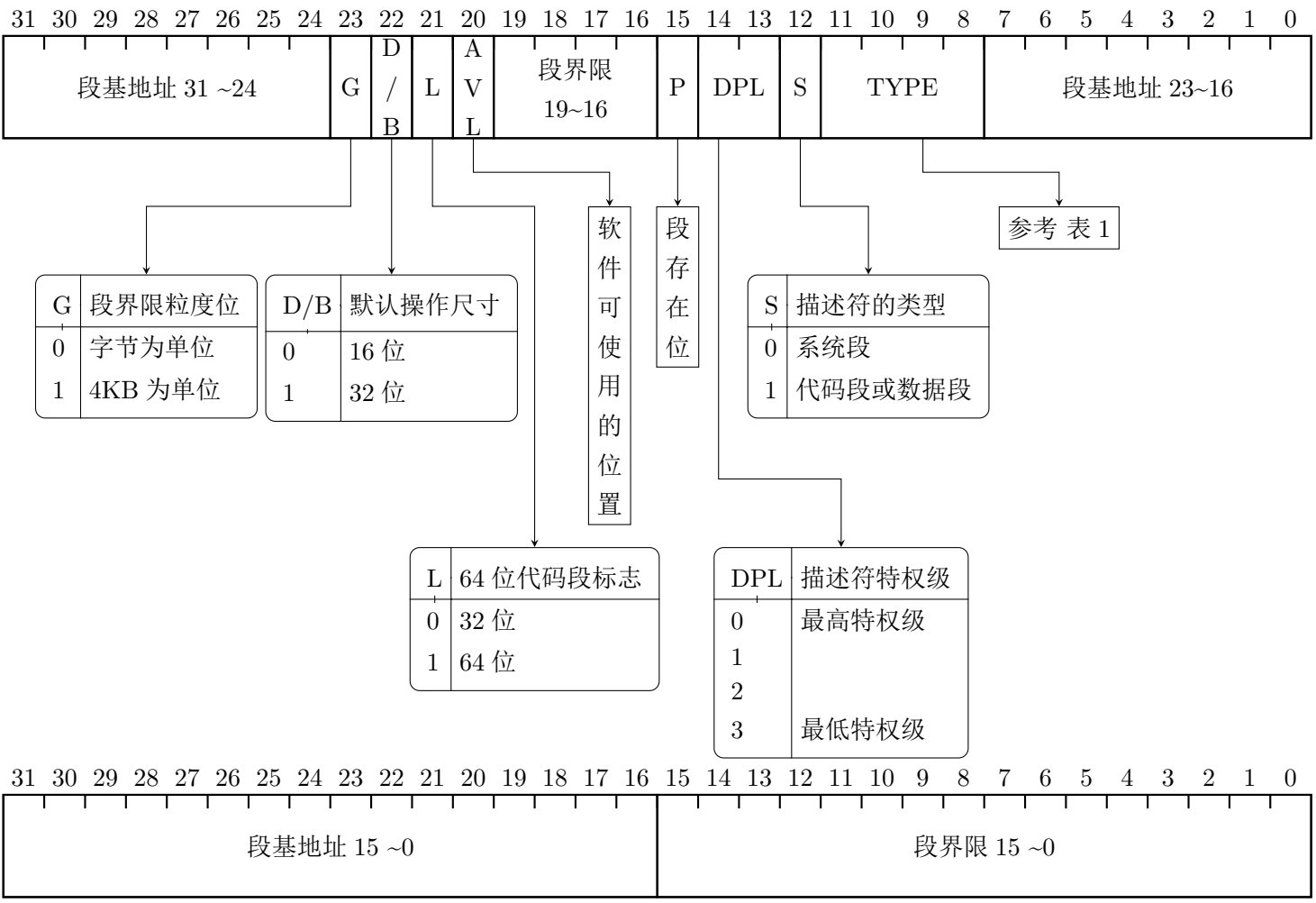

图 1: 段描述符

X	E	W	A	描述符类别	含义
0	0	0	X	数据	只读
0	0	1	X		读、写
0	1	0	X		只读、向下扩展
0	1	1	X		读、写、向下扩展
X	C	R	A	描述符类别	含义
1	0	0	X	代码	只执行
1	0	1	X		只执行
1	1	0	X		只执行、依从的代码段
1	1	1	X		执行、读、依从的代码段

