汇编语言程序设计

吴 向 军

中山大学计算机科学

2003.03.20

中山大學

第5章 微机CPU的指令系统

5.1 汇编语言指令格式

5.1.1 指令格式

汇编语言的指令格式如下:

指令助忆符 [操作数1 [, 操作数2 [, 操作数3]]](*) [;注释]

指令助忆符体现该指令的功能, 它对应一条二进制编码的机器指令。一条指令可以没有操作数, 也可以有一个、二个或三个操作数。

绝大多数指令的操作数要显式的写出来, 但也有指令的操作数是隐含的。

当指令含有操作数, 并要求在指令中显式地写出来时, 则在书写时必须遵守:

- 指令助忆符和操作数之间要有分隔符, 分隔符可以是若干个空格或TAB键
- 如果指令含有多个操作数, 那么, 操作数之间要用逗号“, ”分开
- 指令后面还可以书写注释内容, 不过, 要在注释之前书写分号“;”

(*) 操作数1为第一操作数, 操作数2(如有的话)为第二操作数, 其后以此类推。

第5章 微机CPU的指令系统

5.1.2 了解指令的几个方面

在学习汇编指令时，指令的功能无疑是我们学习和掌握的重点，但要准确、有效地运用这些指令，我们还要熟悉系统对每条指令的一些规定或约束。

归纳起来，对指令还要掌握以下几个方面内容：

- 要求指令操作数的寻址方式
- 指令对标志位的影响、标志位对指令的影响
- 指令的执行时间，对可完成相同功能的指令要选用执行时间短的指令

第5章 微机CPU的指令系统

5.2 指令系统

指令系统是CPU指令的集合，CPU除了具有计算功能的指令外，还有一些实现其它功能的指令，也有为某种特殊的应用而增设的指令。

指令按其功能分成以下几大类：数据传送指令、标志位操作指令、算术运算指令、逻辑运算指令、移位操作指令、位操作指令、比较运算指令、循环指令、转移指令、条件设置字节指令、字符串操作指令、ASCII-BCD码运算调整指令和处理器指令等。

5.2.1 数据传送指令

数据传送指令分：传送指令、交换指令、地址传送指令、堆栈操作指令、查表指令和I/O指令等。

除指令SAHF和POPF指令外，本类的其它指令都不影响标志位。

第5章 微机CPU的指令系统

中山大學

1). 传送指令MOV(Move Instruction)

传送指令是使用最频繁的指令,它相当于高级语言中的赋值语句。格式如下:

MOV Reg/Mem, Reg/Mem/Imm(*)

指令的功能是把源操作数(第二操作数)的值传给目的操作数(第一操作数)。

指令执行后,目的操作数的值被改变,而源操作数的值不变。当存储单元是该指令的一个操作数时,该操作数的寻址方式可以是任意一种存储单元寻址方式。

下面列举几组正确的指令例子:

- 源操作数是寄存器
- 源操作数是存储单元
- 源操作数是立即数

(*) Reg—Register(寄存器), Mem—Memory(存储器), Imm—Immediate(立即数), 它们可以是8位、16位或32位(特别指出其位数的除外)。

中山大學

第5章 微机CPU的指令系统

```
mov al,4                    ; al←4，字节传送
mov cx,0ffh                 ; cx←00ffh，字传送
mov si,200h                 ; si←0200h，字传送
mov byte ptr [si],0ah       ; byte ptr 说明是字节操作
mov word ptr [si+2],0bh     ; word ptr 说明是字操作
```

- 注意立即数是字节量还是字量
- 明确指令是字节操作还是字操作

```
mov ax,bx                   ; ax←bx，字传送
mov ah,al                   ; ah←al，字节传送
mov ds,ax                   ; ds←ax，字传送
mov [bx],al                 ; [bx]←al，字节传送
```

第5章 微机CPU的指令系统


```
mov al,[bx]
mov dx,[bp]          ; dx←ss:[bp]
mov es,[si]          ; es←ds:[si]
```

- 不存在存储器向存储器的传送指令

第5章 微机CPU的指令系统

对MOV指令有以下几条具体规定，其中有些规定对其它指令也同样有效。

① 两个操作数的数据类型要相同，要同为8位、16位或32位；

如：MOV BL，AX等是不正确的；

② 两个操作数不能同时为段寄存器，如：MOV ES，DS等；

③ 代码段寄存器CS不能为目的操作数，但可作为源操作数；

如：指令MOV CS，AX等不正确，但指令MOV AX，CS等是正确的；

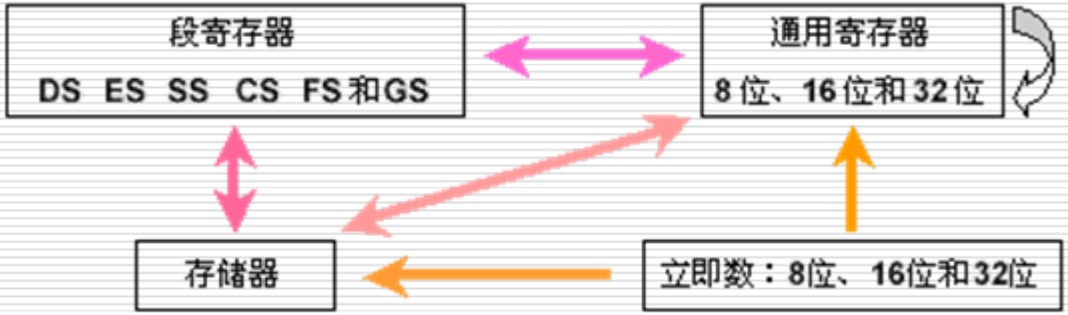
④ 立即数不能直接传给段寄存器，如：MOV DS，100H等；

⑤ 立即数不能作为目的操作数，如：MOV 100H，AX等；

⑥ 指令指针IP，不能作为MOV指令的操作数；

⑦ 两个操作数不能同时为存储单元；

如：MOV VARA，VARB等，其中VARA和VARB是同数据类型的内存变量。



第5章 微机CPU的指令系统

对于规定2、4和7，我们可以用通用寄存器作为中转来达到最终目的。

功 能 描 述	不正确的指令	可选的解决方法
把DS的值传送给ES	MOV ES, DS	MOV AX, DS MOV ES, AX
把100H传给DS	MOV DS, 100H	MOV AX, 100H MOV DS, AX
把字变量VARB的值传送给字变量VARA	MOV VARA, VARB	MOV AX, VARB MOV VARA, AX

第5章 微机CPU的指令系统

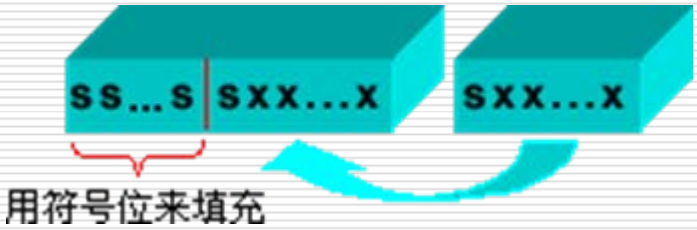
中山大學

2). 传送—填充指令(Move-and-Fill Instruction)

传送—填充指令是把位数短的数据传送给位数长的目的操作数。

MOVSX/MOVZX Reg/Mem, Reg/Mem/Imm ;80386+

指令的主要功能和限制与MOV指令类似，不同之处是：在传送时，对目的操作数的高位进行填充。根据其填充方式，又分为：符号填充和零填充。



例5.1 已知：AL=87H，指令MOVSX CX, AL, MOVZX DX, AL执行后，问CX和DX的值是什么？

第5章 微机CPU的指令系统

3). 交换指令XCHG(Exchange Instruction)

交换指令XCHG是两个寄存器, 寄存器和内存变量之间内容的交换令, 两个操作数的数据类型要相同。

XCHG Reg/Mem, Reg/Mem

该指令的功能和MOV指令不同, 后者是一个操作数的内容被修改, 而前者是两个操作数都会发生改变。

寄存器不能是段寄存器, 两个操作数也不能同时为内存变量。

例5.2 已知: AX=5678H, BX=1234H, 指令XCHG AX, BX执行后, 求各寄存器的值。

第5章 微机CPU的指令系统


```
mov  ax,1234h                ;  ax=1234h
mov  bx,5678h                ;  bx=5678h
xchg  ax,bx
;  ax=5678h，bx=1234h
xchg  ah,al                  ;  ax=7856h
```

```
xchg  ax,[2000h]        ;  字交换 等同于 xchg [2000h],ax
xchg  al,[2000h]        ;  字节交换等同于xchg [2000h],al
```

第5章 微机CPU的指令系统

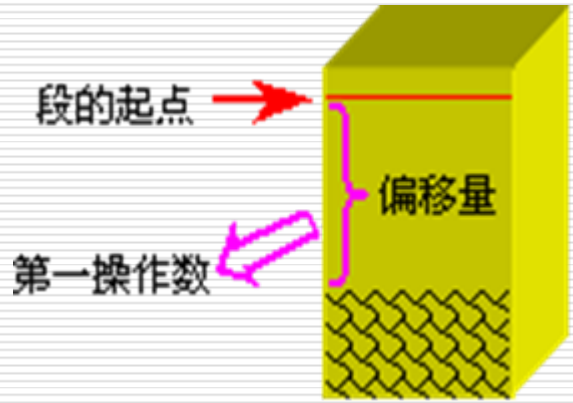
中山大學

4). 取有效地址指令LEA(Load Effective Address)

指令LEA是把一个内存变量的有效地址送给指定的寄存器。其指令格式如下:

LEA Reg, Mem

该指令通常用来对指针或变址寄存器BX、DI或SI等置初值之用。




```
…
BUFFER  DB  100 DUP(?)
…
LEA     SI, BUFFER          ;把字节变量BUFFER在数据段内的偏移量送给SI
LEA     BX, [BX+DI+200]     ;把有效地址BX+DI+200送给BX
…
```

问题: 指令“LEA BX, BUFFER”和“MOV BX, *OFFSET* BUFFER”的执行效果是一样的吗? 指令“LEA BX, [BX+200]”和“MOV BX, *OFFSET* [BX+200]”二者都正确吗?

