

# 第二章

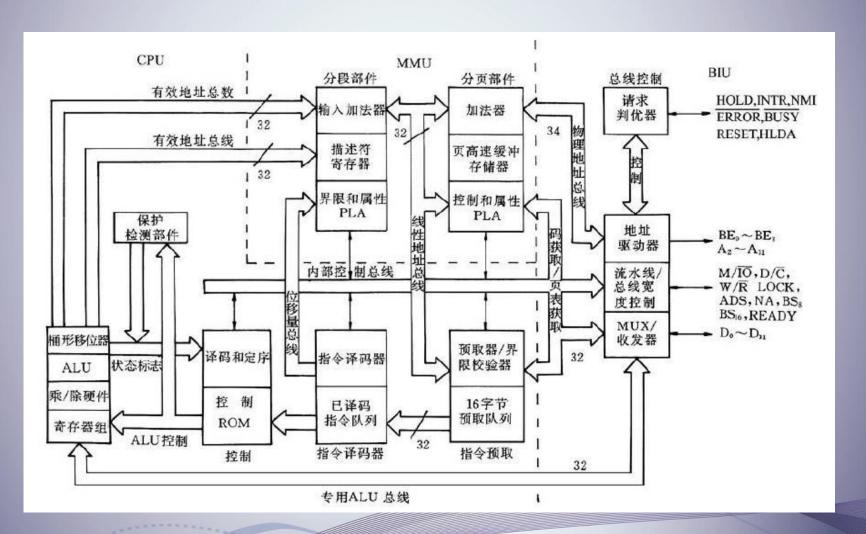
微处理器管理模式

#### 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

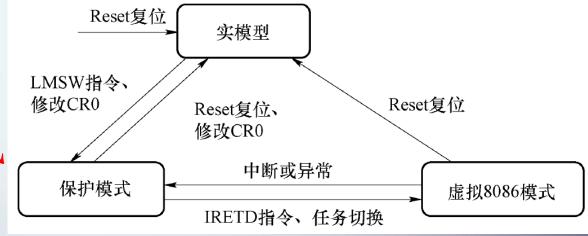


#### 1、微处理器基本结构



#### 2、CPU工作模式

- · 从80386开始, Intel的CPU具有3种运行模式:
- 通常所说的保护模式是指80386处理器及以后的处理器所实现的运行模式。
  - 实模式
  - 保护模式
  - 虚拟8086模式



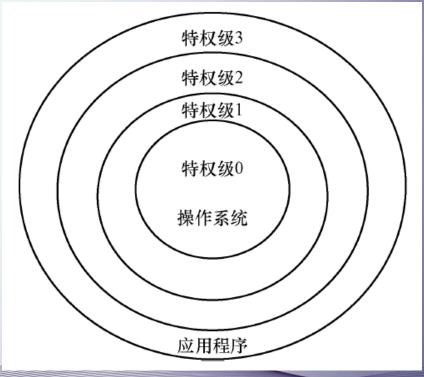
## 实模式

- · 在实模式下,80X86等同于一个16位的8086微 处理器。
- 采用类似于8086的体系结构,其寻址机制、中断处理机制均和8086相同;只使用低20位地址线,只能寻址1M字节的物理地址空间。
- 不支持硬件上的多任务切换;不支持特权级,可以执行所有特权指令。
- · DOS-实模式下; Windows/Linux-保护模式。



#### 特权级

- 在保护模式下, CPU有4个特权级, 分别为特权级0、1、2、3。
- -操作系统运行在高的特权 级(Ring 0)上,而应用 程序运行在低的特权 (Ring 3)上,应用程序 不能直接修改(破坏)操 作系统的程序和数据。



## 保护模式

- · CPU提供了段式和页式内存管理功能,支持多任务和特权级。
- 物理寻址空间高达4GB(80386/80486)或64GB (Pentium及以上CPU)。

## 虚拟8086模式

- · 虚拟8086模式又称"V86模式"。
- 32位80X86处理器给每个以"虚拟8086模式"运行的任务"创造"了一个与真实的8086处理器十分相似的运行环境,以便运行DOS程序。
- 既有保护功能又能执行8086代码的工作模式。

#### 3、寄存器

- 程序可见寄存器组
  - -实模式和保护模式下
- 程序不可见寄存器组/特殊寄存器
  - -保护模式下

#### 程序可见寄存器组

- •程序可见寄存器组包括多个8位、16位和32位寄存器。
  - 通用寄存器
  - -段寄存器
  - -控制寄存器

#### 程序可见寄存器组



## 段寄存器

CS 代码段寄存器

**SS** 堆栈段寄存器

DS 数据段寄存器

ES 附加段寄存器

FS

GS

段寄存器



#### 通用寄存器---数据寄存器

• 数据寄存器

-8位: AL AH BL BH CL CH DL DH

- 16位: AX BX CX DX

- 32位: EAX EBX ECX EDX [386]

#### 通用寄存器---指针寄存器[1]

- 堆栈指针寄存器SP、ESP (386以上):
- · 功能: 存放当前堆栈段栈顶偏移量, 总是与 SS堆栈段寄存器配合存取堆栈中的数据。

· 说明: 实模式使用SP; 保护模式使用ESP。

#### 通用寄存器---指针寄存器[2]

- · 基址指针寄存器BP、EBP(386以上):
- 功能: 存放地址的偏移量部分或数据。若 存放偏移量时, 缺省情况与SS配合。

· 说明: 实模式使用BP; 保护模式使用EBP。

## 通用寄存器---变址寄存器

- 变址寄存器SI、DI、ESI、EDI:
- 功能: 存放地址的偏移量部分或数据。若 存放偏移量时, 缺省情况与DS配合。

· 说明: 实模式使用SI、DI; 保护模式使用 ESI、EDI。

#### 通用寄存器说明

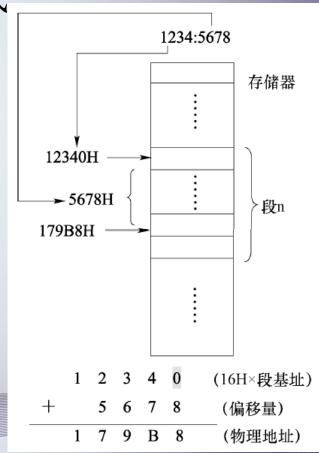
• 注意:除SP、ESP堆栈指针不能随意修改、需要慎用外,其它通用寄存器都可以直接在指令中使用,用以存放操作数,这是它们的通用之处,其它专用用途在具体指令中介绍。

#### 段寄存器

- · 简介: 实模式下的存储器采用分段管理方法组织, 因此一个物理地址用段基址和偏移量表示。一个程序可以由多个段组成。
- 段寄存器功能:段寄存器存放段基址。在 实模式下存放段基地址,在保护模式下存 放段选择子。

## 实模式内存分段管理

- 8086/8088 CPU有20位地址总线
  - 最大寻址范围1MB/0000~FFFFF
- 8086/8088 CPU寄存器是16位
  - 数据表示范围0000~FFFF
- 分段方式对存储器访问
  - -10H×段基址+偏移量
  - 段最大为64KB
  - 定位在16的整数倍上