表 1: 代码段和数据段描述符的 TYPE 字段

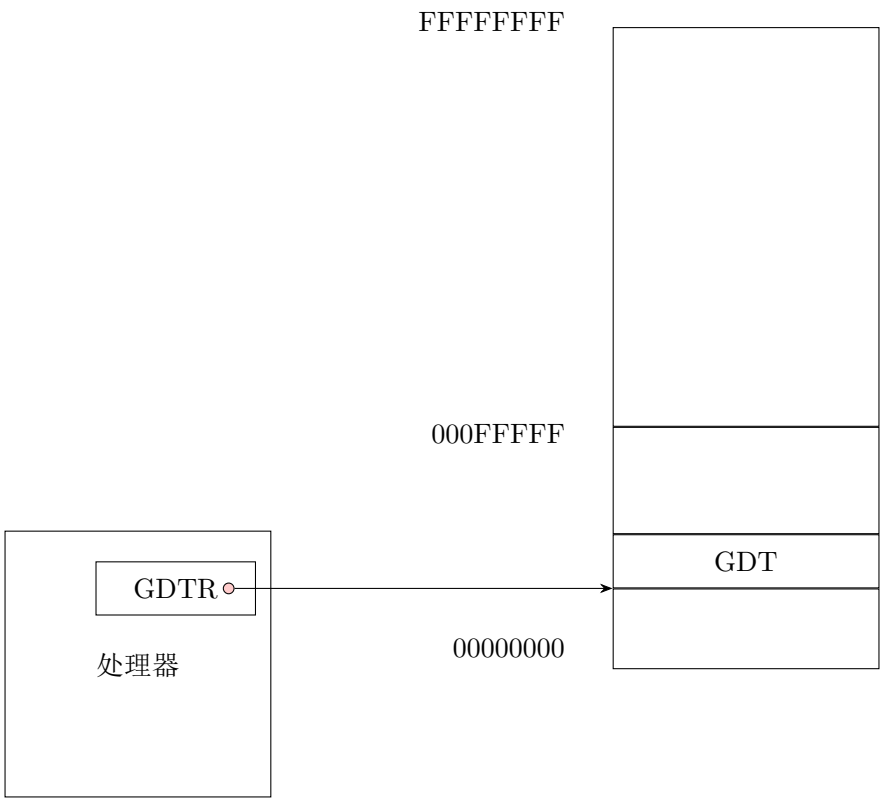

图 2: GDT 和 GDTR 的关系

图 3: 全局描述符表寄存器 GDTR

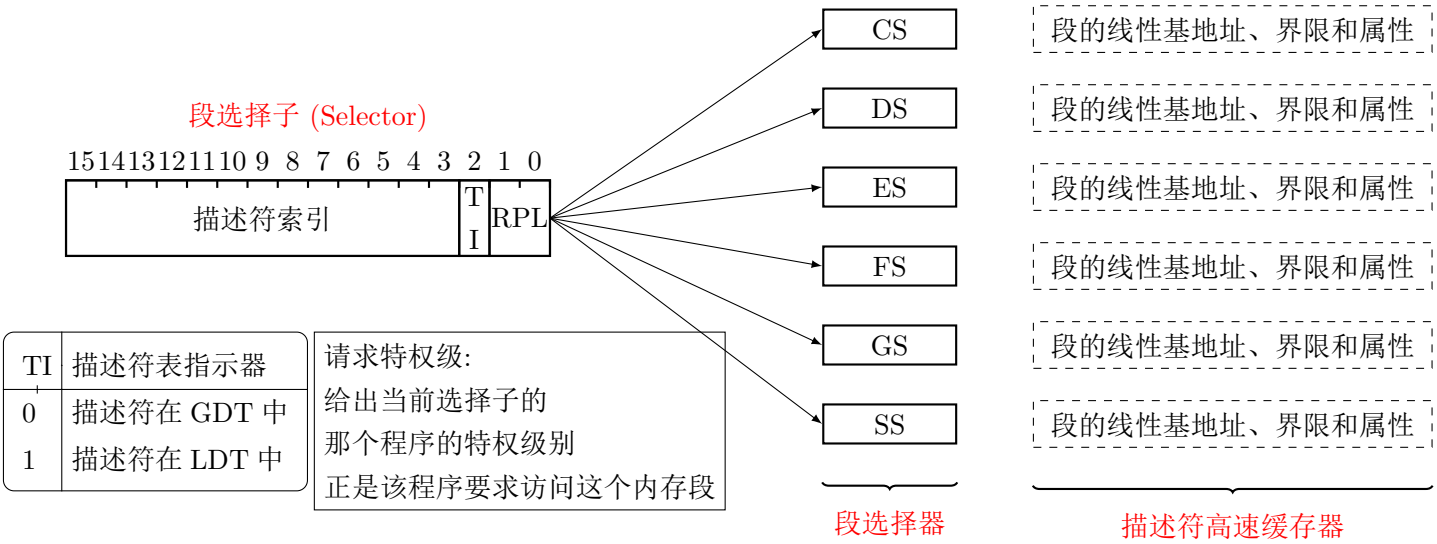

图 4: 段寄存器与段选择子

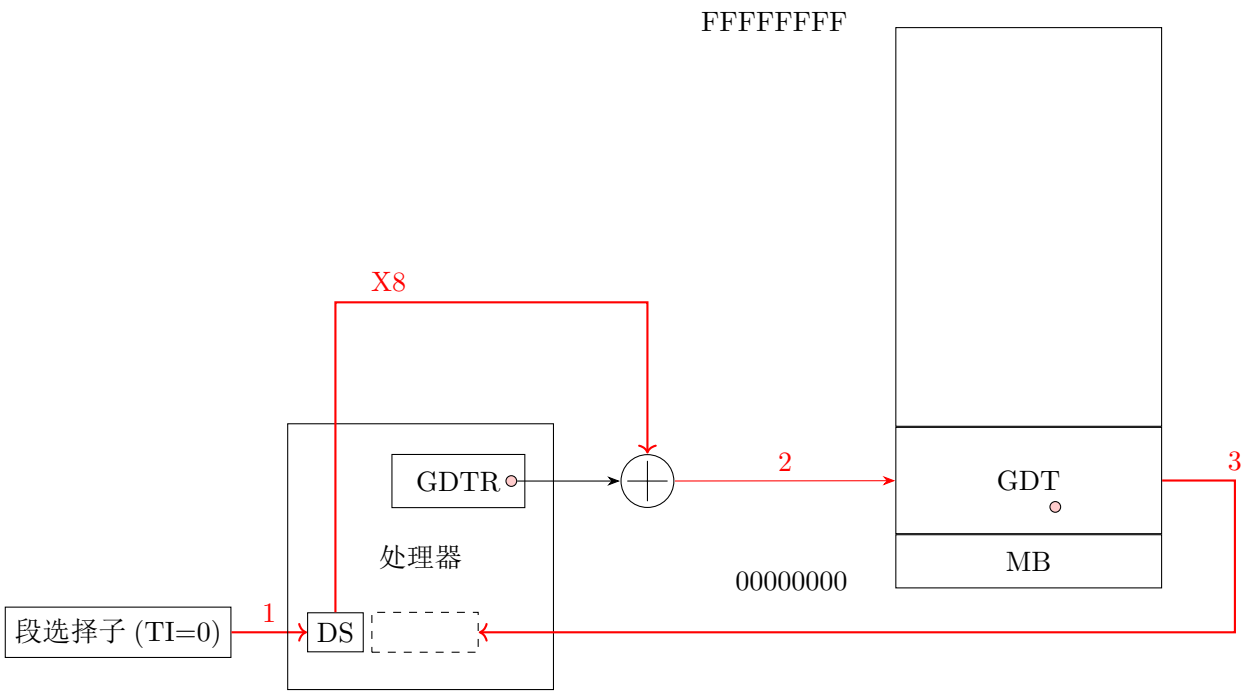

图 5: 段选择器和描述符高速缓存器的加载过程