第5章 微机CPU的指令系统

例：获取有效地址

```
mov bx,0400h
mov si,3ch
lea bx,[bx+si+0f62h];   BX = 0400h + 003ch + 0f62h = 139EH
```

- 获得主存单元的有效地址；不是物理地址，也不是该单元的内容
- 可以实现计算功能

第5章 微机CPU的指令系统

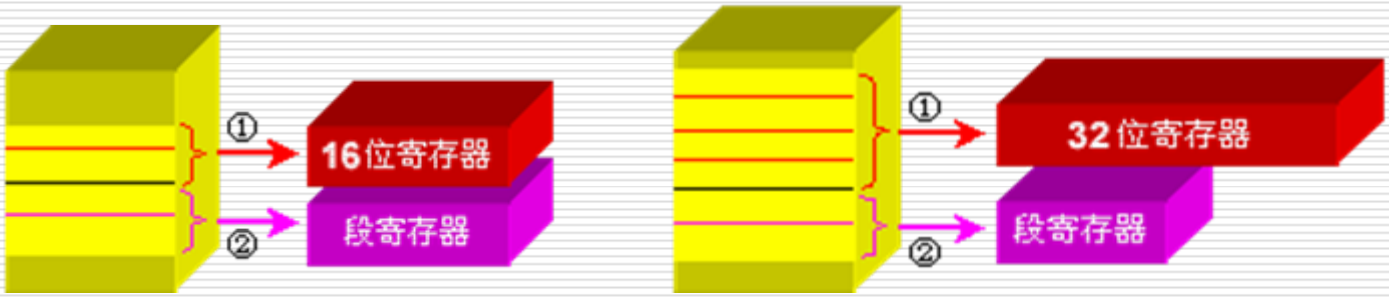
5). 取段寄存器指令(Load Segment Instruction)

该组指令的功能是把内存单元的一个“低字”传送给指令中指定的16位寄存器，把随后的一个“高字”传给相应的段寄存器(DS、ES、FS、GS和SS)。

LDS/LES/*LFS/LGS/LSS* Reg, Mem

指令LDS(Load Data Segment Register)和LES(Load Extra Segment Register)在8086CPU中就存在，而LFS和LGS(Load Extra Segment Register)、LSS(Load Stack Segment Register)是80386及其以后CPU中才有的指令。

- 若Reg是16位寄存器，那么，Mem必须是32位指针；
- 若Reg是32位寄存器，那么，Mem必须是48位指针，其低32位给指令中指定的寄存器，高16位给指令符中的段寄存器。



第5章 微机CPU的指令系统

例如:

```
…
PT1  DD  12345678H
PT2  DF  43219012ABCDH
…
LES  BX,  PT1
LDS  ESI,  PT2
…
```

执行上述二条指令后,各寄存器的内容分别为: (BX) = 5678H, (ES) = 1234H, (ESI) = 9012ABCDH, (DS) = 4321H。

第5章 微机CPU的指令系统

例：地址指针传送

```
mov word ptr [3060h],0100h
mov word ptr [3062h],1450h
les di,[3060h]          ；es=1450h，di=0100h
lds si,[3060h]          ；ds=1450h，si=0100h
```

- mem指定主存的连续4个字节作为逻辑地址（32位的地址指针），送入DS:r16或ES:r16

第5章 微机CPU的指令系统

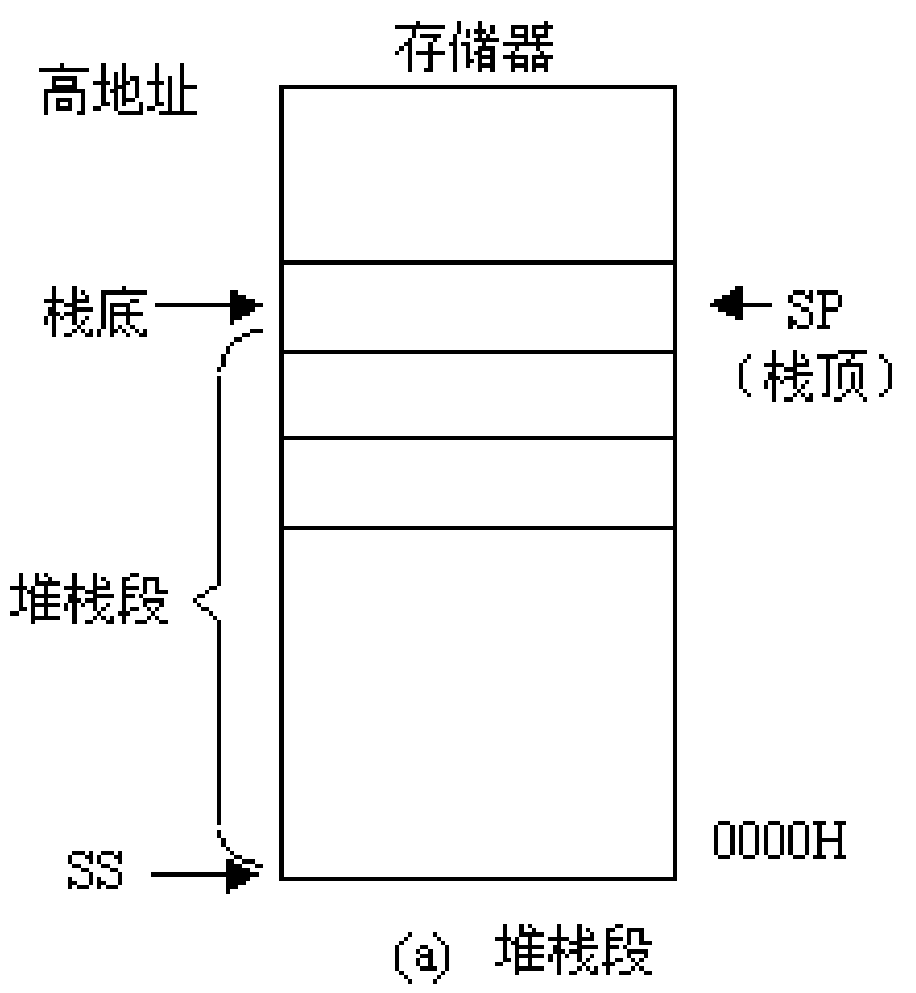
中山大學

6)、进栈操

堆栈是一个“后进先出FILO”（或说“先进后出FILO”）的主存区域，位于堆栈段中；SS段寄存器记录其段地址

堆栈只有一个出口，即当前栈顶；用堆栈指针寄存器SP指定

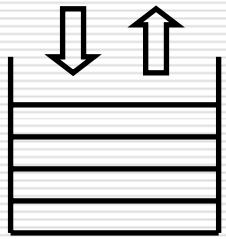
栈顶是地址较小的一端（低端），栈底不变



第5章 微机CPU的指令系统

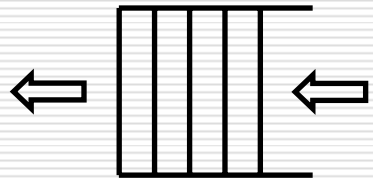
堆栈和队列

堆栈：按照后进先出(LIFO)的原则组织的存储器空间（栈）



LIFO

队列：按照先进先出(FIFO)的原则组织的存储器空间



FIFO

第5章 微机CPU的指令系统

- **PUSH (Push Word or Doubleword onto Stack)**

  指令格式: PUSH Reg/Mem

  PUSH Imm ;80286+

  一个字进栈, 系统自动完成两步操作: SP←SP-2, [SP]←操作数;

