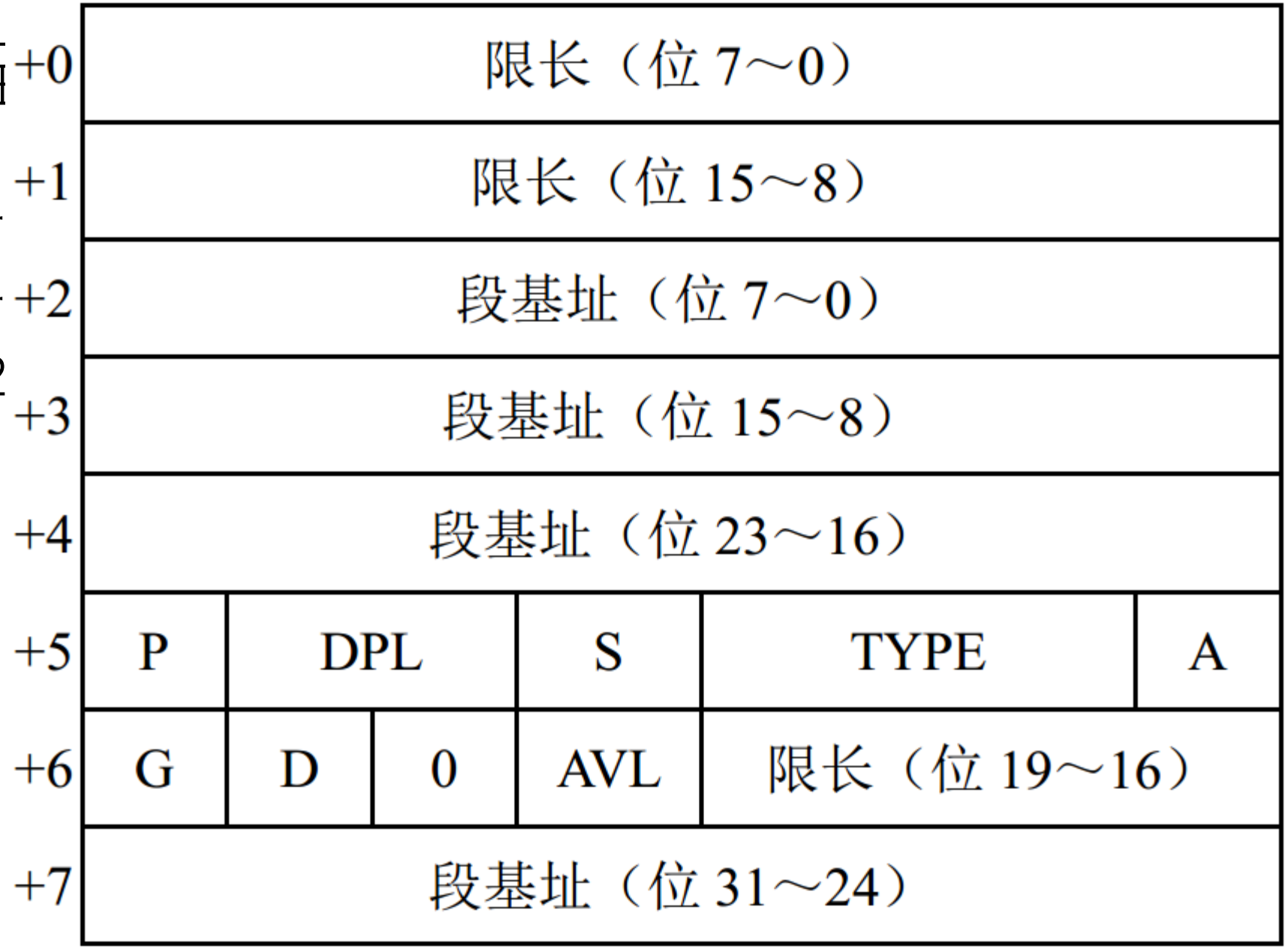
- 保护模式下，逻辑地址同样由段基址:偏移量的格式形成，只不过，原来用来存储段基址的段寄存器不再表示段的起始位置，而是用来表示段选择符。
- 该地址表示形式称为虚拟地址，对应的地址空间称为虚拟地址空间。
- 以CS:EIP为例， CS中存放了一个16位的段选择符，EIP是32位偏移量。16位段选择符加上32位偏移量，总共是48位，其中段选择符的2位RPL与虚拟地址的转换无关，因此可以认为虚拟地址是46位的，段选择符的Index和TI占14位，偏移量为32位，虚拟地址空间为2^{46}B=64TB


段描述符描述的内存区域
CS
EIP
③
④
⑤
⑥
GDT
①
GDTR
②

段描

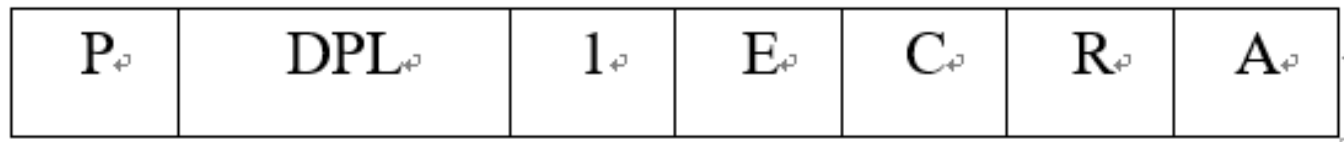
- 段描
- 段描（32

基址）。


+0
限长（位 7～0）
+1
限长（位 15～8）
+2
段基址（位 7～0）
+3
段基址（位 15～8）
+4
段基址（位 23～16）
+5
P
DPL
S
TYPE
A
+6
G
D
0
AVL
限长（位 19～16）
+7
段基址（位 31～24）

S=1 且 E=0 时:

S=1 且 E=1 时:



- P（Present）：存在位。P＝1时表示该段已装入内存；P＝0时表示该段没有在内存中，此时访问该段会产生段异常。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，说明该段的特权级，取值为0～3。
- S（System）：描述符类型位，S=1时，这个段为代码段、数据段或堆栈段；S=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。

- ED（Expansion Direction）：扩展方向位（对数据段或堆栈段）。ED=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。ED=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。限长是指地址上限。一般情况下ED位为0。
- W（Writeable）：写允许位（对数据段或堆栈段）。W=0时，不允许对这个数据段写入；W=1时，允许对这个数据段写入。

- C（Conforming）：一致位（对代码段）。C=0时，这个段不是一致代码段；C=1时，这个段是一致代码段。一致代码段就是操作系统拿出来被共享的代码段，这些代码段允许被低特权级的用户直接调用访问。
- 此时特权级高的程序不允许访问特权级低的代码，即核心态不允许调用用户态的代码；特权级低的程序可以访问到特权级高的代码，但是特权级不会改变。
- 除了一致代码段外，其他的代码段称为非一致代码段。非一致代码段常用于那些为了避免低特权级的访问而被操作系统保护起来的系统代码。非一致代码段只允许特权级相同的程序间访问，绝对禁止不同级访问，即核心态不使用用户态，用户态也不使用核心态。

- R（Readable）：读允许位（对代码段）。R=0时，不允许读这个段的内容；R=1时，允许读这个段的内容。对代码段进行写操作总是被禁止的。
- A（Accessed）：访问位。A = 1表示段已被访问（使用）过；A = 0表示段未被访问过。操作系统利用这个位对段进行使用统计，可以将那些很长时间没有被访问过的段从内存中调出，释放其内存给其他程序所使用。
- G（Granularity）：粒度。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。

- 限长位在描述符中一共占20位。G = 1时，限长的内容加上1后就是段所占的页数，1页的大小为2^{12} = 4KB；G = 0时，段限长以字节为单位，限长的内容+1就是段所占的字节数。
- G = 1，限长位为FFFFFH时，段所占的页数是FFFFFH+1=100000H，即2^{20}页，段的大小为2^{32}=4GB，有效偏移量的范围是00000000H ~ FFFFFFFFH。G = 0，限长位为FFFFFH时，段的大小为2^{20}=1MB，有效偏移量的范围是00000000H ~ 000FFFFFH。

- 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。
- 段的起始地址=段基址=0CD310000H
- （G=0）段的结束地址=段基址+段限长
  =0CD310000H+001FFH=0CD3101FFH
- （G=1）段的结束地址=段基址+段限长
  =0CD310000H+(001FFH+1)×4K−1=0CD50FFFFH

段描述符的实际应用

- 各段寄存器及段描述符的值:
- CS=001B DS=0023 ES=0023 SS=0023 FS=0030 GS=0000
- GDTbase=E003F000 Limit=03FF