#### 段寄存器[1]

- 代码段寄存器CS: 指定当前代码段, 代码段中存放当前正在运行的程序段。
- · 堆栈段寄存器SS: 指定当前堆栈段。
- 说明: 堆栈段是在内存开辟的一块特殊区域 , 其中的数据访问原则是后进先出(LIFO) , 允许插入和删除的一端叫做栈顶。IBM PC 机中SP(或ESP)指向栈顶, SS指向堆栈段 基地址。

#### 段寄存器[2]

- 数据段寄存器DS: 指定当前运行程序所使 用的数据段。
- 附加数据段寄存器ES: 指定当前运行程序 所使用的附加数据段。
- 段寄存器FS和GS: 指定当前运行程序的另外两个存放数据的存储段(只对80386以上机器有效)。

#### 段寄存器说明

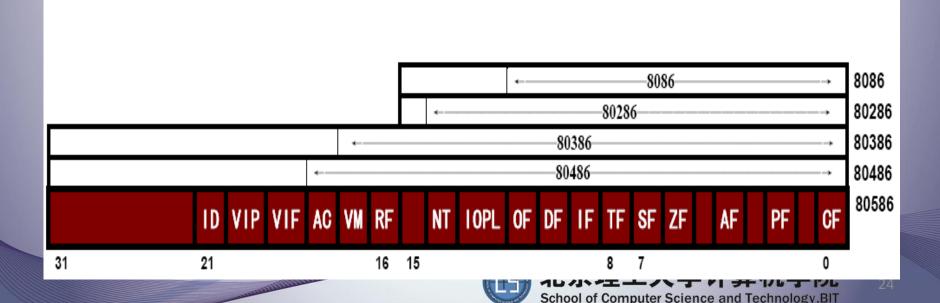
• 说明:虽然DS、ES、FS、GS(甚至于CS、SS)所指定的段中都可以存放数据,但DS是主要的数据段寄存器,在默认情况下使用DS所指向段的数据。若要引用其它段中的数据,通常需要显式说明。

## 控制寄存器

- 控制寄存器包括指令指针寄存器和标志寄存器。注意: 在程序中不能直接引用控制寄存器名。
  - -指令指针寄存器IP/EIP
    - 总是与CS段寄存器配合指出下一条要执行指令的地址, 其中存放偏移量部分。实模式使用IP, 保护模式使用EIP。
  - -标志寄存器(FLAGS/EFLAGS)

#### 标志寄存器FLAGS

- 标志寄存器:被称为状态寄存器,由运算 结果特征标志和控制标志组成。
- 重点:各档CPU均有的标志。即8086拥有的9个标志。8086/286为16位,以上为32位。



#### 标志寄存器

• 运算结果特征标志:用于记录程序中运行结果的特征。8086~Pentium的标志寄存器均含有CF、PF、AF、ZF、SF、OF这6位标志。

#### FLAGS---CF

- CF: 进位标志,记录运算结果的最高位向前产生的进位或借位。可用于检测无符号数运算时是否发生溢出。
  - -CF=1 有进位或借位
  - -CF=0 无进位或借位

#### FLAGS---PF

- PF: 奇偶标志,记录运算结果最低8位中含1的个数。可用于检测数据传送过程中是否发生错误。
  - -PF=1 个数为偶数
  - -PF=0 个数为奇数

#### FLAGS---AF

- AF: 辅助进位标志,记录运算结果最低4 位向前产生的进位或借位。只有在执行十 进制运算指令时才关心此位。
  - -AF=1 有进位或借位
  - -AF=0 无进位或借位

#### FLAGS---ZF / SF

- ZF: 零标志,记录运算结果是否为0。
  - -ZF=1 运算结果为零
  - -ZF=0 结果非零
- SF: 符号标志, 记录运算结果的符号。
  - -SF=1 运算结果为负
  - -SF=0 结果非负

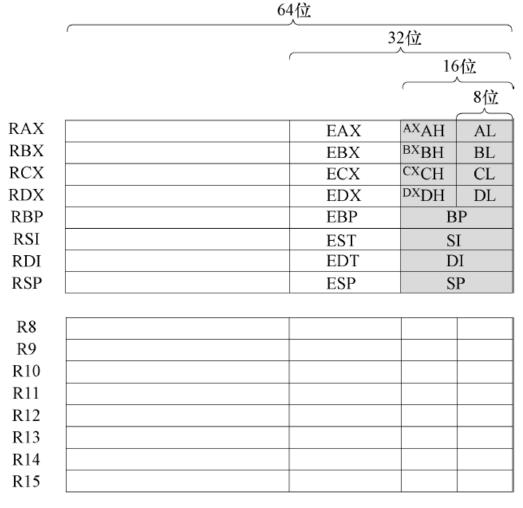
#### FLAGS---OF

- OF: 溢出标志,记录运算结果是否超出了机器所能表示的范围。可用于检测带符号数运算时是否发生溢出。
  - -OF=1 运算结果超出范围
  - OF=0 结果未超出

#### FLAGS---控制标志

- 控制标志: 控制标志控制处理器的操作,要通过专门的指令才能使控制标志发生变化。
  - IF: 中断允许标志。IF的控制只对外部可屏蔽中断请求(INTR)起作用。
    - IF=1/0 允许/禁止 CPU响应INTR
  - DF: 方向标志。专服务于字符串操作指令, 指示串操作时操作数地址的增减方向。
    - DF=1/0 串操作时操作数地址为自动减/增量
  - TF: 陷阱标志。用于程序调试。
    - TF=1/0 CPU处于单步/连续方式

## 保护模式下的 可见寄存器组 —64位



RFLAGS	EFLAGS	FLAGS
RTP	EIP	IP

CS	
Г	S
S	S
Е	S
C	iS
F	S

## 保护模式下通用寄存器

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操作数	RSI	ESI	SI	无

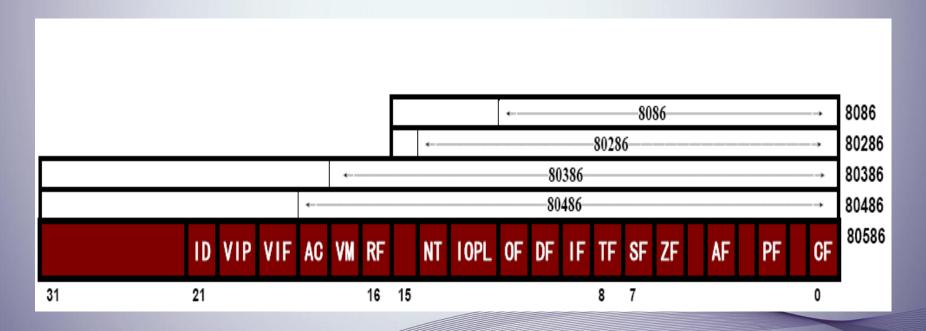
#### 保护模式下的指令堆栈寄存器EIP/ESP

- · 指令指针寄存器EIP和堆栈指针寄存器ESP
  - EIP和ESP都是32位寄存器,它们的低16位分别是IP和SP。

	31 16	6 15 0
EIP ESP		IP
		SP

# 保护模式下的标志寄存器EFLAGS

· 标志寄存器 标志寄存器EFLAGS也扩展为32位





# 保护模式下的标志寄存器EFLAGS

EFLAGS中的位11到位0和8086完全相同。这里主要扩展了4个标志位:

- VM: v86模式位,=1工作在v86模式,=0工作在实模式或保护模式
- -RF:恢复位,与程序调试有关的一个控制位。
- -NT: 嵌套任务位,=1当前任务嵌套在其他任务中
- IOPL: 占2位, 规定了哪些特权级的程序可以执行 I/O指令