  一个双字进栈, 系统自动完成两步操作: ESP←ESP-4, [ESP]←操作数。

- **PUSHA (Push All General Registers)**

  指令格式: PUSHA ;80286+

  其功能是依次把寄存器AX、CX、DX、BX、SP、BP、SI和DI等压栈。

- **PUSHAD (Push All 32-bit General Registers)**

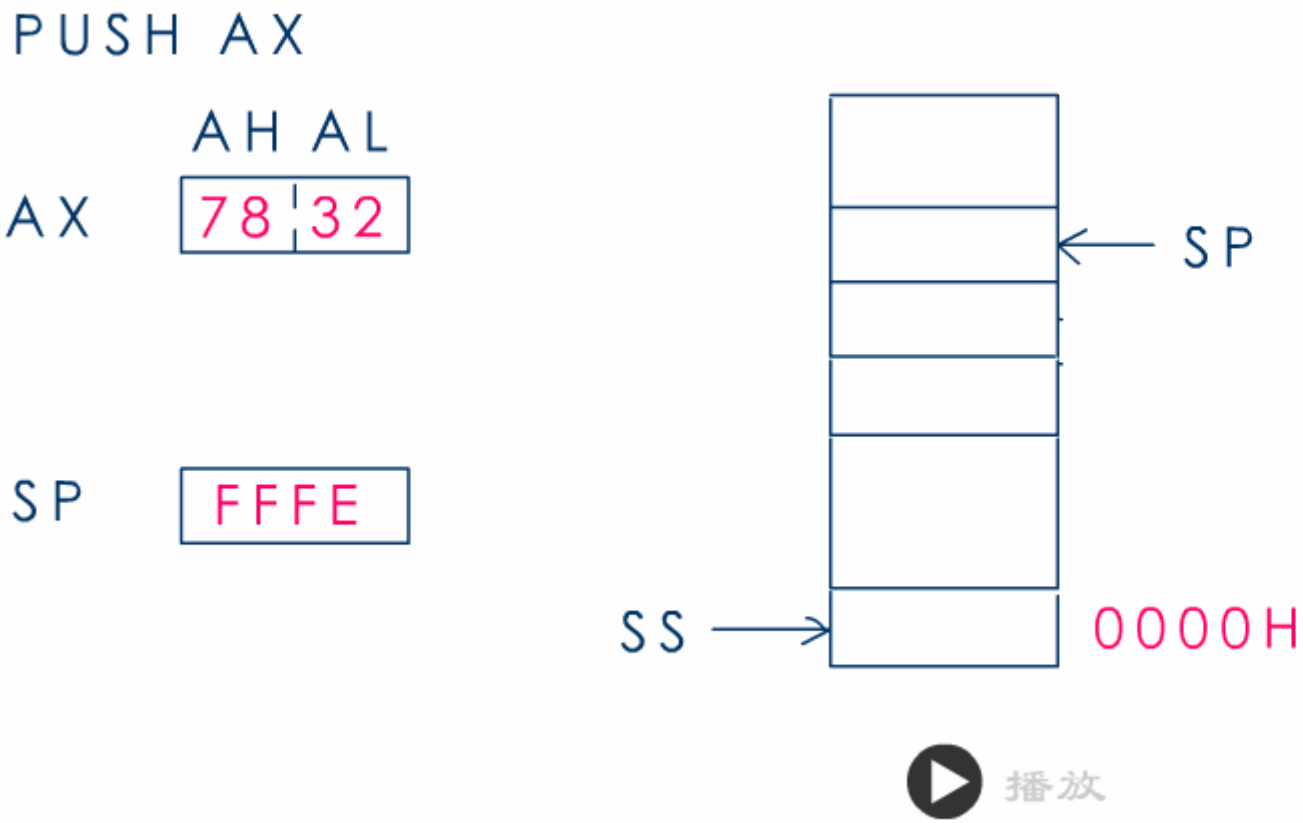
  指令格式: PUSHAD ;80386+

  其功能是把寄存器EAX、ECX、EDX、EBX、ESP、EBP、ESI和EDI等压栈。

第5章 微机CPU的指令系统

```
PUSH r16/m16/seg
; SP←SP－2
; SS:[SP]←r16/m16/seg
```

```
push ax
push [2000h]
```



第5章 微机CPU的指令系统

7)、出栈操作

- **POP (Pop Word or Doubleword off Stack)**

  指令格式: POP Reg/Mem

  弹出一个字, 系统自动完成两步操作: 操作数←[SP], SP←SP+2;

  弹出一个双字, 系统自动完成两步操作: 操作数←[ESP], ESP←ESP+4。

- **POPA (Pop All General Registers)**

  指令格式: POPA ;80286+

  其功能是依次把寄存器DI、SI、BP、SP、BX、DX、CX和AX等弹出栈。实, 程序员不用记住它们的具体顺序, 只要与指令PUSHA对称使用就可以了。

- **POPAD (Pop All 32-bit General Registers)**

  指令格式: POPAD ;80386+

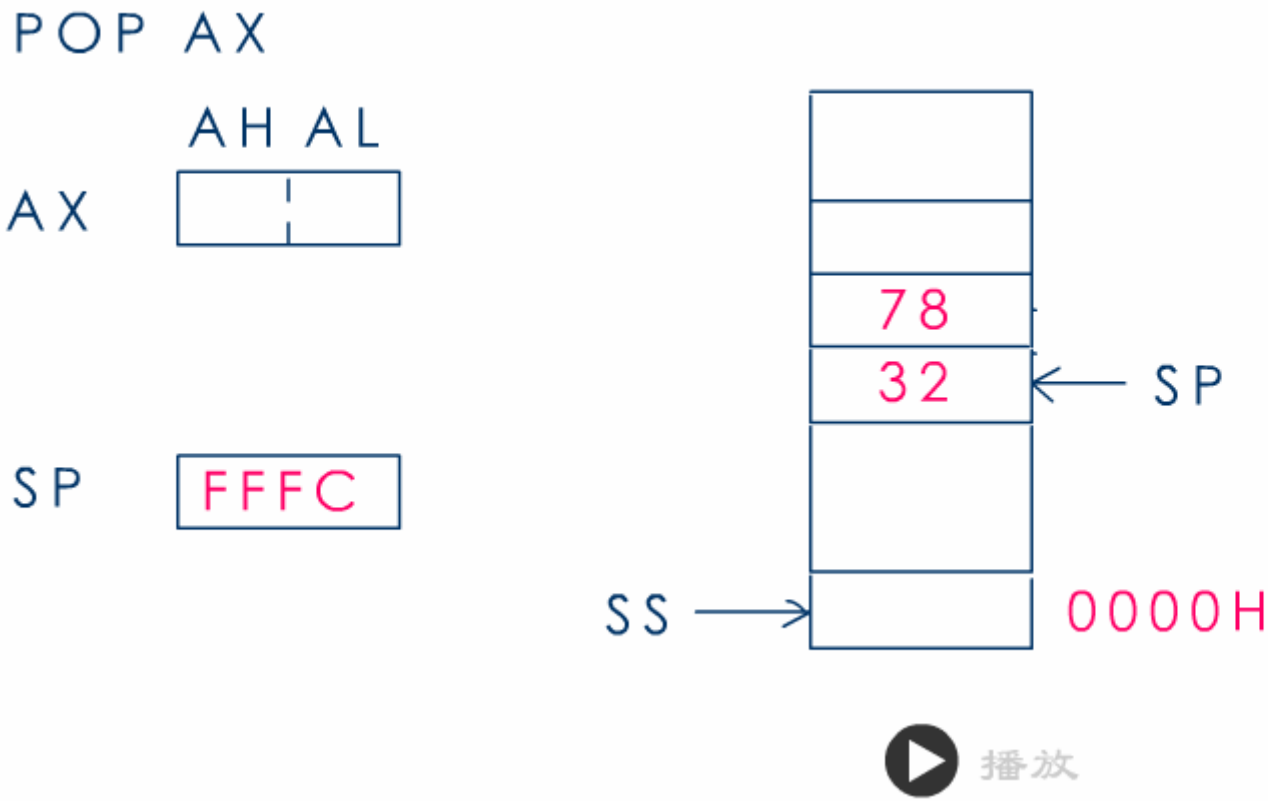
  其功能是依次把寄存器EDI、ESI、EBP、ESP、EBX、EDX、ECX和EAX等弹出栈, 它与PUSHAD对称使用即可。

第5章 微机CPU的指令系统


```
POP r16/m16/seg
; r16/m16/seg←SS:[SP]
; SP←SP+2
```

pop ax

pop [2000h]



第5章 微机CPU的指令系统

堆栈的特点:

堆栈操作的单位是字，进栈和出栈只对字量

字量数据从栈顶压入和弹出时，都是低地址字节送低字节，高地址字节送高字节

堆栈操作遵循先进后出原则，但可用存储器寻址方式随机存取堆栈中的数据

堆栈常用来

- 临时存放数据
- 传递参数
- 保存和恢复寄存器

第5章 微机CPU的指令系统

例：现场保护恢复

```
push ax                    ; 进入子程序后
push bx
push ds
...
pop ds                     ; 返回主程序前
pop bx
pop ax
```

第5章 微机CPU的指令系统

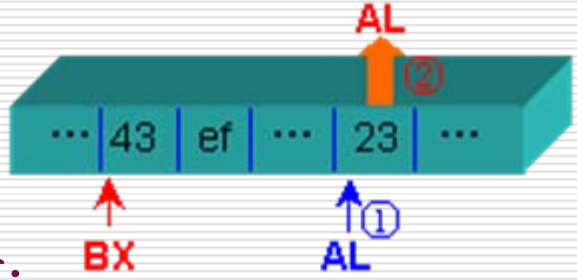
中山大學

8)、转换指令XLAT(Translate Instruction)

转换指令有两个隐含操作数BX和AL。指令格式如下:

XLAT/XLATB

其功能是把BX的值作为内存字节数组首地址、下标为AL的数组元素的值传送给AL。其功能描述的表达式是: AL←BX[AL]。



换码指令执行前:

在主存建立一个字节量表格，内含要转换成的目的代码

表格首地址存放于BX，AL存放相对表格首地址的位移量

换码指令执行后:

将AL寄存器的内容转换为目标代码

第5章 微机CPU的指令系统

例：代码转换

```
mov bx,100h
mov al,03h
xlat
```

- 换码指令没有显式的操作数，但使用了BX和AL；因为换码指令使用了隐含寻址方式——采用默认操作数

第5章 微机CPU的指令系统

5.2.2 标志位操作指令

1)、进位CF操作指令

- 清进位指令CLC(Clear Carry Flag): CF ← 0
- 置进位指令STC(Set Carry Flag): CF ← 1
- 进位取反指令CMC(Complement Carry Flag): CF ← not CF

2)、方向位DF操作指令

- 清方向位指令CLD(Clear Direction Flag): DF ← 0
- 置方向位指令STD(Set Direction Flag): DF ← 1

3)、中断允许位IF操作指令

- 清中断允许位指令CLI(Clear Interrupt Flag): IF ← 0

  其功能是不允许可屏蔽的外部中断来中断其后程序段的执行。

- 置中断允许位指令STI(Set Interrupt Flag): IF ← 1

  其功能是恢复可屏蔽的外部中断的中断响应功能,通常是与CLI成对使用的。

第5章 微机CPU的指令系统

4)、取标志位操作指令

- LAHF (Load AH from Flags): AH←Flags的低8位
- SAHF (Store AH in Flags): Flags的低8位←AH

5)、标志位堆栈操作指令

- PUSHF/PUSHFD (Push Flags onto Stack)

  把16位/32位标志寄存器进栈;

- POPF/POPFD (Pop Flags off Stack)

  把16位/32位标志寄存器出栈;

第5章 微机CPU的指令系统

LAHF

；AH←FLAGS的低字节

LAHF指令将标志寄存器的低字节送寄存器AH

SF/ZF/AF/PF/CF状态标志位分别送入AH的第7/6/4/2/0位，而AH的第5/3/1位任意

SAHF

；FLAGS的低字节←AH

SAHF将AH寄存器内容送FLAGS的低字节

用AH的第7/6/4/2/0位相应设置SF/ZF/AF/ PF/CF标志

第5章 微机CPU的指令系统

PUSHF

; SP←SP－2

; SS:[SP]←FLAGS

PUSHF指令将标志寄存器的内容压入堆栈，同时栈顶指针SP减2

POPF

; FLAGS←SS:[SP]

; SP←SP＋2

POPF指令将栈顶字单元内容送标志寄存器，同时栈顶指针SP加2

第5章 微机CPU的指令系统

例：置位单步标志

```
pushf           ; 保存全部标志到堆栈
pop ax          ; 从堆栈中取出全部标志
or ax,0100h     ; 设置D8=TF=1，
                ; ax其他位不变
push ax         ; 将ax压入堆栈
popf            ; FLAGS←AX
; 将堆栈内容取到标志寄存器
```

第5章 微机CPU的指令系统

5.2.3 算术运算指令

1. 加法指令

- **加法指令ADD(ADD Binary Numbers Instruction)**

  指令格式: **ADD Reg/Mem, Reg/Mem/Imm**

  指令的功能是把源操作数的值加到目的操作数中。

- **带进位加指令ADC(ADD With Carry Instruction)**

  指令格式: **ADC Reg/Mem, Reg/Mem/Imm**

  指令的功能是把源操作数和进位标志位**CF**的值**(0/1)**一起加到目的操作数中。

- **加1指令INC (Increment by 1 Instruction)**

  指令格式: **INC Reg/Mem**

  指令的功能是把操作数的值加**1(**不影响**CF)**。

- **交换加指令XADD (Exchange and Add)**

  指令格式: **XADD Reg/Mem, Reg ;80486+**

  指令的功能是先交换两个操作数的值, 再进行算术“加”法操作。

第5章 微机CPU的指令系统

例：加法运算

```
mov al,0fbh                    ; al=0fbh
add al,07h                     ; al=02h
mov word ptr [200h],4652h      ; [200h]=4652h
mov bx,1feh                    ; bx=1feh
add al,bl                      ; al=00h
add word ptr [bx+2],0f0f0h     ; [200h]=3742h
```

第5章 微机CPU的指令系统

例：双字加法

```
mov  ax,4652h          ;  ax=4652h
add  ax,0f0f0h         ;  ax=3742h，CF=1
mov  dx,0234h          ;  dx=0234h
adc  dx,0f0f0h         ;  dx=f325h，CF=0
;  DX.AX = 0234  4652H
              + F0F0  F0F0H
              = F325  3742H
```

中山大學

增量指令INC（increment）

INC指令对操作数加1（增量）

INC指令不影响进位CF标志，按定义设置其他状态标志

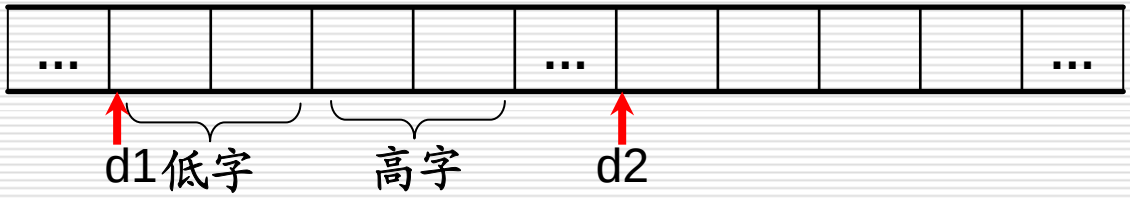
```
INC reg/mem
; reg/mem←reg/mem+1
```

```
inc bx
inc byte ptr [bx]
```

第5章 微机CPU的指令系统

例5.3 已知有二个32位数d1和d2(用类型DD说明), 编写程序片段把d2的值加到d1中。

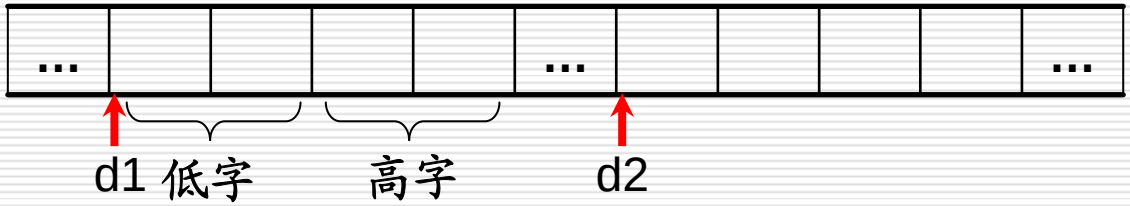
解: 32位数d1和d2在内存中如下所示。



方法1: 用16位寄存器编写程序

```
MOV  AX,  word ptr d1      ;由于d1是双字类型, 必须使用强制类型说明符。
MOV  DX,  word ptr d1+2    ;(DX, AX)构成一个32位数据
ADD  AX,  word ptr d2      ;低字相加
ADC  DX,  word ptr d2+2    ;高字相加。在低字相加时, 有可能会产生“进位”
MOV  word ptr d1,  AX      ;低字送给d1的低字
MOV  word ptr d1+2,  DX    ;高字送给d1的高字
```

第5章 微机CPU的指令系统



方法2: 用32位寄存器编写程序

```
MOV  EAX,  d1
ADD  EAX,  d2
MOV  d1,  EAX
```

从上面两段程序不难看出: 用32位寄存器来处理32位数据显得简单、明了, 而16位微机虽然也能处理32位数据, 但做起来就要复杂一些。

第5章 微机CPU的指令系统

2. 减法指令

◆减法指令SUB (Subtract Binary Values Instruction)

指令格式: SUB Reg/Mem, Reg/Mem/Imm

指令的功能是从目的操作数中减去源操作数。

◆带借位减SBB (Subtract with Borrow Instruction)

指令格式: SBB Reg/Mem, Reg/Mem/Imm

指令的功能是把源操作数和标志位CF的值从目的操作数中一起减去。

◆减1指令DEC (Decrement by 1 Instruction)

指令格式: DEC Reg/Mem

指令的功能是把操作数的值减去1(不影响CF)。

◆求补指令NEG (Negate Instruction)

指令格式: NEG Reg/Mem

指令的功能: 操作数 = 0 - 操作数, 即改变操作数的正负号。

第5章 微机CPU的指令系统

例：减法运算

```
mov  al,0fbh          ;  al=0fbh
sub  al,07h           ;  al=0f4h，CF = 0
```

第5章 微机CPU的指令系统

例：双字减法

```
mov  ax,4652h          ;  ax=4652h
sub  ax,0f0f0h         ;  ax=5562h，CF=1
mov  dx,0234h          ;  dx=0234h
sbb  dx,0f0f0h         ;  dx=1143h，CF=1
;  DX.AX = 0234  4652H
          - F0F0  F0F0H
          = 1143  5562H
```