E003F000 00 00 00 00 00 00 00 00-FF FF 00 00 00 9B CF 00

E003F010 FF FF 00 00 00 93 CF 00-FF FF 00 00 00 FB CF 00

E003F020 FF FF 00 00 00 F3 CF 00-AB 20 00 20 04 8B 00 80

E003F030 01 00 00 F0 DF 93 C0 FF-FF 0F 00 00 00 F3 40 00

E003F040 FF FF 00 04 00 F2 00 00-00 00 00 00 00 00 00 00

选择符	类型	段基址	结束地址	DPL		
0008	Code32	00000000	FFFFFFFF	0	P	RE
0010	Data32	00000000	FFFFFFFF	0	P	RW
001B	Code32	00000000	FFFFFFFF	3	P	RE
0023	Data32	00000000	FFFFFFFF	3	P	RW
0028	TSS32	80042000	000020AB	0	P	B
0030	Data32	FFDFF000	00001FFF	0	P	RW
003B	Data32	00000000	00000FFF	3	P	RW
0043	Data16	00000400	0000FFFF	3	P	RW
0048	Reserved	00000000	00000000	0	NP	

内容	二进制		7						0
FF	11111111_2	+0	限长（位 7～0）						
FF	11111111_2	+1	限长（位 15～8）						
00	00000000_2	+2	段基址（位 7～0）						
00	00000000_2	+3	段基址（位 15～8）						
00	00000000_2	+4	段基址（位 23～16）						
FB	11111011_2	+5		DPL			TYPE		A
CF	11001111_2	+6				AVL	限长（位 19～16）		
00	00000000_2	+7	段基址（位 31～24）						

- 以CS=001BH为例，001BH=0000 0000 0001 1011b，001BH是段选择符，Index=0 0000 0000 0011b=3H，TI=0，RPL=11b=3H。Index×8=3H×8=0018H，因此，在GDT表中的位置0018H处开始的8个字节，存放的就是001BH选择符对应的段描述符。GDT表的基址为E003F000H，这个段描述符从E003F000H+0018H=0003F018H开始，取出这8个字节为：FF FF 00 00 00 FB CF 00。

- 段基址（位31～24）=00H，段基址（位23～16）=00H，段基址（位15～8）=00H，段基址（位7～0）=00H，所以段基址（位31～0）=00000000H。限长（位19～16）=FH，限长（位15～8）=FFH，限长（位7～0）=FFH，所以限长（位19～0）=FFFFFH。
- G=1，限长是以页为单位的，段的大小为（FFFFFH+1）×2^{12}=100000H×2^{12}=2^{32}=4GB。段的基址为00000000H，长度为4GB，其线性地址范围为00000000H～FFFFFFFFH。

D=1，因此这是一个32位的段。

AVL=0。

P=1，这个段在内存中。

DPL=11_2=3_{16}，段的特权级为3。

S=1，这不是一个系统段，而是一个代码、数据段或堆栈段。

E=1，这是一个代码段（可执行）。

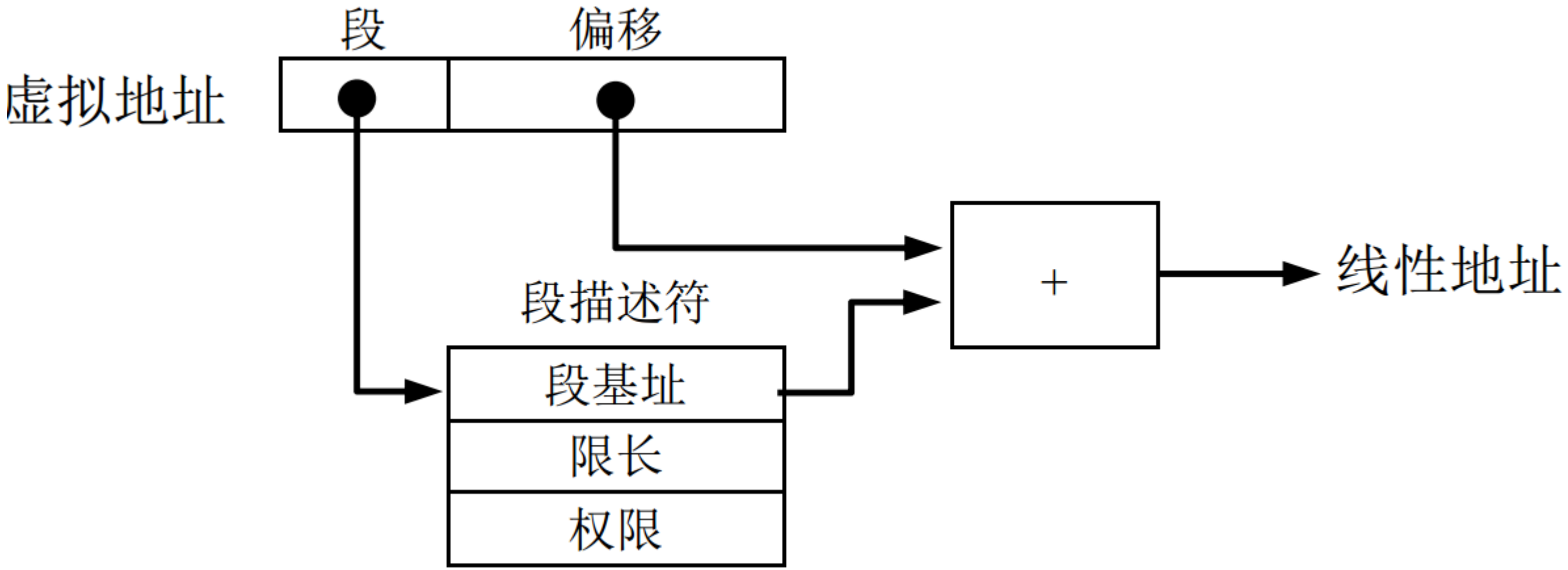
C=0，非一致代码段。因为E=1，这一位是C；若E=0，这一位代表ED。

R=1，可以对这个段进行读操作。因为E=1，这一位是R；若E=0，这一位代表W。

段描述符高速缓存

- 如果每一次都需要到内存中去读取段描述符，那么CPU的运行效率就会极大地降低。为解决这个问题，CPU在内部设置了段描述符高速缓存。段描述符高速缓存总是与CS、DS、ES、SS、FS、GS段寄存器和描述符的当前值保持一致，只有段寄存器的值发生改变时，才需要到GDT或LDT中装入段描述符。
- 这些缓存器是不可见的