## FLAGS--- I OPL/NT [80286]

• IOPL (I/O Privilege Level):特权标志,占两位。当在保护模式工作时,IOPL指定要求执行I/O指令的特权级。

• NT: 嵌套任务标志。保护模式在执行中断返回指令IRET时要测试NT值。

# FLAGS---RF/VM[80386]

- RF: 重启动标志。该标志控制是否接受调 试故障。
  - -RF=0时接受; RF=1时忽略。

· VM:虚拟方式标志。当CPU处于保护模式时,若VM=1则切换到虚拟模式,以允许执行多个DOS程序;否则CPU工作在一般的保护模式。

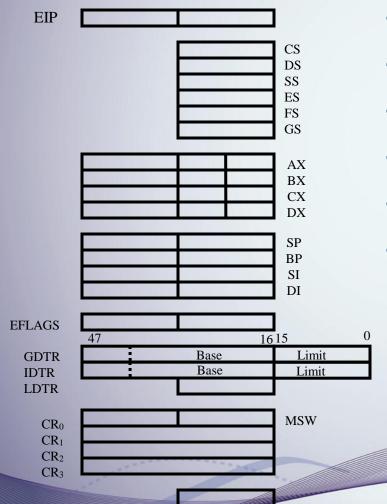
## FLAGS---AC[80486]

- AC: 地址对齐检查标志。若AC=1时进行地址对齐检查。若AC=0时不进行地址对齐检查。
  - 所谓地址不对齐是指以下情形: 一个字从奇地址开始, 或一个双字不是从4的倍数的地址开始。

## FLAGS---[Pentium]

- ID:标识标志。若ID=1,则表示Pentium 支持CPUID指令,CPUID指令给系统提供 Pentium微处理器有关版本号及制造商等信息。
- · VIP: 虚拟中断挂起标志。与VIF配合,用于多任务环境给操作系统提供虚拟中断挂起信息。
- VIF: 虚拟中断标志, 是虚拟方式下中断标 志位的映像。

# 保护模式下的特 殊寄存器



- GDTR: global descriptor table
- LDTR: local descriptor table
- IDTR: interrupt descriptor table
- TR: task register
- EIP: 32 bits in length
- EFLAGS
- CR<sub>0</sub>, CR<sub>1</sub>, CR<sub>2</sub>, CR<sub>3</sub>
- DR<sub>0</sub>-DR<sub>7</sub>(Debug registers)
- TR<sub>6</sub>-TR<sub>7</sub>(Test registers)





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

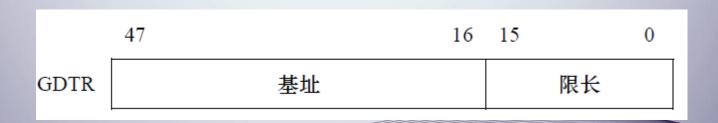
# 保护模式下的特殊寄存器

- 保护模式增加了四个寄存器:
  - -全局描述符表寄存器GDTR
  - 中断描述符表寄存器IDTR
  - 局部描述符表寄存器LDTR
  - -任务寄存器TR
  - -控制寄存器CR0~CR3
  - -调试寄存器DR0~DR7
  - 测试寄存器TR6~TR7



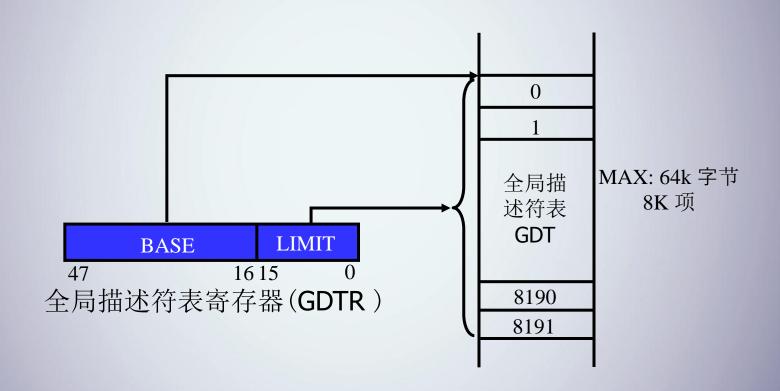
#### **GDTR**

- GDTR (Global Descriptor Table Register)
  - 其最低16位是限长,给出全局描述符表(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。
  - LGDT指令可以将描述符表的起始位置装入GDTR





#### **GDTR**



#### **GDT**

- 全局存储器是一种共享系统资源,该存储器可以被所有任务访问。
- 全局描述符表GDT是用来定义全局存储器空间的一种机制,它用段描述符说明一个全局存储器中的段,每个GDT最多含有8192个描述符(8192×8=64KB)。
- GDT是位于存贮器中,用全局描述符表寄存器GDTR 指出它的位置和大小,可软件编程改变。
- 全局描述符表只能有一个。



#### **IDTR**

- IDTR (Interrupt Descriptor Table Register)
  - 中断描述符表寄存器48位。
  - 其最低16位是限长,给出中断描述符表IDT的字节大小(其值比IDT的长度少1)
  - 其高32位是基址,指出IDT在物理存储器中存放的基地址

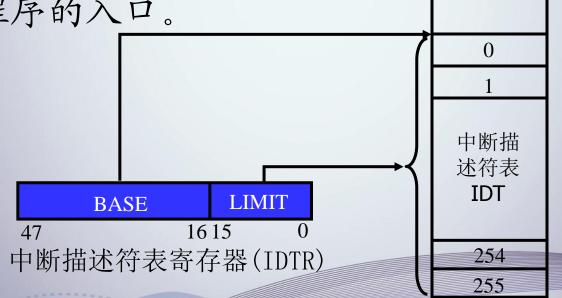




#### IDT

#### • 中断描述符表

- IDT中保存的是中断门描述符。每个门描述符包含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。中断门指出的是中断服务程序的入口。



每个门描述符8字节 最多256个门描述符

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

#### IDT

- 保护模式下的中断描述符表的功能,类似于实模式下的中断向量表
  - IDT的位置可变,由相应的描述符说明。
  - -实模式下的中断向量表的地址是固定的,必须 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。



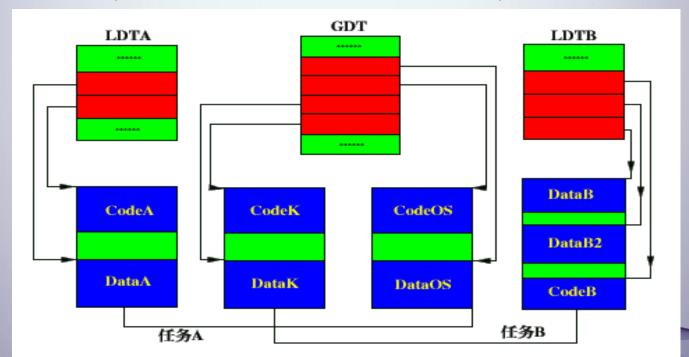
#### **LDTR**

- LDTR (Local Descriptor Table Register)
  - -局部描述符表寄存器/16位寄存器!
  - -保护模式下多任务的环境中每个任务都有一个 局部描述符表LDT。
  - LDT只含有与系统中某一个任务相关的各个段的描述符。
  - -和所有任务有关的公用段的描述符放在全局描述符表GDT中。
  - -每一任务的代码段、数据段、堆栈段与系统其 他部分隔离开,达到保护的目的。