第5章 微机CPU的指令系统

减量指令DEC（decrement）

DEC指令对操作数减1（减量）

DEC指令不影响进位CF标志，按定义设置其他状态标志

```
DEC reg/mem
; reg/mem←reg/mem－1
```

- INC指令和DEC指令都是单操作数指令
- 主要用于对计数器和地址指针的调整

第5章 微机CPU的指令系统

求补指令NEG（negative）

NEG指令对操作数执行求补运算：用零减去操作数，然后结果返回操作数

求补运算也可以表达成：将操作数按位取反后加1

NEG指令对标志的影响与用零作减法的SUB指令一样

```
NEG reg/mem
; reg/mem←0－reg/mem
```

第5章 微机CPU的指令系统

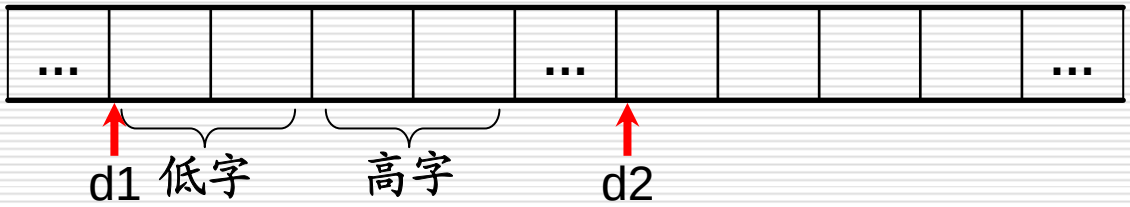
例：求补运算

```
mov ax,0ff64h
neg al
; ax=ff9ch，OF=0、SF=1、ZF=0、PF=1、CF=1
sub al,9dh
; ax=ffffh，OF=0、SF=1、ZF=0、PF=1、CF=1
neg ax
; ax=0001h，OF=0、SF=0、ZF=0、PF=0、CF=1
dec al
; ax=0000h，OF=0、SF=0、ZF=1、PF=1、CF=1
neg ax
; ax=0000h，OF=0、SF=0、ZF=1、PF=1、CF=0
```

第5章 微机CPU的指令系统

例5.4 已知有二个32位数d1和d2, 编写程序片段从d1中减去d2的值。

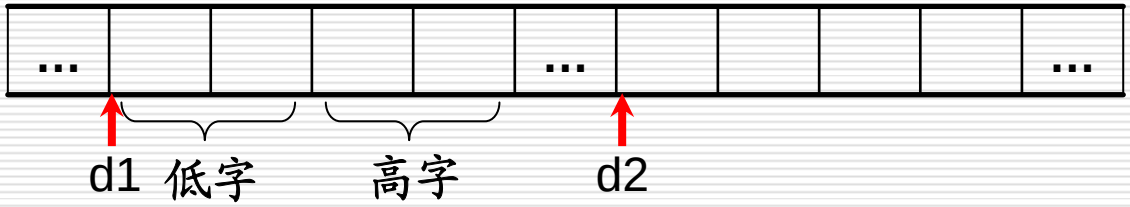
解:



方法1: 用16位寄存器编写程序

```
MOV  AX, word ptr d1        ;取低字
MOV  DX, word ptr d1+2      ;取高字, (DX, AX)构成一个32位数据
SUB  AX, word ptr d2        ;低字相减
SBB  DX, word ptr d2+2      ;高字相减。低字相减时, 有可能会产生“借位”
MOV  word ptr d1, AX        ;低字送给d1的低字
MOV  word ptr d1+2, DX      ;高字送给d1的高字
```

第5章 微机CPU的指令系统



方法2: 用32位寄存器编写程序

```
MOV  EAX,  d1
SUB  EAX,  d2
MOV  d1,  EAX
```

第5章 微机CPU的指令系统

3. 乘法指令

乘法指令分为无符号乘法和有符号乘法指令，它们的唯一区别就在于：数据的最高位是作为“数值”参与运算，还是作为“符号位”参与运算。

乘法指令的被乘数大多数是隐含的，乘数在指令中显式地写出来。CPU会根据乘数是8位、16位，还是32位操作数，来自动选用被乘数：AL、AX或EAX。

◆ 无符号数乘法指令MUL (Unsigned Multiply Instruction)

指令格式：MUL Reg/Mem

受影响的标志位：CF和OF(AF、PF、SF和ZF无定义)

指令的功能是把显式操作数和隐含操作数(都作为无符号数)相乘，所得的乘积按下表的对应关系存放。

乘数位数	隐含的被乘数	乘积的存放位置	举例
8位	AL	AX	MUL BL
16位	AX	DX - AX	MUL BX
32位	EAX	EDX - EAX	MUL ECX

第5章 微机CPU的指令系统

◆有符号数乘法指令IMUL (Signed Integer Multiply Instruction)

指令格式: 1)、IMUL Reg/Mem

2)、IMUL Reg, Imm ;80286+

3)、IMUL Reg_1, Reg_2/Mem, Imm ;80286+

4)、IMUL Reg_1, Reg_2/Mem ;80386+

1)、指令格式1 —— 该指令的功能是把显式操作数和隐含操作数(都作为有符号数)相乘, 所得的乘积按表5.2的对应关系存放。

2)、指令格式2 —— 其寄存器必须是16位/32位通用寄存器, 其计算方式为:

Reg ← Reg × Imm

3)、指令格式3 —— 其寄存器只能是16位通用寄存器, 其计算方式为:

$Reg_1 \leftarrow Reg_2 \times Imm$ 或 $Reg_1 \leftarrow Mem \times Imm$

4)、指令格式4 —— 其寄存器必须是16位/32位通用寄存器, 其计算方式为:

$Reg_1 \leftarrow Reg_1 \times Reg_2$ 或 $Reg_1 \leftarrow Reg_1 \times Mem$

在指令格式2~4中, 各操作数的位数要一致。如果乘积超过目标寄存器所能存储的范围, 则系统将置溢出标志OF为1。

第5章 微机CPU的指令系统

乘法指令如下影响OF和CF标志：

- MUL指令——若乘积的高一半（AH或DX）为0，则OF=CF=0；否则OF=CF=1
- IMUL指令——若乘积的高一半是低一半的符号扩展，则OF=CF=0；否则均为1

乘法指令对其他状态标志没有定义

- 对标志没有定义：指令执行后这些标志是任意的、不可预测（就是谁也不知道是0还是1）
- 对标志没有影响：指令执行不改变标志状态

中山大學

例2.21：乘法运算

```
mov al,0b4h          ; al=b4h=180
mov bl,11h           ; bl=11h=17
mul bl               ; ax=Obf4h=3060
; OF=CF=1，AX高8位不为0
mov al,0b4h          ; al=b4h=－76
mov bl,11h           ; bl=11h=17
imul bl              ; ax=faf4h=－1292
; OF=CF=1，AX高8位含有效数字
```

第5章 微机CPU的指令系统

4. 除法指令

除法指令的被除数是隐含操作数，除数在指令中显式地写出来。CPU会根据除数是8位、16位，还是32位，来自动选用被除数AX、DX-AX，还是EDX-EAX。

除法指令功能是用显式操作数去除隐含操作数，可得到商和余数。当除数为0，或商超出数据类型所能表示的范围时，系统会自动产生0号中断。

◆ 无符号数除法指令DIV (Unsigned Divide Instruction)

指令格式: DIV Reg/Mem

指令的功能是用显式操作数去除隐含操作数(都作为无符号数)，所得商和余数按表5.3的对应关系存放。指令对标志位的影响无定义。

除数位数	隐含的被除数	商	余 数	举 例
8位	AX	AL	AH	DIV BH
16位	DX-AX	AX	DX	DIV BX
32位	EDX-EAX	EAX	EDX	DIV ECX

第5章 微机CPU的指令系统

5. 有符号数除法指令IDIV(Signed Integer Divide Instruction)

指令格式: IDIV Reg/Mem

指令的功能是用显式操作数去除隐含操作数(都作为有符号数), 所得商和余的对应关系。

中山大學

例2.22：除法运算

```
mov ax,0400h        ;  ax=400h=1024
mov bl,0b4h         ;  bl=b4h=180
div bl              ;  商al = 05h = 5
                    ;  余数ah = 7ch = 124
mov ax,0400h        ;  ax=400h=1024
mov bl,0b4h         ;  bl=b4h= －76
idiv bl             ;  商al = f3h =  －13
                    ;  余数ah = 24h = 36
```

第5章 微机CPU的指令系统

6. 类型转换指令(Type Conversion Instruction)

系统提供了四条数据类型转换指令: CBW、CWD、CWDE和CDQ。指令的执行不影响任何标志位。

- 字节转换为字指令CBW (Convent Byte to Word)

  指令格式: CBW

  该指令的隐含操作数为AH和AL。其功能是用AL的符号位去填充AH, 即: 当AL为正数, 则AH = 0, 否则, AH = 0FFH。

- 字转换为双字指令CWD (Convent Word to Doubleword)

  指令格式: CWD

  该指令的隐含操作数为DX和AX, 其功能是用AX的符号位去填充DX。

第5章 微机CPU的指令系统

- 字转换为扩展的双字指令CWDE
  (Convent Word to Extended Doubleword)

  指令格式: CWDE ;80386+

  该指令的隐含操作数为AX和EAX, 其功能是用AX的符号位填充EAX的高位。

- 双字转换为四字指令CDQ
  (Convent Doubleword to Quadword)

  指令格式: CDQ ;80386+

  该指令的隐含操作数为EDX和EAX, 指令的功能是用EAX的符号位填充EDX。

中山大學

例2.23：符号扩展

```
mov al,80h          ; al=80h
cbw                 ; ax=ff80h
add al,255          ; al=7fh
cbw                 ; ax=007fh
```