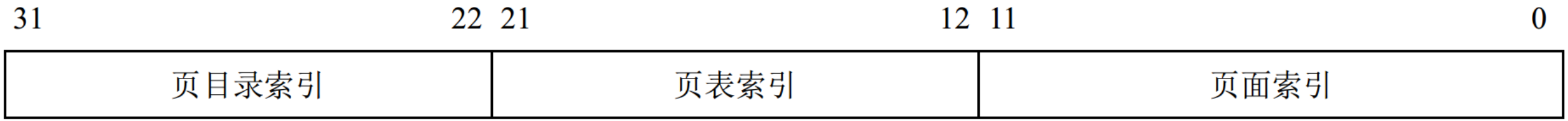
段式地址转换

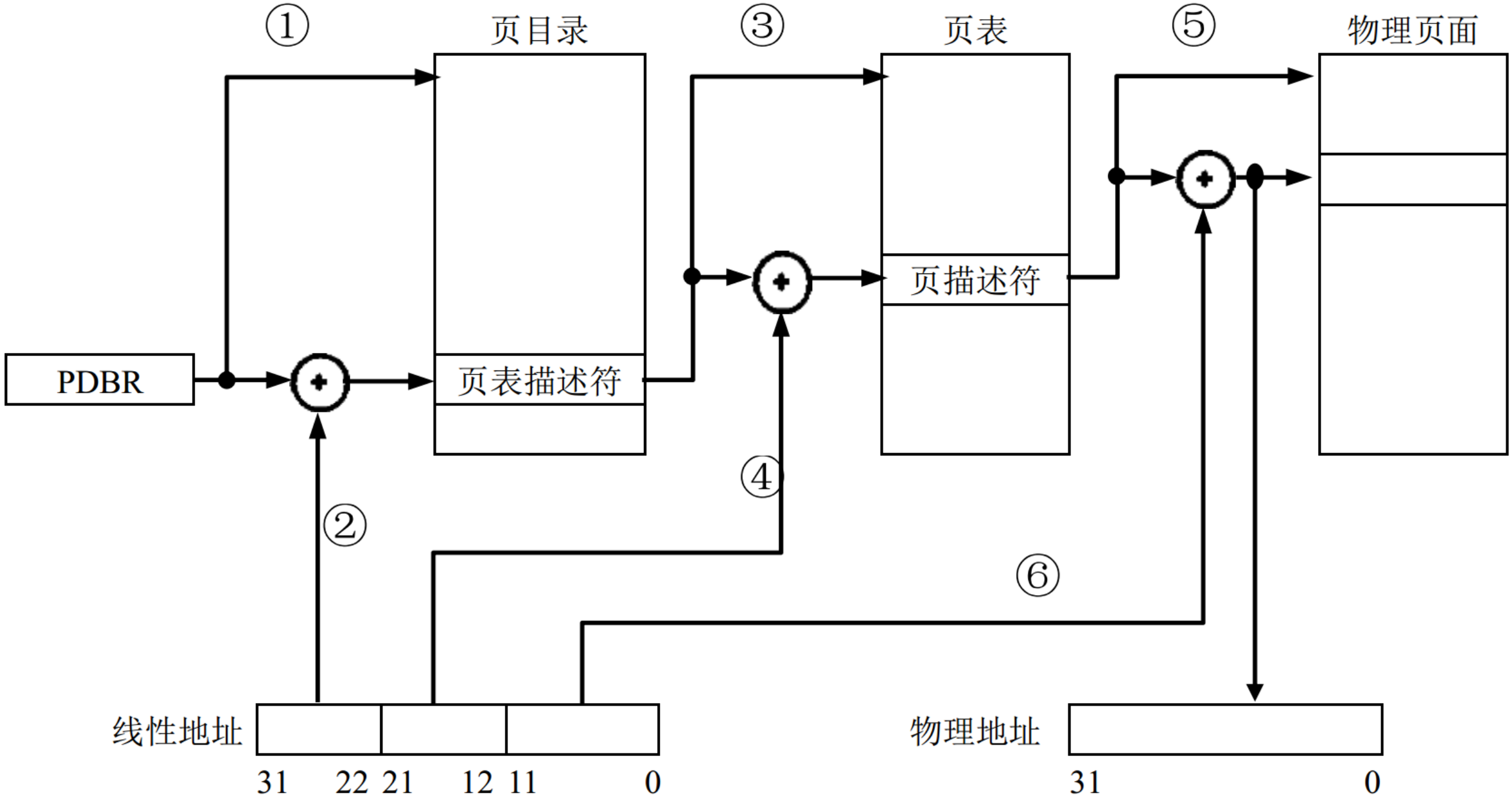


页式内存管理

- 每一个任务都有它自己的一个线性地址空间，由于线性地址是32位的，所以线性地址空间为2^{32}=4GB。分段管理时，段的长度不固定，段和段之间也允许重叠；而页面的划分则严格得多。所有页的长度固定，页与页之间也没有重叠。
- 页面大小为4KB，则32位CPU将4GB的线性地址空间划分成2^{20}页。
- 分页机制就是一种将线性地址的页面映射到物理地址页面的手段，也就是从线性地址到物理地址的转换过程。分页机制中用到了两个表：页目录表和页表。

- 32位线性地址被划分为3个部分：页目录索引（10位），页表索引（10位），页面字节索引（12位），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（Page Frame）内的偏移。



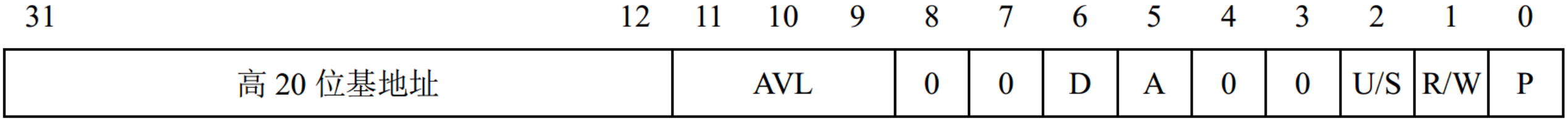

①
页目录
③
页表
⑤
物理页面
PDBR
页表描述符
页描述符
②
④
⑥
线性地址
物理地址
31
22
21
12
11
0
31
0

- ①：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。页目录的大小为4KB，由1024个页表描述符组成，每个页表描述符占4个字节。
- ②：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。
- ③：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。

- ④：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。
- ⑤：页描述符的高20位给出了物理页面的基地址的高位20位。
- ⑥：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。
- 片内转换检测缓冲器（32个页描述符）

页表项

- 页目录表、页表和页面的基地址的低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；D位和A位提供统计信息。

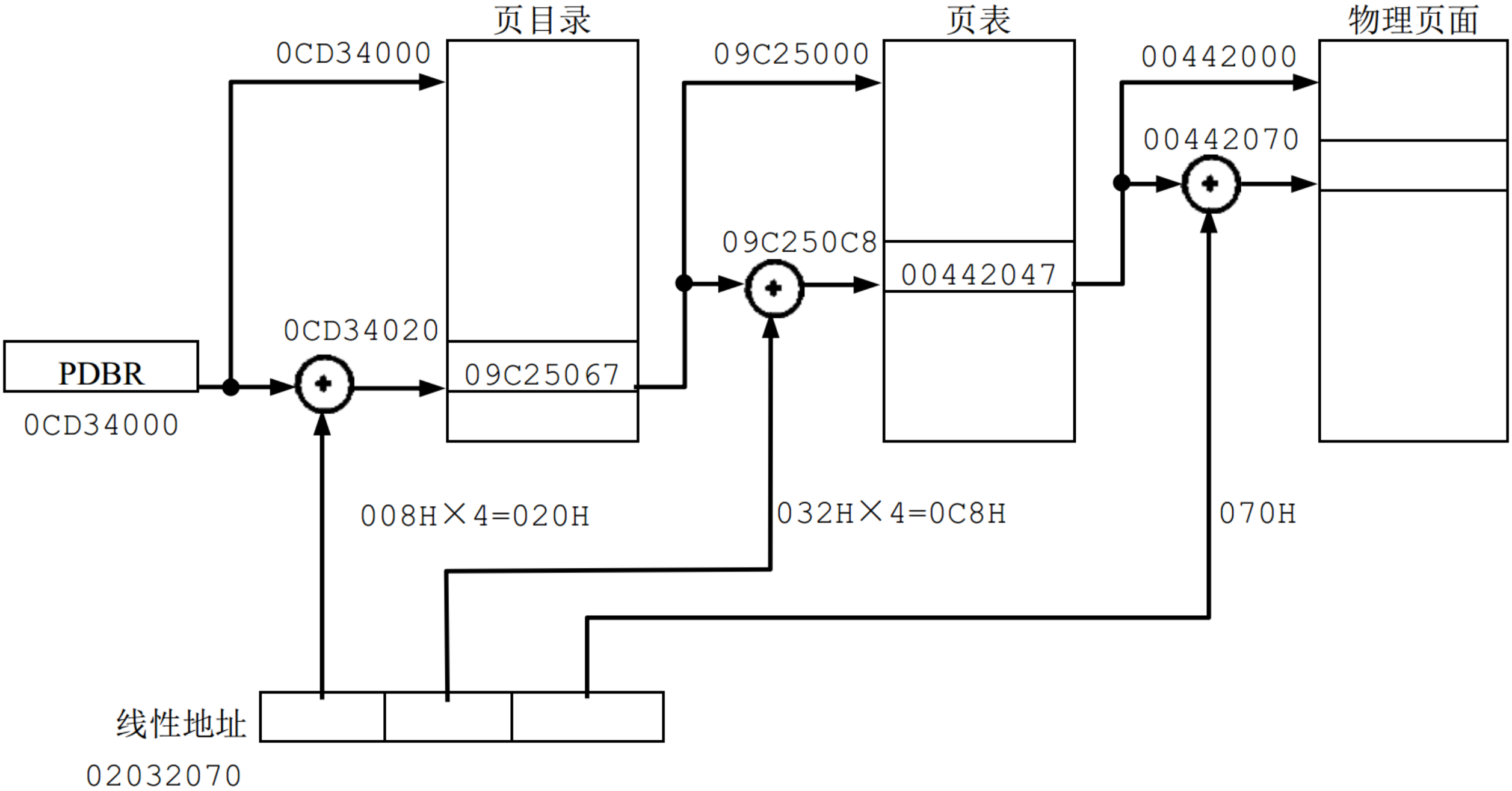


- U/S（User/Supervisor）：用户/管理员位。U/S=0时，只有操作系统程序可以访问该页，不允许用户程序访问，这可以保护操作系统使用的页面不被用户程序破坏（读写）；U/S=1时，允许用户程序访问该页。
- R/W（Read/Write）：读写位。R/W = 0，用户程序对页面只有读权限，不能写入；R/W = 1时，可读/写。不论R/W位设置如何，操作系统的程序都可以对页面进行读写。
- P（Present）：存在位。P=1，页表或页存在于物理内存中；P=0，页表或页不在物理内存中。如果内存不足时，操作系统会选择那些使用频率低的页面，设置它们的P位为0，将这些物理页面释放出来供其他程序使用。产生缺页异常时，程序中使用的32位线性地址被保存在CR2中。