### LDT

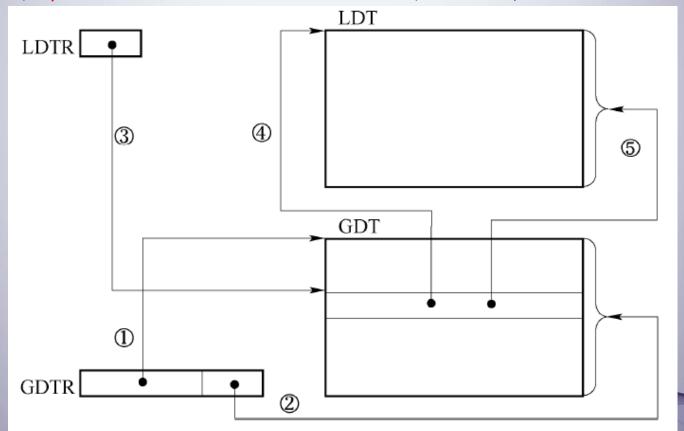
• 由于每项任务都有自己的LDT(且每项任务 只能有一个LDT),因此保护模式的软件系 统可能有多个LDT。但GDT只有一个。





### **LDTR**

• LDTR是一个16位的选择符,包含LDT描述符在GDT中的索引,不直接指出LDT的位置和大小。





- 段选择符 (selector)
  - 段寄存器的内容,是一个被称为段描述符的索引值,所检索到的表项叫做段描述符。

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
选择符		Index								TI	RPL					

- RPL (Requestor Privilege Level): 请求特权级, 2位二进制数字, 范围为0~3。
  - -00代表特权级0,01代表特权级1,10代表特权级2,11代表特权级3。请求特权级是将要访问的段的特权级。

- TI (Table Indicator): 表指示符。为0时, 从GDT中选择描述符; 为1时, 从LDT中选择描述符。
- 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=213个描述符。

- 段寄存器/选择符
- · 32位CPU有6个段寄存器:
- · CS——代码段寄存器, 其值为代码段的段值;
  - DS——数据段寄存器, 其值为数据段的段值;
  - ES——附加段寄存器, 其值为附加数据段的段值;
  - SS——堆栈段寄存器, 其值为堆栈段的段值;
  - FS——附加段寄存器,其值为附加数据段的段值;
  - GS——附加段寄存器,其值为附加数据段的段值。

- CS:EIP=0008:804F1266

SS:ESP=0010:8053CED8

-DS=0023 ES=0023 FS=0030 GS=0000

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

选择符 Index TI RPL

• 有四个系统控制寄存器CR0~CR3。CR0的低5 位是系统控制标志,被称为机器状态字MSW; 分页机制中用到CR3、CR2和CR0的其他位。

CRx	31	30 ∼12	11~5	4	3	2	1	0		
CR0	PG	000000000000000000	0000000	ET	TS	EM	MP	PE		
CR1	保留未用									
CR2	页故障线性地址									
CR3	页目录基	基址	000000000000							



- CR2: 页面故障线性地址寄存器,发生页异常时报告出错信息。
- CR3: 页目录基址寄存器PDBR (Page Directory Base Register) 高20位用于保存页目录表的起始物理地址的高20位。

CRx	31	30 ∼12	11~5	4	3	2	1	0			
CR0	PG	000000000000000000	0000000	ET	TS	EM	MP	PE			
CR1	保留未用										
CR2	页故障组	页故障线性地址									
CR3	页目录基址 0000000000000										



机器状态字MSW中各位的含义如下:

- ▶ PE (Protection mode Enable):保护模式允许标志。 = 0为实模式,CPU复位(启动)时自动进入实模式;= 1为保护模式。可以通过软件设置PE位,进入或退出保护模式。
- ➤ MP (Monitor coprocessor extension): 运算协处理器存在位。=1表示系统中有运算协处理器。
- ▶ ET (Extension Type): 协处理器选择标志。

	31	30~19	4	3	2	1	0
CR0	PG	0000	ET	TS	EM	MP	PE



- ►EM (Emulate processor extension): =1时系 统用软件模拟器执行数学运算。
- ▶TS (Task Switched): 任务切换标志。当任务 间切换时,自动设置此位为1。
- ▶PG (Paging enable): =1时,存储器管理单元允许分页。=0时,分页功能被关闭,线性地址等于物理地址。

# 保护模式下的调试和测试寄存器

- 8个32位调试寄存器和8个32位测试寄存器
  - DRO~DR3保存线性断点地址供分析; DR4、DR5为保留位; DR6为调试状态寄存器,检测异常; DR7为调试控制寄存器,用来规定断点字段的长度、断点访问类型、"允许"断点和"允许"所选择的调试条件。
  - 测试寄存器中TRO~TR5保留,用户可见的只有TR6、TR7。它们用于转换检测缓冲区(Translation Look-aside Buffer, TLB)中测试相关存储器。

# 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246B=64TB,



# 保护模式下存储器分段管理

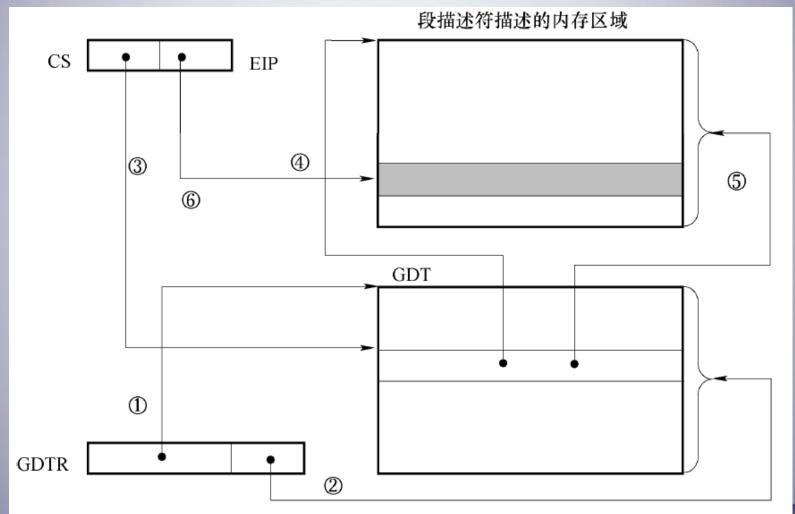


图 2-14 全局存储空间中寻址示例



## 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

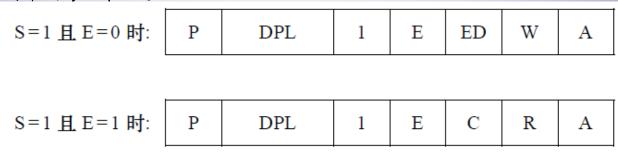
	7					0			
+0	限长 (位7~0)								
+1	限长 (位 15~8)								
+2	段基址 (位 7~0)								
+3	段基址 (位 15~8)								
+4	段基址 (位 23~16)								
+5	P DPL S TYPE A								
+6	G D 0 AVL 限长 (位 19~16)								
+7	段基址 (位 31~24)								

图 2-15 段描述符的格式



## 段描述符

- ▶ P (Present)存在位。=1时表示该段己装入内存; = 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。



#### 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表示段己被访问(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=1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