- 利用符号扩展指令得到除法指令所需要的倍长于除数的被除数
- 对无符号数除法应该采用直接使高8位或高16位清0的方法，获得倍长的被除数

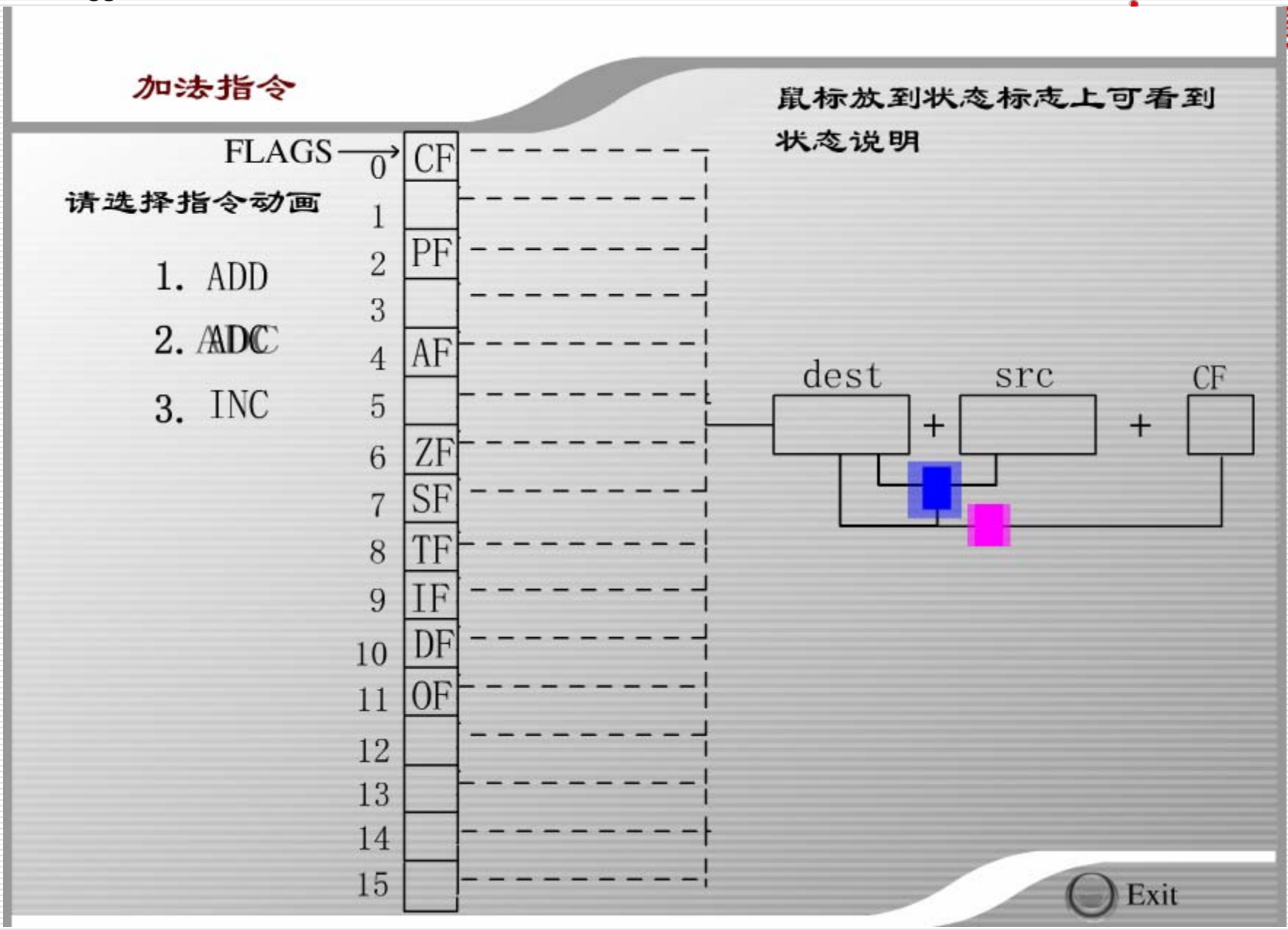
例5.5: 编写一程序段计算下面公式, 并把所得的商和余数分别存入X和Y中。

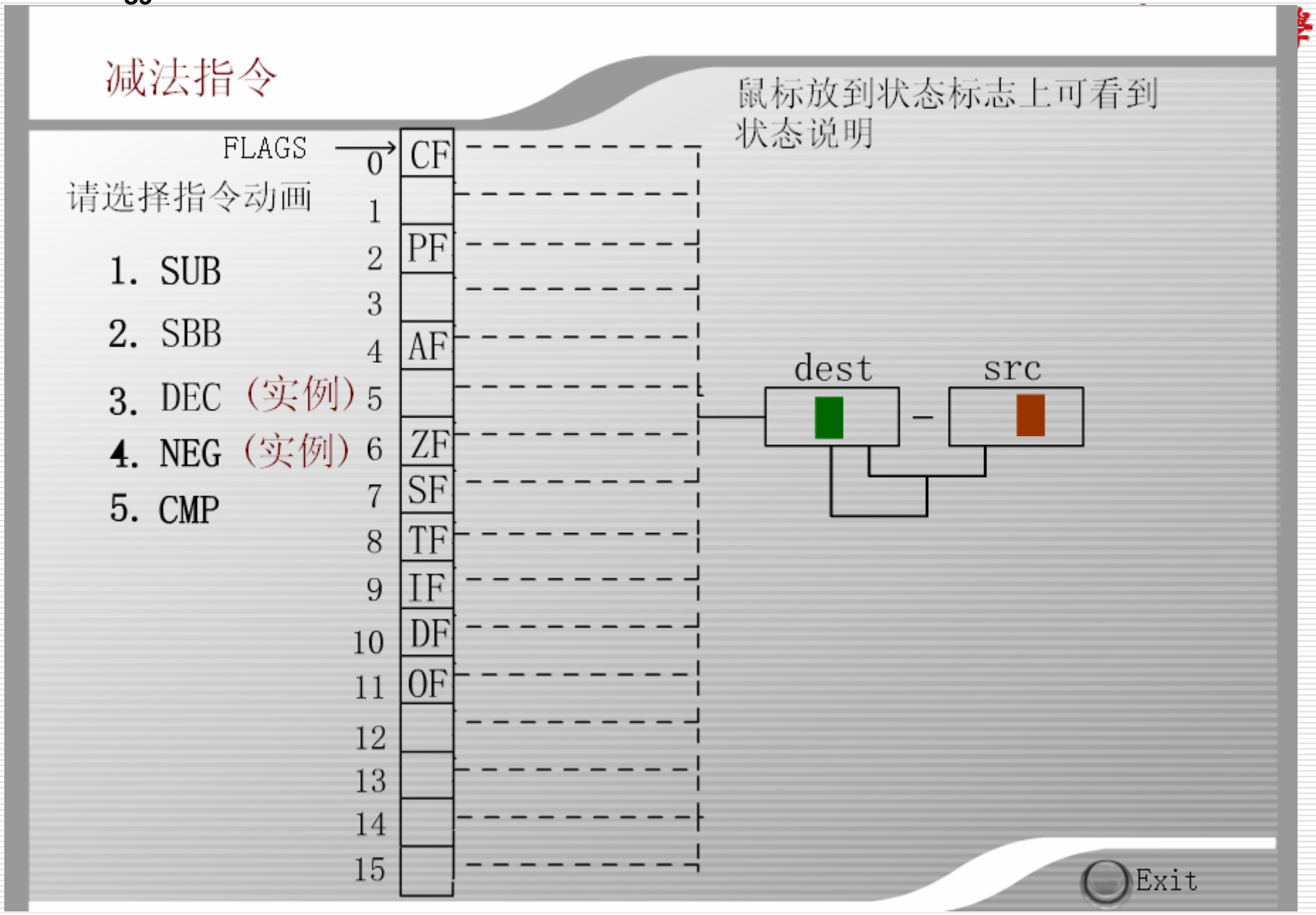
(C - 120 + A*B) / C

其中: A, B, C, X和Y都是有符号的字变量。

解:

```
    …
    A  DW  ?
    B  DW  ?
    C  DW  ?
    X  DW  ?
    Y  DW  ?
    …
    MOV   AX,  C
    SUB   AX,  120D          ;书写指令“ADD AX,  -120D”也可以
    CWD
    MOV   CX,  DX
    MOV   BX,  AX            ;(CX, BX) ← (DX, AX), 为作乘法准备必要的寄存器
    MOV   AX,  A
    IMUL   B                 ;(DX, AX) ← A*B
    ADD   AX,  BX            ;计算32位二进制之和, 为作除法作准备
    ADC    DX,  CX
    IDIV    C                ;AX是商, DX是余数
    MOV   X,  AX             ;分别保存商和余数到指定的字变量单元里
    MOV   Y,  DX
    …
```


加法指令
鼠标放到状态标志上可看到状态说明
FLAGS
请选择指令动画
1. ADD
2. ADC
3. INC
0 CF
1
2 PF
3
4 AF
5
6 ZF
7 SF
8 TF
9 IF
10 DF
11 OF
12
13
14
15
dest
src
CF
+
+
Exit


减法指令
鼠标放到状态标志上可看到状态说明
FLAGS
请选择指令动画
1. SUB
2. SBB
3. DEC (实例)
4. NEG (实例)
5. CMP
0 CF
1
2 PF
3
4 AF
5
6 ZF
7 SF
8 TF
9 IF
10 DF
11 OF
12
13
14
15
dest
src
Exit

第5章 微机CPU的指令系统

5.2.4 逻辑运算指令

逻辑运算指令是一组重要的指令, 它包括: 与、或、非和异或等指令。

◆逻辑与操作指令AND (Logical AND Instruction)