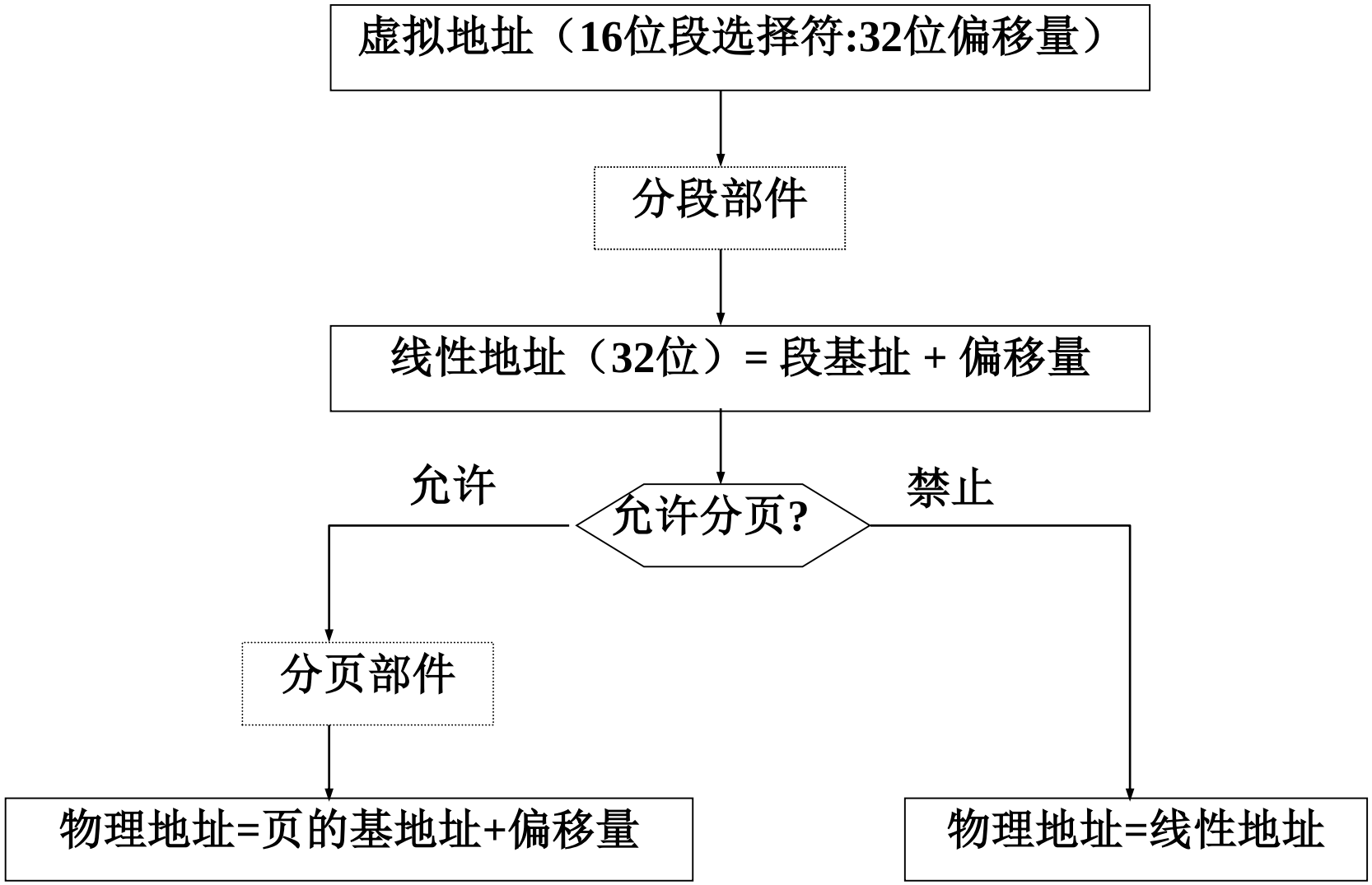
- A（Accessed）：访问标志。如果对某页表或页访问过，CPU设置页表项中的A位为1。操作系统定期扫描该项，统计使用次数。当需要调出（释放）页面时，操作系统一般选择那些最少使用的或长期不用的页。
- D（Dirty）：写入位。D=1时表示对该页进行过写操作，D=0时表示对该页还没有进行过写操作。D位只用于页描述符，它是写入标志。当需要释放某个物理页面时，若其D=0表示磁盘中交换页面的内容和物理页面的内容一致，不需要向磁盘重写；若D=1则需要将这个页面写到磁盘

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

- 一个物理页存在两级保护属性，一个是页表描述符中的保护属性，另一个是页描述符中的保护属性。在两级保护属性不一致的情况下，CPU从二者中取一个较严格的保护权限。例如，页表描述符的属性R/W = 0（只读），页描述符的属性R/W = 1（可读写），则最后结果是R/W = 0（只读）。


页目录
页表
物理页面
0CD34000
09C25000
00442000
00442070
09C250C8
00442047
0CD34020
PDBR
09C25067
0CD34000
008H×4=020H
032H×4=0C8H
070H
线性地址
02032070

- 地方

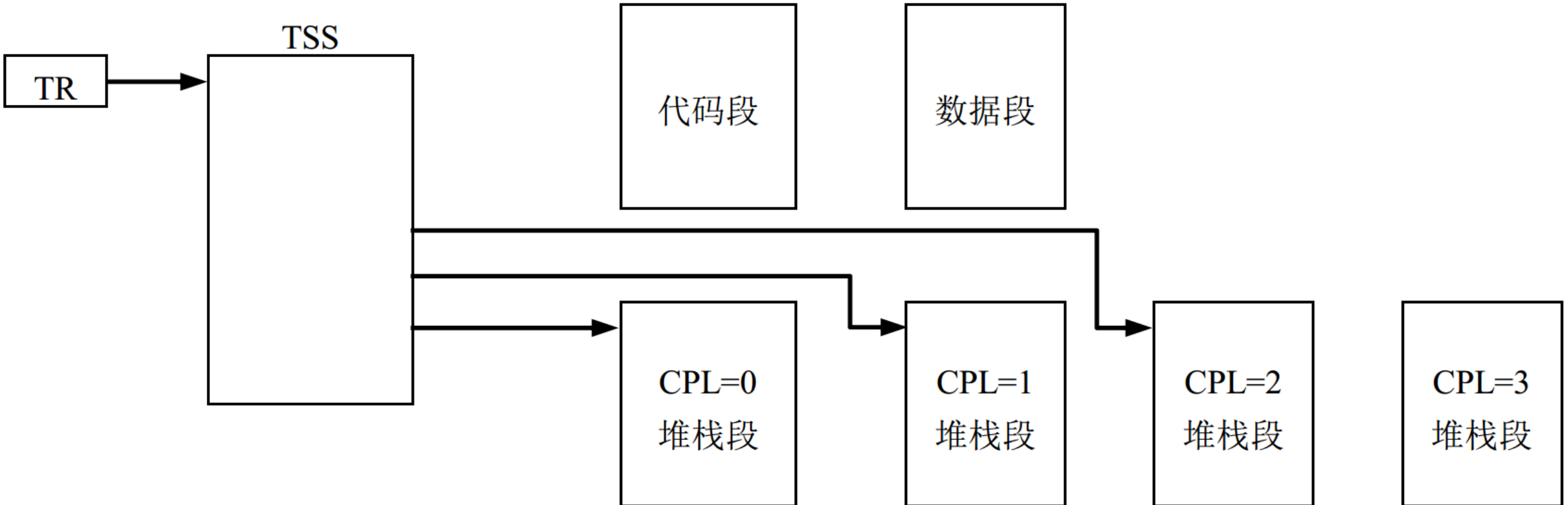

虚拟地址（16位段选择符:32位偏移量）
分段部件
线性地址（32位）= 段基址 + 偏移量
允许
允许分页?
禁止
分页部件
物理地址=页的基地址+偏移量
物理地址=线性地址

任务

- 在保护模式下每个任务是独立的，CPU提供了实现任务间快速切换的高效机制。在任何时刻都有一个当前任务，由TR寄存器指定，CPU在这个任务的环境下执行。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。
- 每个任务都由两个部分组成：任务执行环境（Task Executation Space）和任务状态段TSS（Task State Segment）。

任务执行环境

- 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。



- 每个任务都有一个LDT描述符表，构成一个局部地址空间，局部空间的数据和代码不能被其他任务访问。

任务状态段

- 任务状态段TSS（Task State Segment）中保存了任务的各种状态信息。它在任务切换过程中起着重要作用，通过它可以实现任务的挂起和恢复。
- 在任务切换过程中，首先CPU中各寄存器的当前值被自动保存到TR所指定的TSS（当前任务状态段）中；然后下一任务的TSS的选择符被装入TR；最后从TR所指定的TSS中取出各寄存器的值送到处理器的各寄存器中。