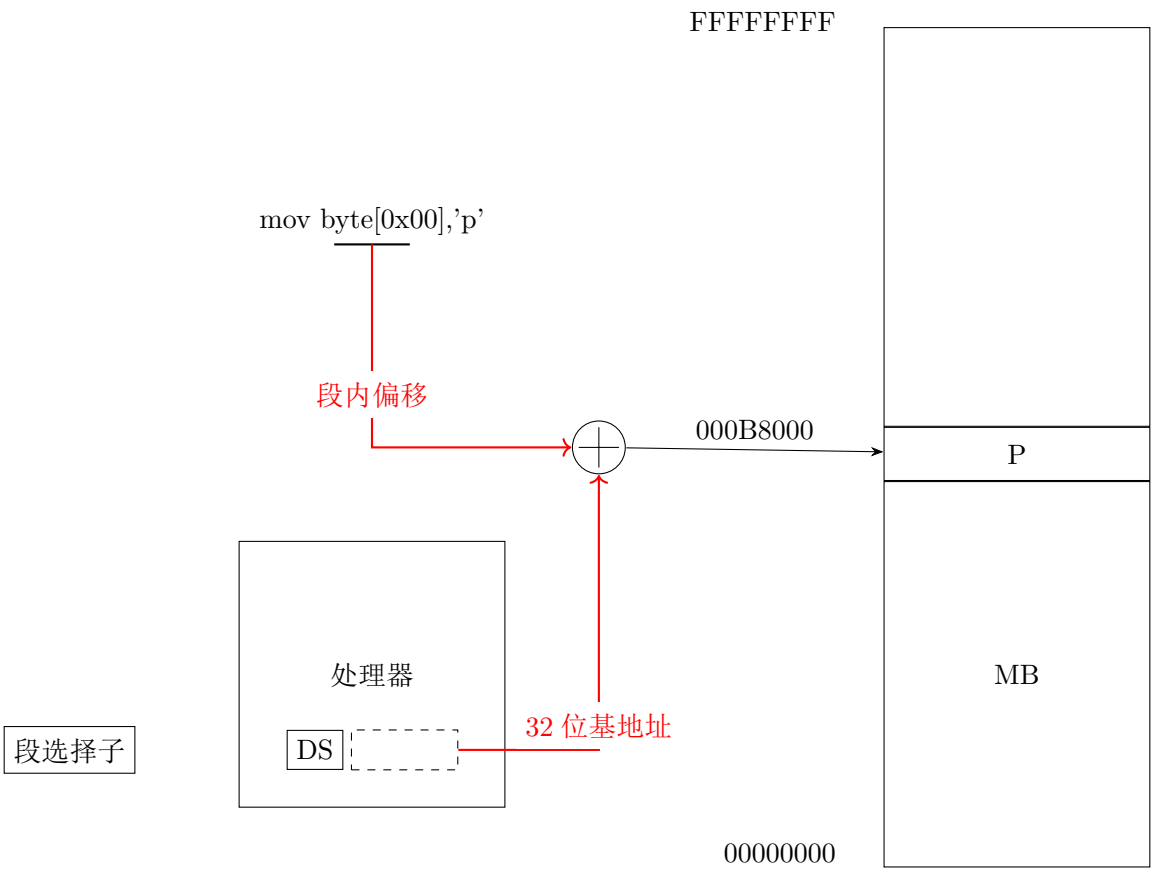

图 6: 保护模式下的内存访问

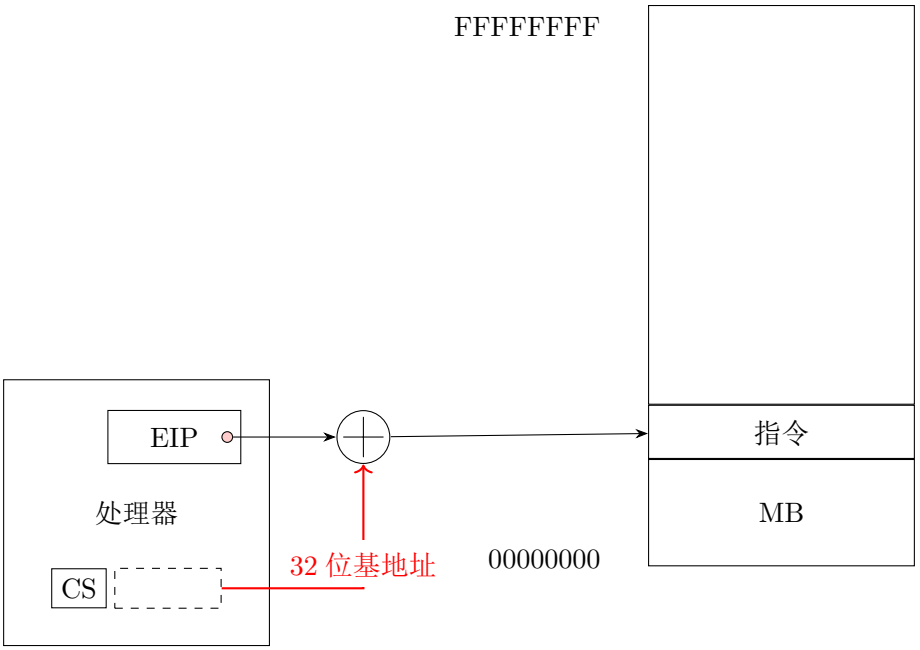

图 7: 保护模式下处理器取指令的过程

```
.text
entry start
start:
mov ax,cs
mov ss,ax
mov sp,#0x7c00
```

初始化栈, 参考保护模式前的内存映像图 8

`mov ah,#0x00`
`mov al,#0x03`
`int 0x10`
	bios 清屏功能

`seg cs`
`mov ax, gdt_base+0x7c00`
`seg cs`
`mov dx, gdt_base+0x7c00 + 0x02`
`mov bx,#16`
`div bx`
	div bx
	$$\frac{DX:AX}{BX} = AX.DX$$
	商在 AX 中, 余数在 DX 中
	在此示例中:
	将 GDT 的基地址右移 4 位得到段地址存到 AX 中，段内偏移地址存到 dx 中