指令格式: AND Reg/Mem, Reg/Mem/Imm

指令的功能是把源操作数中的每个二进制位与目的操作数中的相应二进制进行逻辑“与操作”, 操作的结果再置回到目标操作数中。

例5.6: 已知(BH) = 67H, 要求把其的第0、1和5位置为0。

解: 可以构造一个立即数, 其第0、1和5位的值为0, 其它位的值为1, 该立即数即为: 0DCH或11011100B, 然后用指令AND来实现此功能。

```
  01100111
AND 11011100
------------
  01010100
```

AND指令设置CF = OF = 0，根据结果设置SF、ZF和PF状态，而对AF未定义

第5章 微机CPU的指令系统

◆ 逻辑或操作指令OR (Logical OR Instruction)

指令格式: OR Reg/Mem, Reg/Mem/Imm

指令的功能是把源操作数中的每个二进制位与目的操作数中的相应二进制进行逻辑“或操作”, 操作的结果再置回到目标操作数中。

例5.7: 已知(BL) = 46H, 要求把其的第1、3、4和6位置为1。

解: 构造一个立即数, 使其第1、3、4和6位的值为1, 其它位的值为0, 该立即数即5AH或01011010B, 然后用指令OR来实现此功能。

```
  01000110
OR 01011010
----------
  01011110
```

OR指令设置CF = OF = 0，根据结果设置SF、ZF和PF状态，而对AF未定义

第5章 微机CPU的指令系统

◆ 逻辑非操作指令NOT(Logical NOT Instruction)

指令格式: NOT Reg/Mem

其功能是把操作数中的每位变反, 指令的执行不影响任何标志位。

例5.8: 已知(AL) = 46H, 执行指令“NOT AL”后, AL的值是什么?

解: 执行该指令后, (AL) = 0B9H。

NOT 01000110

10111001

NOT指令是一个单操作数指令

NOT指令不影响标志位

第5章 微机CPU的指令系统

◆逻辑异或操作指令XOR (Exclusive OR Instruction)

指令格式: XOR Reg/Mem, Reg/Mem/Imm

指令的功能是把源操作数中的每个二进制位与目的操作数中的相应二进制进行逻辑“异或操作”, 操作的结果再置回到目标操作数中。

例5.9: 已知(AH) = 46H, 要求把其的第0、2、5和7位的二进制值变反。

解: 构造一个立即数, 使其第0、2、5和7位的值为1, 其它位的值为0, 该立即数即为: 0A5H或10100101B, 然后再用指令XOR来实现此功能。

```
  01000110
XOR 10100101
----------
  11100011
```

XOR指令设置CF = OF = 0，根据结果设置SF、ZF和PF状态，而对AF未定义

中山大學

例题：逻辑运算

```
mov al,45h          ; 逻辑与 al=01h
and al,31h          ; CF=OF=0，SF=0、ZF=0、PF=0

mov al,45h          ; 逻辑或 al=75h
or al,31h           ; CF=OF=0，SF=0、ZF=0、PF=0

mov al,45h          ; 逻辑异或 al=74h
xor al,31h          ; CF=OF=0，SF=0、ZF=0、PF=1

mov al,45h          ; 逻辑非 al=0bah
not al              ; 标志不变
```

中山大學

例题：逻辑指令应用

；AND指令可用于复位某些位（同0相与），不影响其他位：将BL中D3和D0位清0，其他位不变

```
and bl,11110110B
```

；OR指令可用于置位某些位（同1相或），不影响其他位：将BL中D3和D0位置1，其他位不变

```
or bl, 00001001B
```

；XOR指令可用于求反某些位（同1相异或），不影响其他位：将BL中D3和D0位求反，其他不变

```
xor bl, 00001001B
```

第5章 微机CPU的指令系统

5.2.5 移位操作指令

移位操作指令包括算术移位、逻辑移位、循环移位、双精度移位和带进位的循环移位等五大类。

移位指令都有指定移动二进制位数的操作数，该操作数可以是立即数或CL的值。在8086中，该立即数只能为1，但在其后的CPU中，该立即数可以是1~31之内的数。

◆算术移位指令

指令有：算术左移SAL(Shift Algebraic Left)和算术右移SAR(Shift Algebraic Right)。它们的指令格式如下：

SAL/SAR Reg/Mem, CL/Imm

算术左移SAL把目的操作数的低位向高位移，空出的低位补0；

算术右移SAR把目的操作数的高位向低位移，空出的高位用最高位填补。

第5章 微机CPU的指令系统

例5.10: 已知(AH) = 12H, (BL) = 0A9H, 试给出分别用算术左移和右移指令移动1位后, 寄存器AH和BL的内容。

解: 用算术左移和右移指令移动1位后, 寄存器AH和BL的结果如下表所示。

操作数的初值	执行的指令	执行后操作数的内容
(AH) = 12H	SAL AH, 1	(AH) = 24H
(BL) = 0A9H	SAL BL, 1	(BL) = 52H
(AH) = 12H	SAR AH, 1	(AH) = 09H
(BL) = 0A9H	SAR BL, 1	(BL) = 0D4H

思考题: 下面有两组指令序列, 问每组指令执行后, 寄存器AX的不会变化吗?

```
SAL  AX, 1        或        SAR  AX, 1
SAR  AX, 1                  SAL  AX, 1
```

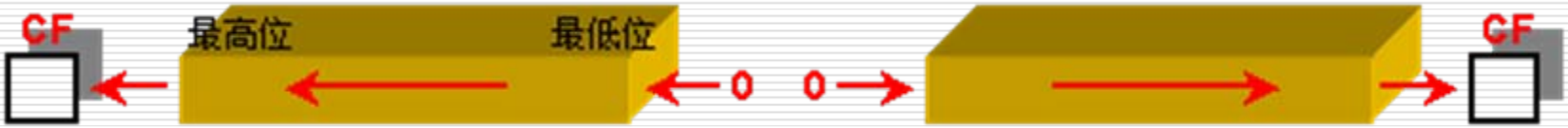
第5章 微机CPU的指令系统

◆逻辑移位指令

此组指令有: 逻辑左移SHL (Shift Logical Left)和逻辑右移SHR (Shift Logical Right)。

SHL/SHR Reg/Mem, CL/Imm

逻辑左移/右移指令只有它们的移位方向不同, 移位后空出的位都补0。



例5.11: 已知(AH) = 12H, (BL) = 0A9H, 试给出分别用逻辑左移和右移指令移动1位后, 寄存器AH和BL的内容。

解: 用算术左移和右移指令移动1位后, 寄存器AH和BL的结果如下表所示。

操作数的初值	执行的指令	执行后操作数的内容
(AH) = 12H	SHL AH, 1	(AH) = 24H
(BL) = 0A9H	SHL BL, 1	(BL) = 52H
(AH) = 12H	SHR AH, 1	(AH) = 09H
(BL) = 0A9H	SHR BL, 1	(BL) = 54H

- 移位指令的第一个操作数是指定的被移位的操作数，可以是寄存器或存储单元
- 后一个操作数表示移位位数，该操作数为1，表示移动一位；当移位位数大于1时，则用CL寄存器值表示，该操作数表达为CL

- 按照移入的位设置进位标志CF
- 根据移位后的结果影响SF、ZF、PF
- 对AF没有定义
- 如果进行一位移动，则按照操作数的最高符号位是否改变，相应设置溢出标志OF：如果移位前的操作数最高位与移位后操作数的最高位不同（有变化），则OF = 1；否则OF = 0。当移位次数大于1时，OF不确定

例2.33：移位指令

```
mov cl,4
mov al,0f0h                ; al=f0h
shl al,1                   ; al=e0h
; CF=1, SF=1、ZF=0、PF=0, OF=0
shr al,1                   ; al=70h
; CF=0, SF=0、ZF=0、PF=0、OF=1
sar al,1                   ; al=38h
; CF=0, SF=0、ZF=0、PF=0、OF=0
sar al,cl                  ; al=03h
; CF=1, SF=0、ZF=0、PF=1
```

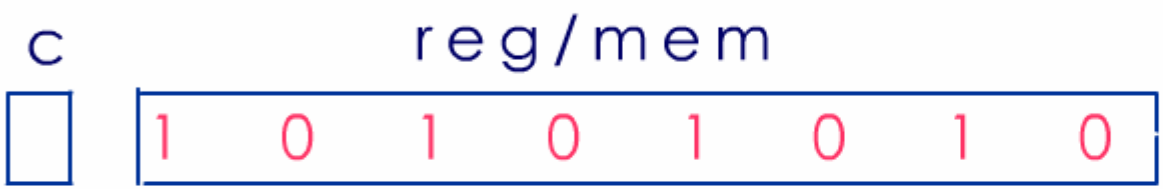
中山大學

例2.34：移位实现乘

```
mov si,ax
shl si,1                ; si←2×ax
add si,ax               ; si←3×ax
mov dx,bx
mov cl,03h
shl dx,cl               ; dx←8×bx
sub dx,bx               ; dx←7×bx
add dx,si               ; dx←7×bx+3×ax
```

- 逻辑左移一位相当于无符号数乘以2
- 逻辑右移一位相当于无符号数除以2

SHL reg/mem,1 (SAL reg/mem, 1)