- 任务状态段在主存中的位置、大小和属性等信息由任务状态段描述符（即TSS描述符）来描述。TSS描述符属于系统描述符，S位等于0，它必须放在GDT表中，而不能放在LDT或IDT中。
- 在任务切换或执行LTR指令时，会将新的任务状态段选择符装载到TR寄存器。
- TSS的基本格式由104字节（000H～067H）组成。这104字节的基本格式是不可改变的，在此之外系统软件还可在TSS段中定义若干附加信息。基本的104字节可分为寄存器保存区域、内层堆栈指针区域、地址映射寄存器区域、域链接字段区域、I/O许可位图等。

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

- LDTR和CR3的值与特定任务相关，随着任务的切换，LDTR和CR3的值也要切换。TSS中保存了该任务的CR3和LDTR。
- 链接字段位于在TSS偏移0开始的双字中，其高16位未用，而低16位保存前一任务的TSS的选择符。如果当前的任务由段间调用指令CALL、中断/异常而激活，那么当前任务TSS中的链接字段保存被挂起任务的TSS的选择符，并且标志寄存器EFLAGS中的NT位被置为1，表示嵌套。
- I/O许可位图区域是为了实现输入/输出的保护。I/O许可位图作为TSS的扩展部分，TSS内偏移66H处的字用于存放I/O许可位图在TSS内的偏移（从TSS的头开始计算）。

门

- 4种：调用门、任务门、中断门、陷阱门
- 调用门用于控制传送，来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。
- 门描述符属于系统描述符

门描述符

	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）							

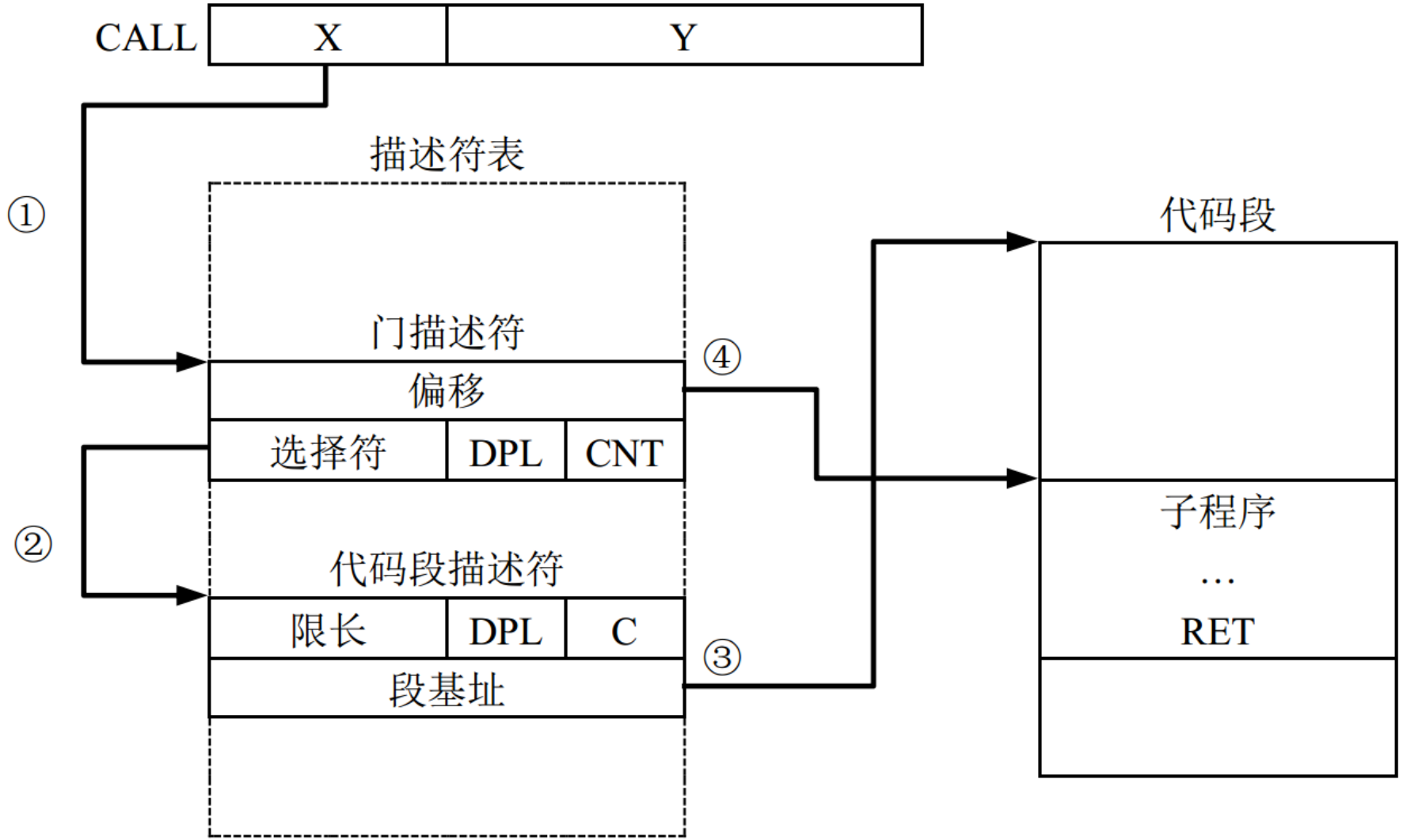
- 系统描述符中设置了一个类型（TYPE）字段，4位可表示16种类型

TYPE	含义	TYPE	含义
0	非法	8	非法
1	80286可用任务状态段(Available TSS)	9	80386可用任务状态段(Available TSS)
2	局部描述符表(LDT)	A	保留
3	80286忙任务状态段(Busy TSS)	B	80386忙任务状态段(Busy TSS)
4	80286调用门(Call Gate)	C	80386调用门(Call Gate)
5	任务门(Task Gate)	D	保留
6	80286中断门(Interrupt Gate)	E	80386中断门(Interrupt Gate)
7	80286陷阱门(Trap Gate)	F	80386陷阱门(Trap Gate)

调用门

- 调用门可以实现当前任务从低特权级到更高特权级的间接控制转移，它在更高级特权级的段中定义了一个入口点，该入口点的虚拟地址（目标选择符和偏移量）包含在调用门中。
- 调用门可以驻留在GDT中，也可以驻留在LDT中，但是不可在IDT中。
- 参数计数值表示有多少个（最多31个）参数必须从主程序（低特权级代码段）的堆栈复制到被调用子程序（高特权级代码段）的堆栈。将它乘以4后（32位系统）或者乘2（16位系统），得到参数在堆栈中的字节数。其他门描述符的参数计数值无意义。

使用调用门进行段间调用操作过程



- 调用指令“CALL X:Y”指令中虚拟地址X:Y，其中X是一个选择符，由它指向了一个调用门描述符（第①步），而Y的值不起作用。
- 调用门描述符中的选择符指向了一个段描述符（第②步），段描述符指出了被调用段的段基址（第③步），而入口点的偏移量就是门描述符中的偏移量（第④步），决定了调用哪一个代码段以及子程序在代码段中的偏移。
- 这种形式的CALL指令可以通过调用门转移到更高的特权级，在更高的特权级下执行所调用的子程序，子程序执行完毕后，由RET指令返回CALL指令所在的较低级别的程序。

任务门

- 任务门内的选择符必须指示GDT中的任务状态段TSS描述符，门中的偏移量无意义。任务的入口点保存在TSS中。利用段间转移指令JMP和段间调用指令CALL，通过任务门可实现任务切换。