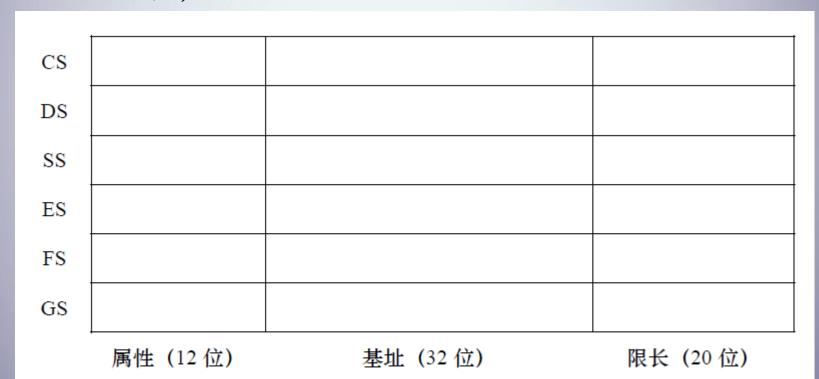
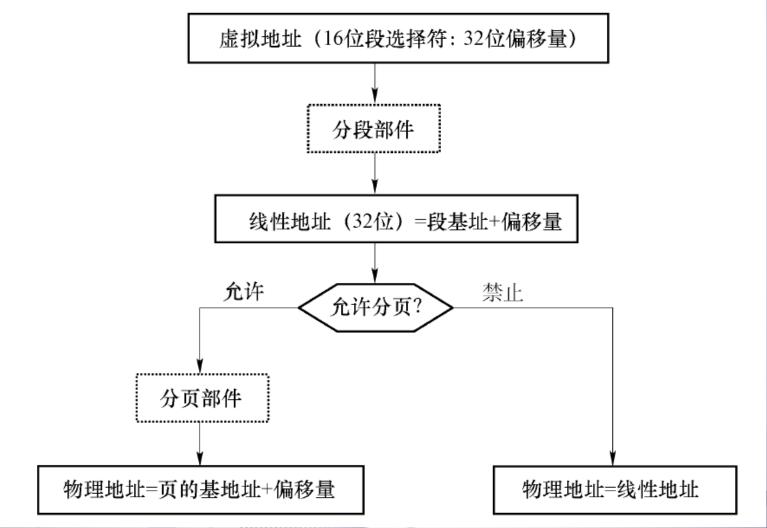


图 2-18 段描述符高速缓存



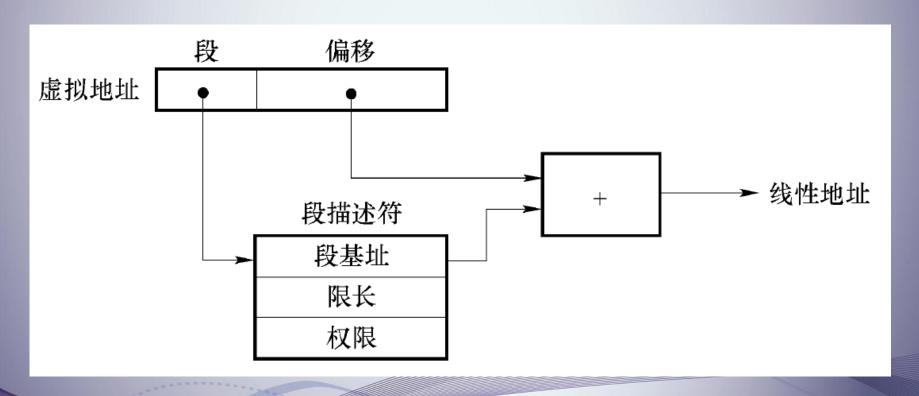
### 虚拟地址到物理地址





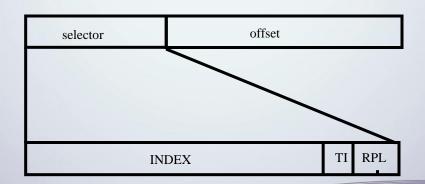
### 段式地址转换

• 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 $0000000_{16}$ -FFFFFFFF $_{16}$ 。



#### 物理地址

物理地址是微处理器引脚输出的地址信号。

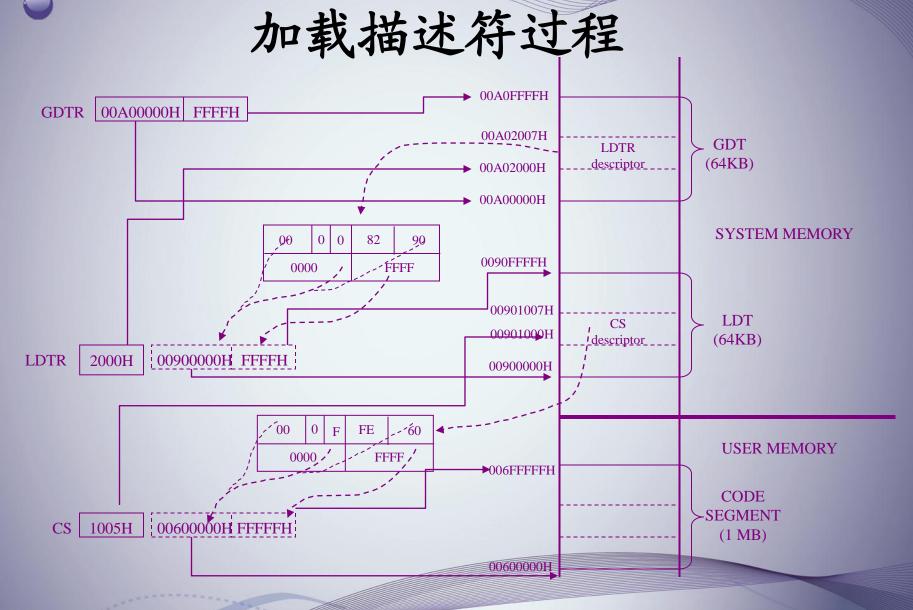
同一个线性地址得到的物理地址可以不同。

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。



CS=1003H, 其过程?

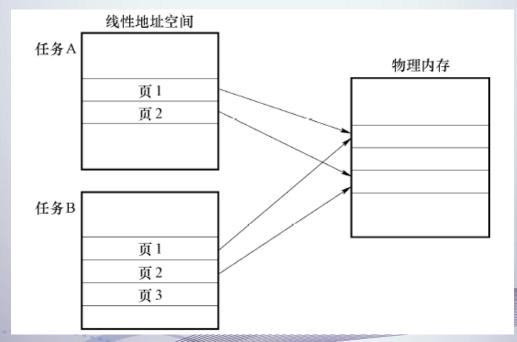


- 保护模式下的CPU支持分页机制,并且分页管理是在分段管理机制的基础上工作,它将分段管理机制得到的线性地址转换为物理地址。
- 使用分页机制的好处在于,它可以把每个活动任务当前所必需的少量页面放在内存中,而不必将整个段调入内存,从而提高了内存的使用效率。



- 分页
  - 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220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。