`mov ds,ax` `mov bx,dx`	将 gdt 基地址的段地址存到 ds, 偏移地址存到 bx 中
`mov dword 0(bx),#0x00` `mov dword 4(bx),#0x00`	GDT 的第一个描述符为空描述符
`mov dword 0x08(bx),#0x8000ffff` `mov dword 0x0c(bx),#0x0040920b`	段线性地址=0x000B8000 G=0: 段粒度为字节 段界限=FFFF: 结合 G=0，该段的长度为 64KB S=1: 存储器的段 D=1：32 位操作尺寸 P=1: 该段目前位于内存中 DPL=000: 段的特权级为 0 TYPE=0010: 可读可写、向上扩展的数据段
`seg cs` `mov word gdt_size + 0x7c00,#15`	设置 gdt 的边界
`seg cs` `lgdt gdt_size+0x7c00`	ldgt m：加载描述符表的线性基地址和界限 在有效地址 m 处，包含了 GDT 的 32 位线性地址和 16 位界限值, 共 6 字节, 参考图 3

`in al,#0x92`
`or al,#0x02`
`out 0x92,al`
`cli`
	打开 A20

`mov eax,cr0`
`or eax,#1`
`mov cr0,eax`
	开启保护模式

`mov cx,#0x08` `mov ds,cx`	将段选择子传到段选择器，参考图 5

mov byte 0x00,#0x50 mov byte 0x02,#0x72 mov byte 0x04,#0x6F mov byte 0x06,#0x74 mov byte 0x08,#0x65 mov byte 0x0a,#0x63 mov byte 0x0c,#0x74 mov byte 0x0e,#0x20 mov byte 0x10,#0x6D mov byte 0x12,#0x6F mov byte 0x14,#0x64 mov byte 0x16,#0x65 mov byte 0x18,#0x20 mov byte 0x1A,#0x4F mov byte 0x01C,#0x4B mov byte 0x1E,#0x2E	将数据传到到段描述符指定的线性地址加偏移量处
hlt	cpu 暂停
gdt_size: .word 0	存放 GDT 的界限值
gdt_base: .word 0x7e00,0x0000	声明 GDT 的起始物理地址
.org 510 .word 0xAA55	cpu 暂停

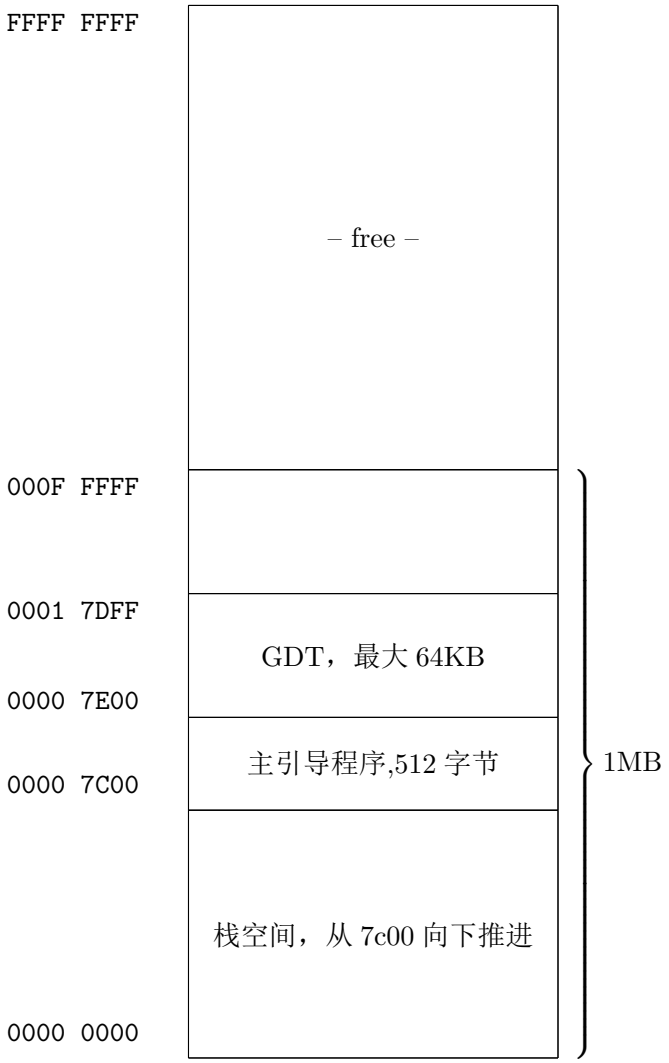

图 8: 进入保护模式前的内存映像