+0	未使用							
+2	TSS段选择符(位15～0)							
+4	P	DPL	=0	0	1	0	1	未使用
+6	未使用							

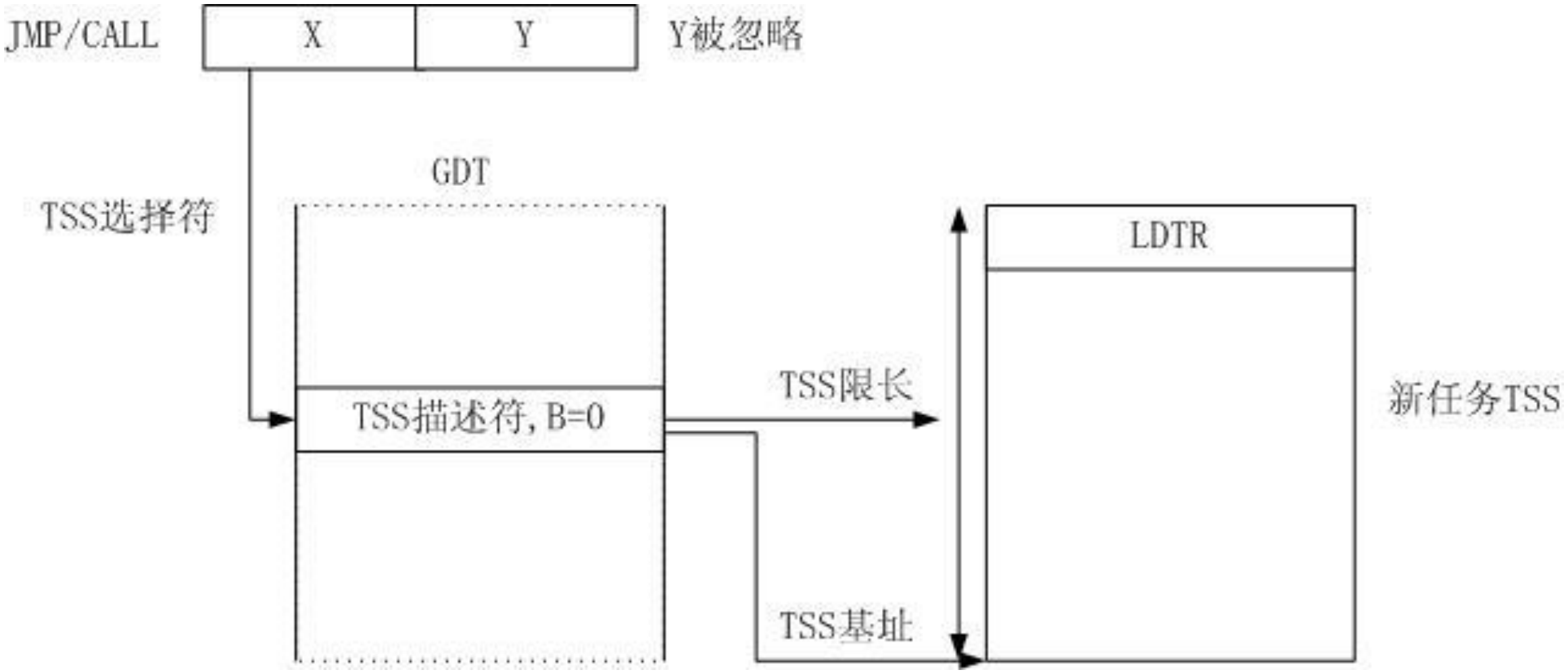
中断门和陷阱门

- 中断门和陷阱门描述中断/异常处理程序的入口点
- 中断门和陷阱门内的选择符必须指向代码段描述符，门内的偏移就是对应代码段的入口点的偏移。
- 中断门和陷阱门只有在中断描述符表IDT中才有效。

任务切换的场景

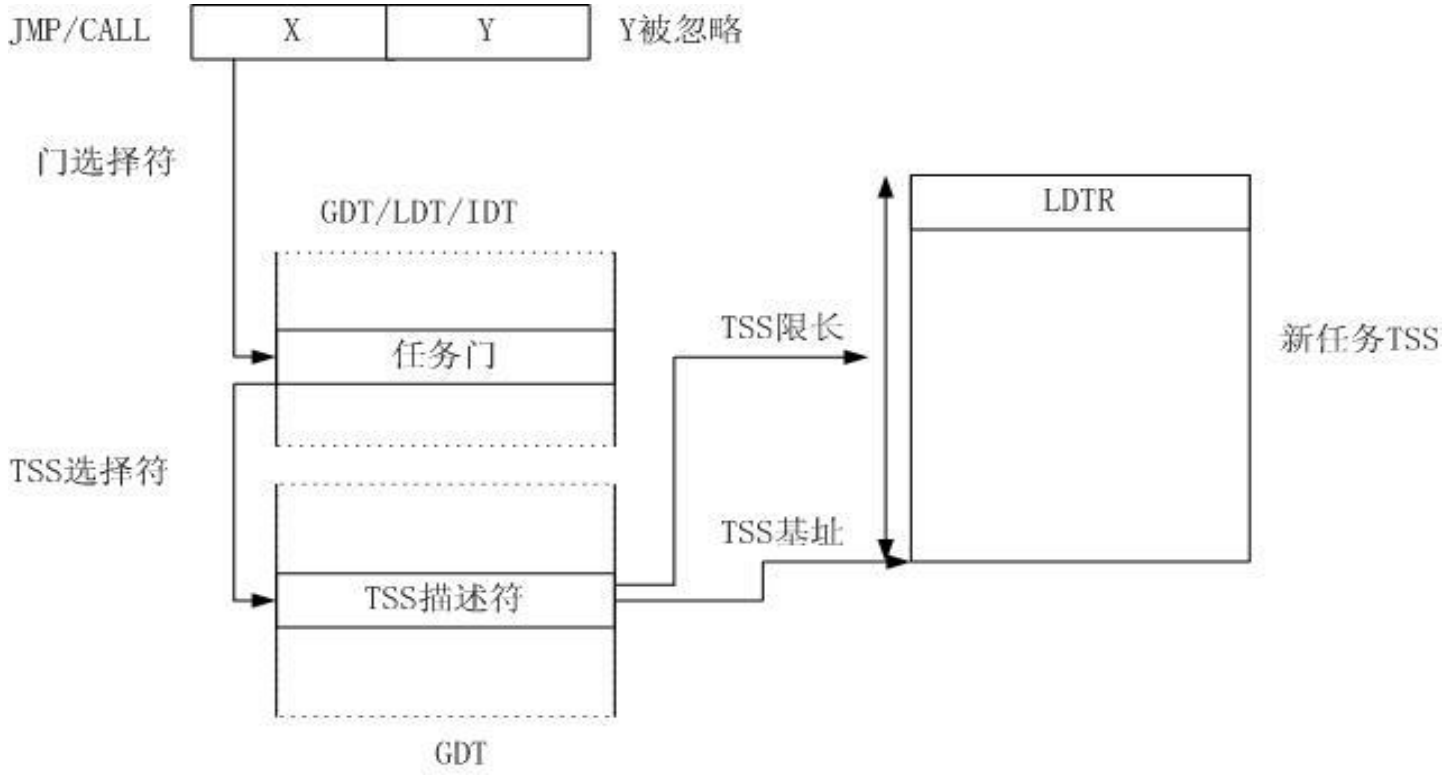
- 当前的程序、任务、过程执行远程JMP或者CALL指令，选择了GDT中的TSS描述符，此时指令中的偏移量忽略。
- 当前的程序、任务、过程执行远程JMP或者CALL指令，从GDT或者LDT中选择了任务门，目标地址的偏移量部分被忽略，新的TSS选择符在门中。
- 发生了中断或异常，中断向量选择了IDT中的任务门。新TSS选择符在门中。
- 当FLAGS中的NT=1时，执行IRET指令，目的任务选择符在执行IRET任务的TSS链接域中。

直接任务切换


JMP/CALL
X
Y
Y被忽略
GDT
TSS选择符
TSS描述符, B=0
TSS限长
LDTR
新任务TSS
TSS基址

间接任务切换

- 间接任务切换通过任务门来完成。利用段间转移指令JMP和段间调用指令CALL，通过任务门可实现任务切换。
- 任务的入口点保存在TSS中。



- 间接任务切换时，指令中的段寄存器内容为任务门描述符。
- 当使用JMP/CALL X:Y指令时，由X指向一个任务门描述符去查找GDT或者LDT；为中断响应时查找IDT，获得任务门。取出任务门中的TSS选择符，去GDT中查找获得新TSS描述符。注意新TSS的B位必须为0。
- JMP或CALL指令内的偏移（Y）没有被使用。
- 任务门的DPL（DPL_{GATE}）规定了访问该任务门的最低特权级，只有在同级或更高级别的程序中才可以访问它。即$DPL_{GATE} \geqslant MAX$（CPL，RPL），RPL是任务门选择符X的最后2位。

任务切换的步骤（概要）

- ①特权级检查;
- ②是否存在内存和限长检查;
- ③更新TR，并保存旧的TSS动态部分;
- ④加载TR，使新任务B=1，加载任务中各种寄存器内容。当CR0中TS被置位时表示任务切换完毕。

任务切换的步骤（详细）

- 假设：JMP X:Y或CALL X:Y指令进行任务切换时，假定从任务A切换到任务B，此时任务A为当前任务，B为目标任务
  - 保存寄存器现场到当前任务的TSS（任务A的TSS）。把通用寄存器、段寄存器、EIP及EFLAGS的当前值保存到当前TSS中。保存的EIP值是返回地址，指向引起任务切换指令的下一条指令。
  - 把目标任务（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。此后当前任务A改称为原任务A，目标任务B改称为当前任务B。
  - 恢复当前任务（任务B）的寄存器现场。根据任务B的TSS中各字段的值，恢复各通用寄存器、段寄存器、EFLAGS及EIP。