- 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。



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

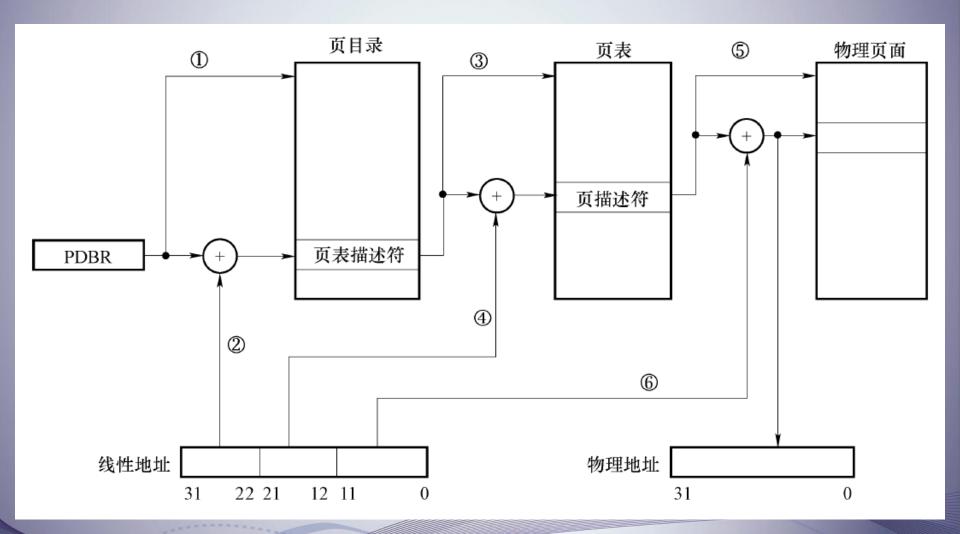
> 10位 10位 12位

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内的偏移。

页目录表

页表 页面/页帧





- 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。



- · 片内转换检测缓冲器TLB
  - 每次內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。

- 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。

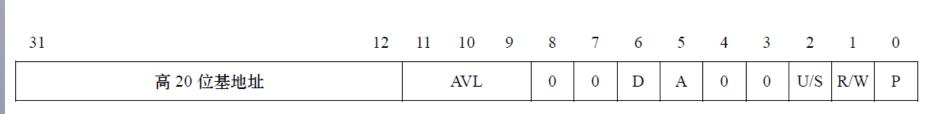


图 2-24 页表项的格式

- 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 系统页面	
U/S=0, R/W=0	不可读写	可读写		
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

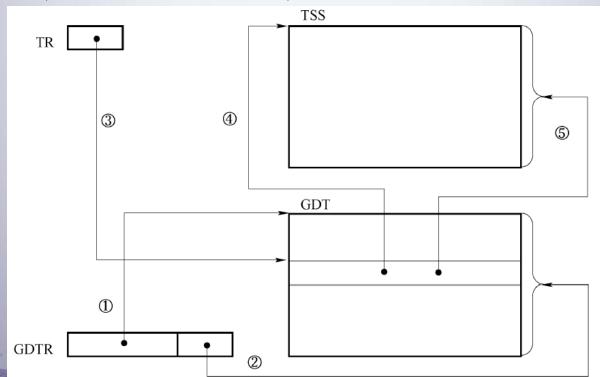
- · 在多任务环境下,每个任务都有属于自己的任务状态段TSS, TSS中包含启动任务所必需的信息。
- ·任务状态段TSS在存储器的基地址和限长 (大小)由TSS描述符指出。
- TSS描述符放在全局描述符表GDT中,TR内容 为选择符,它指出TSS描述符在GDT中的顺序 号。
- ·由描述符说明各TSS的位置和限长。



### 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。

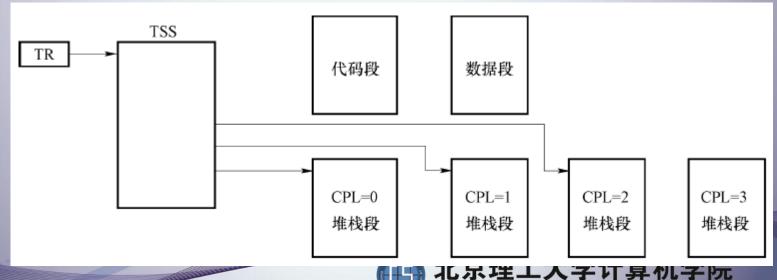
- ③步依据TR在GDT中取出TSS描述符。
- ④和⑤中,在TSS描述符中取得TSS的基址和限长。



算机学院

### 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。



School of Computer Science and Technology.BIT

# 任务状态段

• 104个字节

31	16	15	0 偏移					
	0000 0000 0000 0000	链接字段	+00					
	E	ESP0						
	0000 0000 0000 0000	SS0	+08					
	E	ESP1						
	0000 0000 0000 0000	SS1	+10					
	E	ESP2						
	0000 0000 0000 0000	SS2	+18					
		CR3						
		EIP						
		EFLAGS						
		EAX						
ECX EDX								
						_		EBX
_	ESP							
-	EBP							
-		ESI						
-	EDI							
-	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	Ź			
+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