- 进行链接处理。由JMP指令引起的任务切换没有链接/解链处理；由CALL指令、中断引起的任务切换要实施链接处理；NT位为1时，由IRET指令引起的任务切换要实施解链处理。对于JMP或IRET指令，清除当前任务TSS中的B位；对于CALL、异常或中断引起的任务，置当前任务的B位为1。
- 把CR0中的TS标志置为1，这表示已发生过任务切换。
- 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。将LDT选择符装入LDTR中，从GDT中读出对应的LDT描述符，再把所读的LDT描述符装入LDTR高速缓冲寄存器。
- 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。

注意

- 在JMP指令引起任务切换时，不实施链接，不导致任务的嵌套，它要求目标任务是可用任务（B=0）。切换过程中把原任务置为“可用”（B=0），目标任务置为“忙”（B=1）。
- 在段间调用指令CALL或由中断/异常引起任务切换时，实施链接，导致任务的嵌套。它同样要求目标任务是可用的任务（B=0），在切换过程中把目标任务置为“忙”（B=1），原任务仍保持“忙”（B=1）；标志寄存器EFLAGS中的NT位被置为1，表示当前任务是嵌套任务。

任务内特权级变化时堆栈指针的使用

- 通过调用门向高特权级转移
- 只有在特权级变化时，才会切换堆栈。
- TSS中包含有指向0级、1级和2级堆栈的指针。（为什么？）
  - 还有SS和ESP
- 在特权级提升时，根据新的特权级使用TSS中相应的堆栈指针对SS及ESP寄存器进行初始化，建立起一个空栈。再把低特权级程序的SS及ESP寄存器的值压入堆栈，以使得相应的返回指令可恢复原来的堆栈。（为什么？）
  - 原特权级无法直接获取

- 从低特权级堆栈复制以双字为单位的调用参数到高特权级堆栈中，调用门中的CNT字段值决定了参数的个数。
- 通过复制栈中的参数，使高特权级的子程序可以访问主程序传递过来的参数。
- 与使用CALL指令通过调用门向高特权级转移相反，使用RET指令实现向低特权级转移。段间返回指令RETF从堆栈中弹出返回地址。
- 段间返回指令RETF从堆栈中弹出返回地址（CS:EIP）。选择符的RPL确定返回后的特权级。
- RET指令所使用的返回地址的选择符只能是代码段描述符，而不能是系统描述符或门描述符。与CALL指令不能向低特权级转移相对应，RET指令不能向高特权级转移。

保护

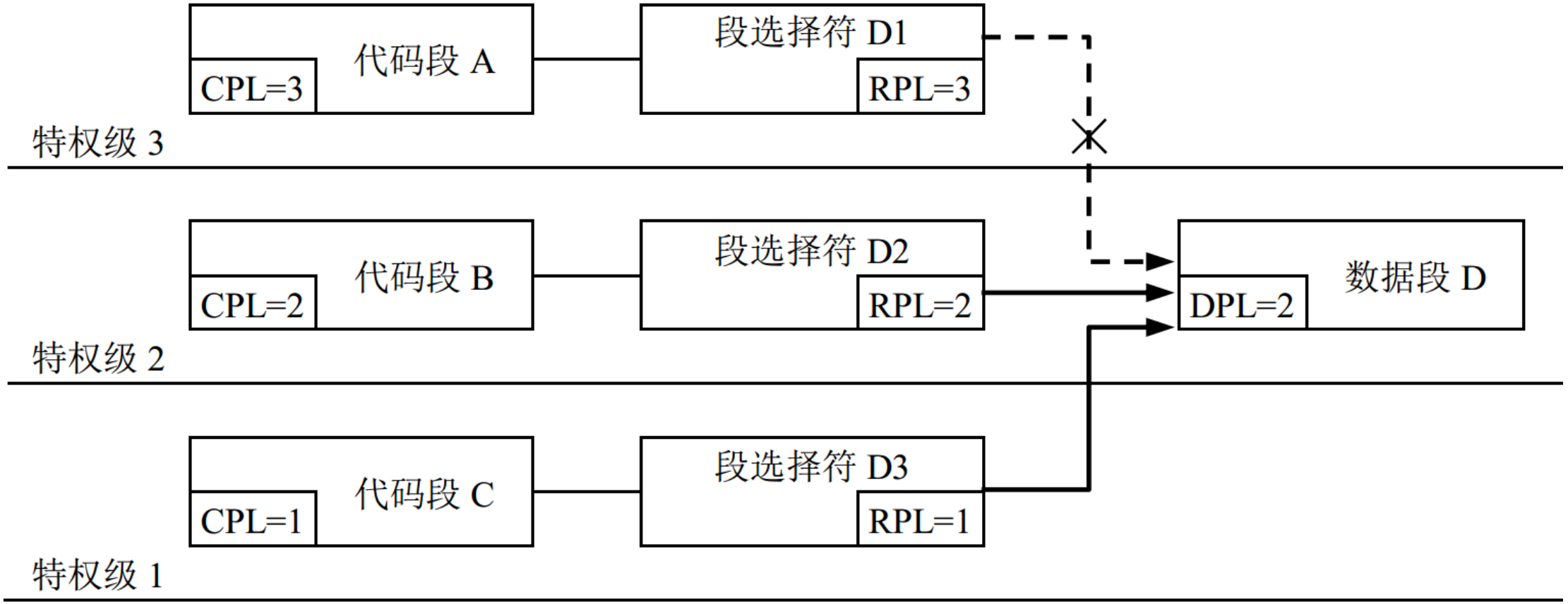
- 对数据的保护
- 对程序的保护
- 对输入输出的保护

数据访问保护

- 类型检查。装入CS时，段描述符中的E位必须为1，标记为可执行的段；装入DS、SS等寄存器时，E位必须为0。数据段描述符的W位为0时，不能对数据段进行写操作。代码段描述符的R位为0时，不能对代码段进行读操作。
- 限长检查
- 其他属性检查。主要是P位（存在位）

特权级检查

- DPL≥MAX（CPL，RPL），即程序只能访问特权级相同或者较低的数据

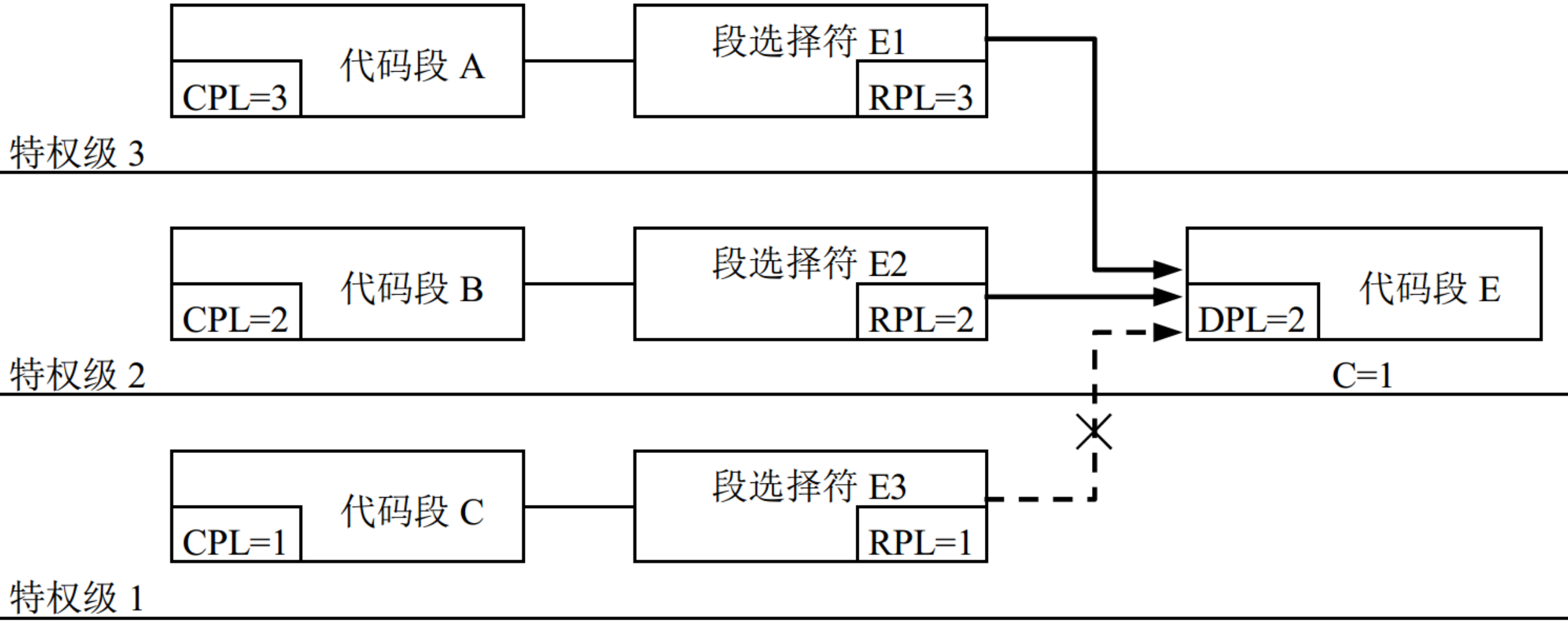


- 给SS赋值时，除了要满足DPL≥MAX（CPL，RPL）的条件之外，还必须满足RPL=CPL。也就是说SS的RPL总是等于当前执行程序的CPL。

对程序的保护-直接转移

- 直接转移：不涉及门，段内或者段间直接取得段描述符。
- 段内转移
  - 在同一代码段内转移时，显然转移前后不重新加载CS，特权级不发生变化，使用普通的跳转（JMP）或调用（CALL）指令即可。这时只需要检查限长
- 段间转移
  - 需使用远跳转或远调用指令，指令的格式如：“JMP X:Y”、“CALL X:Y”等，其中X是16位段选择符，Y是32位偏移。此时需要重新加载CS寄存器，处理器除了要检查限长以外，还要执行特权检查。

权限检查

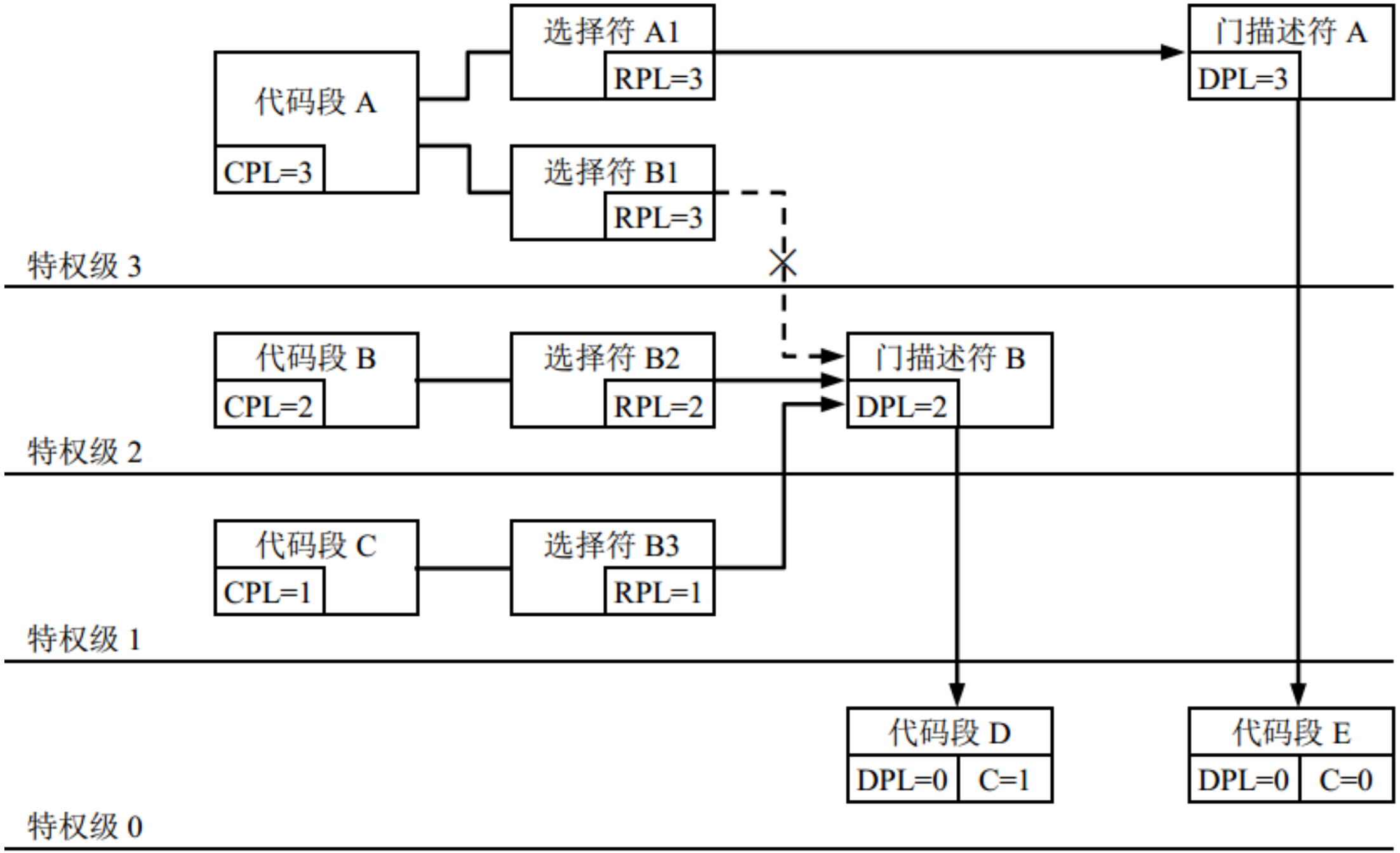

代码段 A
CPL=3
段选择符 E1
RPL=3
特权级 3
代码段 B
CPL=2
段选择符 E2
RPL=2
代码段 E
DPL=2
C=1
特权级 2
代码段 C
CPL=1
段选择符 E3
RPL=1
特权级 1

间接转移的保护

- 当CPL > DPL_{TSS}时，就不能采用直接切换，必须通过任务门进行任务切换。
- 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，启动新的任务。
- 涉及到的特权级
  - 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

间接转移条件-使用CALL指令

- $DPL_{GATE} \geqslant MAX（CPL，RPL）$
- $DPL_{CODE} \leqslant CPL$
- 在满足前面两条特权级要求的情况下：
  - 如果C_{CODE}为1，表示一致代码段，则当前特权级不变；
  - 如果C_{CODE}为0，则当前特权级提升为DPL_{CODE}（$DPL_{CODE} < CPL$）或者维持不变（$DPL_{CODE} = CPL$）。


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

间接转移条件-使用JMP指令

- $DPL_{GATE} \geqslant MAX$（CPL，RPL）
- C_{CODE}为1且$DPL_{CODE} \leqslant CPL$ 或 C_{CODE}为0且$DPL_{CODE} = CPL$
- JMP指令只能转移到同级的代码，而不能转移到更高的特权级。

输入输出保护

- CPU采用I/O特权级IOPL（I/O Privilege Level）和TSS段中I/O许可位图的方法来控制输入/输出，实现对应用程序I/O指令的限制。
- CPL≤IOPL时，可以执行I/O敏感指令。
- 任务状态段中的I/O许可位图也影响I/O敏感指令的执行。I/O许可位图规定了I/O空间中的哪些地址可以由该任务的任何特权级程序所访问，而不受IOPL的限制。

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

- 当前特权级CPL比IOPL高或者相同时，可以正常执行所有I/O敏感指令；
- CPL特权级比IOPL特权级低时，执行CLI和STI指令将引起通用保护异常，而其他4条指令是否能够执行要根据访问的I/O地址及任务的I/O许可位图来决定，如果条件不满足时，那么将引起保护异常。

I/O许可位图的保护

- 只用IOPL来限制I/O指令的执行会使得在特权级3执行的应用程序要么可以访问所有I/O地址，要么不能访问所有I/O地址，使用很不方便。
- 在任务状态段中设置了I/O许可位图。每个任务的I/O许可位图都可以由操作系统来设置。I/O许可位图就是一个二进制位串。位串中的每一位对应一个I/O地址，位串的第0位对应I/O地址0，位串的第n位对应I/O地址n。如果位串中的第n位为0，那么I/O地址n就可以由任何特权级的程序访问，而不加限制；如果第n位为1，I/O地址n只能由在IOPL特权级或更高特权级运行的程序访问

I/O许可位图的保护

- 如果一条I/O指令涉及多个I/O地址，例如“IN EAX, DX”涉及4个I/O地址，则在检查许可位时，这条指令用到的所有I/O地址的许可位都必须为0才允许访问。
- Intel 80x86系列计算机的I/O地址空间范围是0000H ~ 0FFFFH，所以I/O许可位图的二进制位串最大为8KB。
- TSS内偏移66H的字确定I/O许可位图在TSS段中的位置，即I/O位图基址。I/O许可位图的大小s等于TSS段的长度减去I/O位图基址，它定义了I/O地址空间0 ~ s×8–1的许可位。CPU认为大于或等于s×8的I/O地址许可位全部为1。

I/O许可位图的保护

- 由于I/O许可位图最长可达8KB，所以开始偏移应小于56KB，但必须大于等于104B，因为TSS中前104字节为TSS的固定格式，用于保存任务的状态。

对IOPL的保护

- 只有在特权级0下执行的程序才可以修改IOPL位及VM位；只有IOPL级或更高的特权级的程序才可以修改IF位。指令POPF不能改变VM位，而PUSHF指令所压入的标志中的VM位总为0。