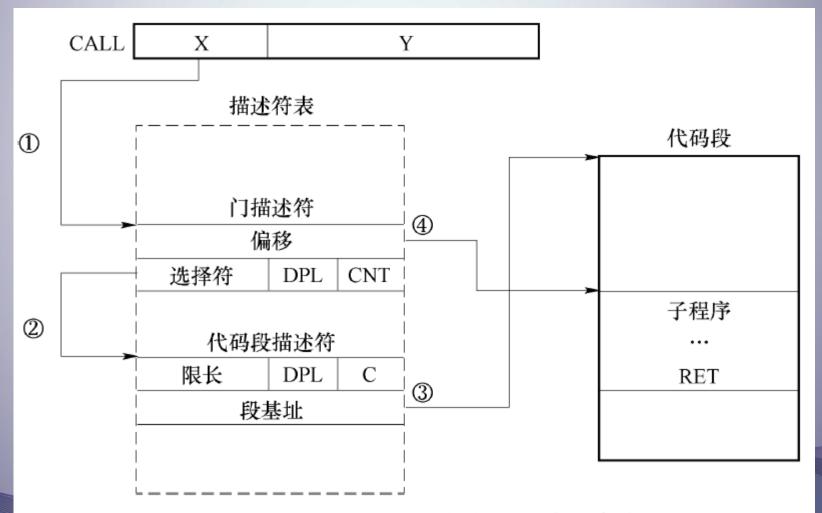


图 2-33 通过调用门转移到更高特权级

### 任务门

• 选择符指示TSS



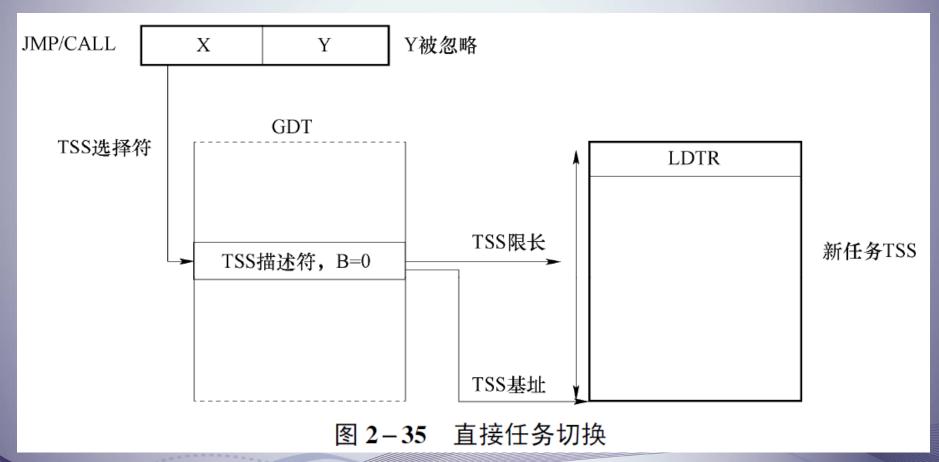
图 2-34 任务门描述符的格式

### 任务切换

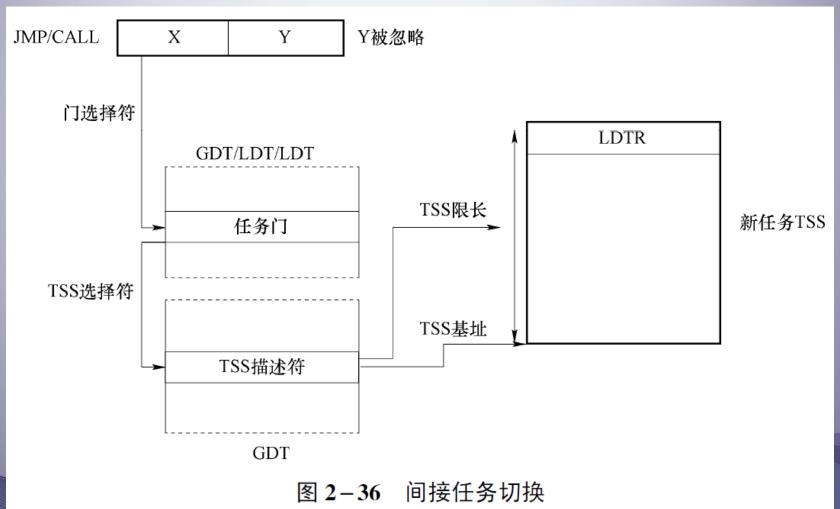
- 四种情况
  - -执行远程JMP 或者CALL 指令,选择了GDT 中的TSS 描述符。
  - 执行远程JMP 或者CALL 指令,从GDT 或者LDT 中选择了任务门。
  - 发生了中断或异常,中断向量选择了IDT 中的任务门。
  - 当FLAGS 中的NT=1 时,执行IRET 指令,目的任务选择符在执行IRET 任务的TSS链接域中。



#### 直接任务切换



#### 间接任务切换



school of configure science and recimology, or

#### 直接任务切换

- DPL<sub>TSS</sub>≥MAX (CPL, RPL)【值比较】
- 间接任务切换
- · 当CPL>DPL<sub>TSS</sub>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。



- 把CRO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

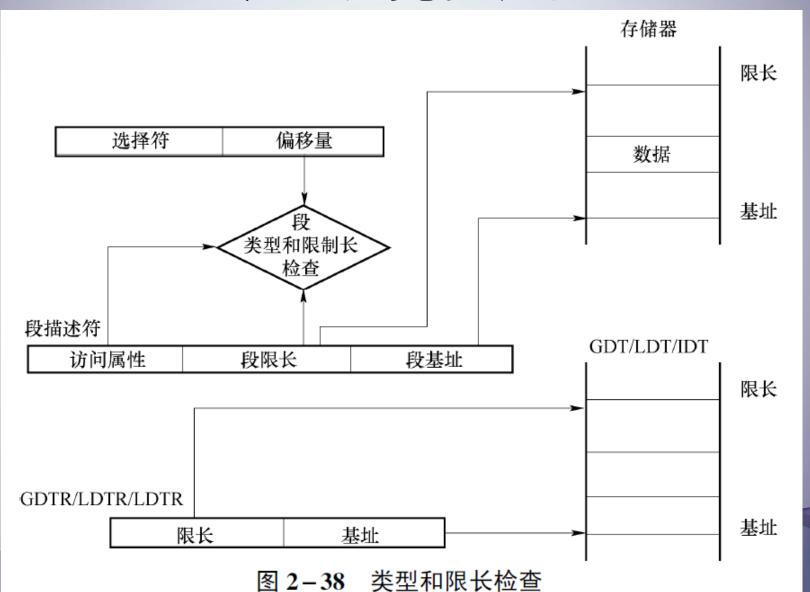
保护模式下设置了0、1、2、3共4个特权级,以区分操作系统程序和应用程序。这种特权级机制阻止了用户程序对操作系统的非法访问,保证高特权级的代码或数据不被低特权级的程序所破坏。

## 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



## 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

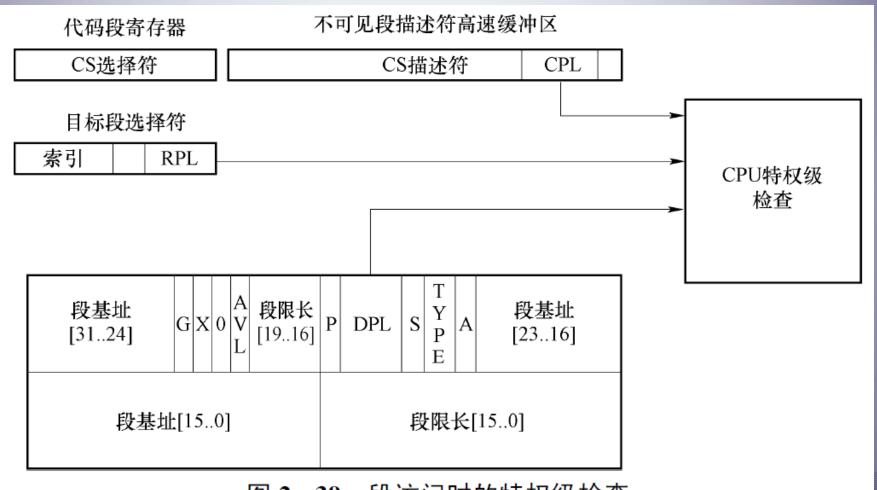


图 2-39 段访问时的特权级检查



## 数据段访问示例

• DPL ≥ MAX (CPL, RPL)

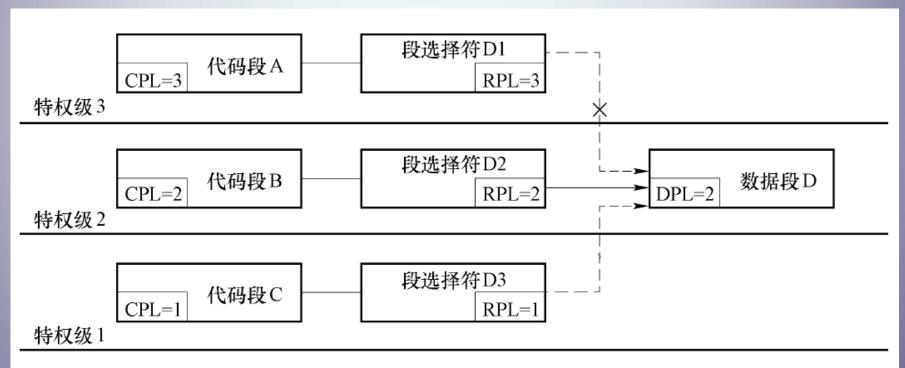


图 2-40 特权级保护数据段访问的一个例子

# 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

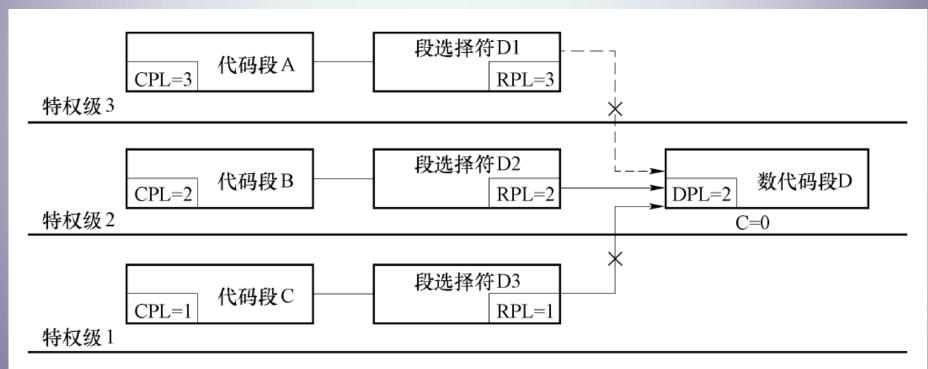


图 2-41 C=0 的代码段



## 对转移程序的保护

· C=1, 一致代码段

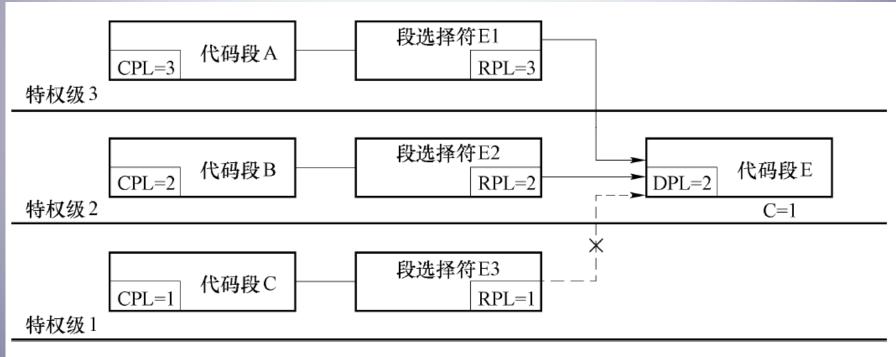


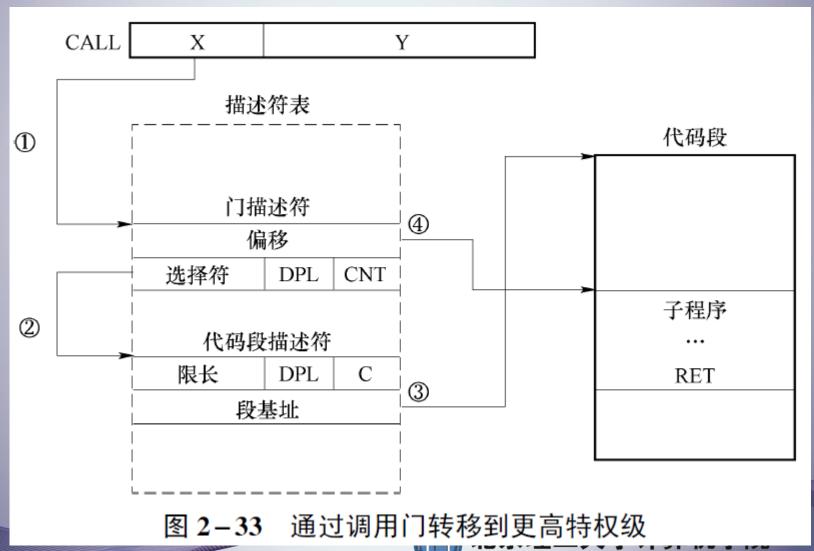
图 2-42 C=1 的代码段

## 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;
  - DPLGATE, 即门描述符的DPL;
  - DPLcode, 即目标代码段描述符的DPL;
  - CCODE, 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



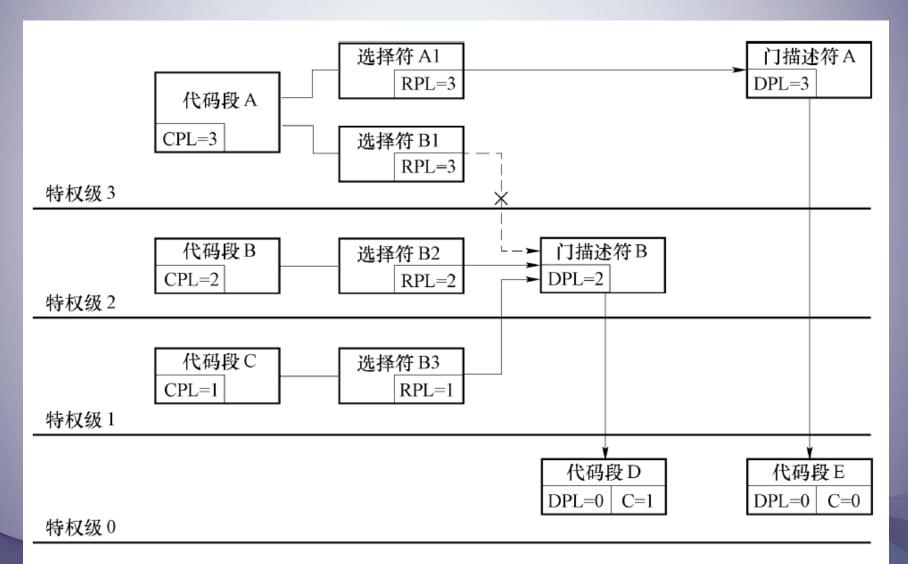
#### CALL指令

- 满足两个条件
  - DPLGATE ≥ MAX (CPL, RPL)
  - DPLcode ≤ 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:
    - DPLcode (DPLcode < CPL) 或者维持不变 (DPLcode=CPL)
    - 防止执行调用门后特权级降低

#### JMP指令

- 满足两个条件
  - DPLGATE ≥ MAX (CPL, RPL)
  - DPLcode ≤ CPL (C=1) 或者 DPLcode = CPL (C=0)

## 调用门特权级检查



## 输入输出保护

- 10端口对系统运行很重要
- CPU采用1/0特权级10PL和TSS段中1/0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 10PL。CPL≤10PL时,可以执行1/0敏感指令。

## 10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



## 10许可位图

- 10PL限制导致应用程序要么访问所有10地址,要不都不能访问,不够灵活。
- 0-不限制, 1-10PL级别或更高
- IN EAX, DX涉及4个地址

	7	6	5	4	3	2	1	0	
I/O位图基址 ->	$E_7$	$E_6$	$E_5$	$E_4$	E <sub>3</sub>	$E_2$	$E_1$	E <sub>0</sub>	
	E <sub>15</sub>	E <sub>14</sub>	E <sub>13</sub>	E <sub>12</sub>	E <sub>11</sub>	E <sub>10</sub>	E <sub>9</sub>	E <sub>8</sub>	
		 		   		   	   		〉s 字节
TSS段限长	$E_{s*8-1}$	$E_{s*8-2}$	$E_{s*8-3}$	$E_{s*8-4}$	E <sub>s*8-5</sub>	E <sub>s*8-6</sub>	$E_{s*8-7}$	E <sub>s*8-8</sub>	$\int$

• 作业

- 习题1

1.9

- 习题2

2. 15, 2. 21, 2. 23, 2. 24