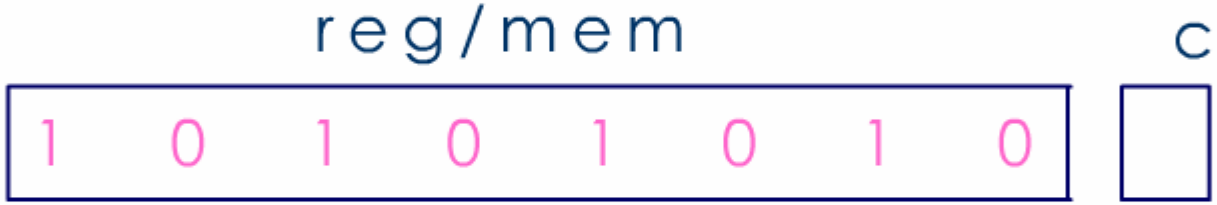


播放

返回

SHL / SAL指令

SHR reg/mem,1



SHR指令

SAR指令

第5章 微机CPU的指令系统

◆双精度移位指令

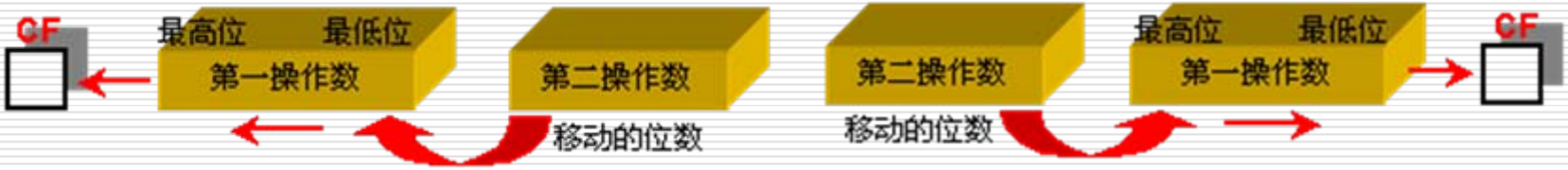
此组指令有：双精度左移SHLD (Shift Left Double)和双精度右移SHRD (Shift Right Double)。它们都是具有三个操作数的指令：

SHLD/SHRD Reg/Mem, Reg, CL/Imm ;80386+

其中：第一操作数是一个16位/32位的寄存器或存储单元;第二操作数(与前者具有相同位数)一定是寄存器;第三操作数是移动的位数，它可由CL或一个立即数来确定。

在执行SHLD指令时，第一操作数向左移n位，其“空出”的低位由第二操作数的高n位来填补，但第二操作数自己不移动、不改变。

在执行SHRD指令时，第一操作数向右移n位，其“空出”的高位由第二操作数的低n位来填补，但第二操作数自己也不移动、不改变。



第5章 微机CPU的指令系统

◆循环移位指令

指令有：循环左移ROL (Rotate Left)和循环右移ROR (Rotate Right)。

ROL/ROR Reg/Mem, CL/Imm

循环左移/右移指令只有移位的方向不同，它们移出的位不仅要进入CF，而且要填补空出的位。

第5章 微机CPU的指令系统

下面是几个循环移位的例子及其执行结果。

循环移位指令	指令操作数的初值	指令执行后的结果
ROL AX, 1	(AX) = 6789H	(AX) = 0CF12H
ROL AX, 3	(AX) = 6789H	(AX) = 3C4BH
ROR AX, 2	(AX) = 6789H	(AX) = 59E2H
ROR AX, 4	(AX) = 6789H	(AX) = 9678H

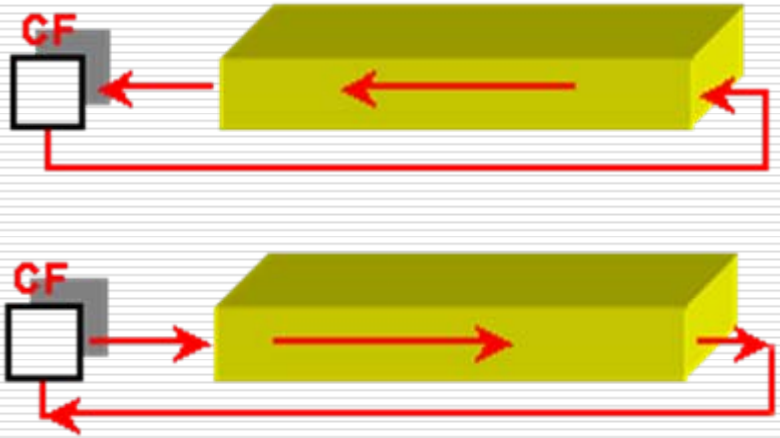
第5章 微机CPU的指令系统

◆带进位的循环移位指令

带进位的循环移位指令有：带进位的循环左移RCL (Rotate Left Through Carry)和带进位的循环右移RCR (Rotate Right)。

RCL/RCR Reg/Mem, CL/Imm

带进位的循环左移/右移指令只有移位的方向不同，它们都用原CF的值填补空出的位，移出的位再进入CF。



第5章 微机CPU的指令系统

下面是几个带进位循环移位的例子及其执行结果。

双精度移动指令	指令操作数的初值	指令执行后的结果
RCL AX, 1	CF = 0, (AX) = 0ABCDH	(AX) = 579AH
RCL AX, 1	CF = 1, (AX) = 0ABCDH	(AX) = 579BH
RCR AX, 2	CF = 0, (AX) = 0ABCDH	(AX) = AAF3H
RCR AX, 2	CF = 1, (AX) = 0ABCDH	(AX) = EAF3H

例5.12: 编写指令序列把由DX和AX组成的32位二进制算术左移、循环左移1位。

解: (DX, AX)算术左移1位指令序列

```
SHL  AX, 1
RCL  DX, 1
```

(DX, AX)循环左移1位指令序列

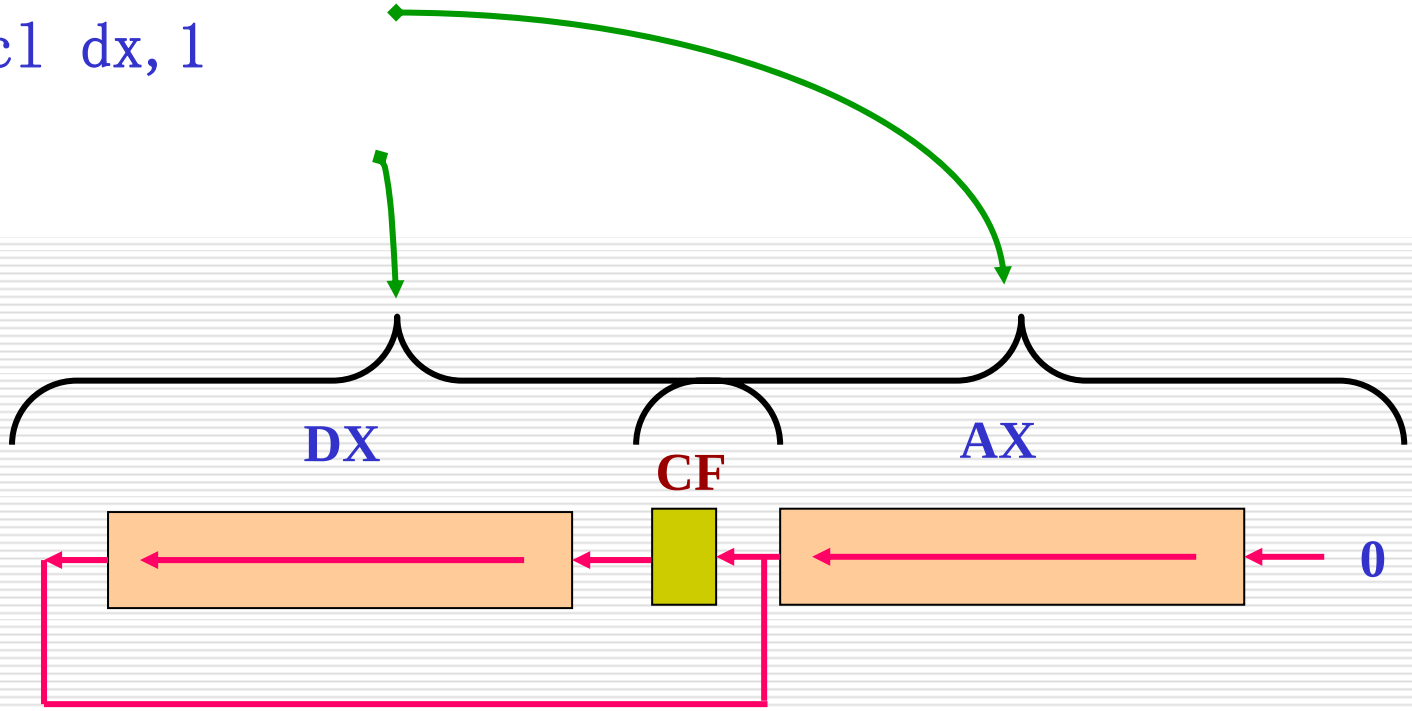
```
SHLD  DX, AX, 1
RCL   AX, 1
```

- 按照指令功能设置进位标志CF
- 不影响SF、ZF、PF、AF
- 如果进行一位移动，则按照操作数的最高符号位是否改变，相应设置溢出标志OF：如果移位前的操作数最高位与移位后操作数的最高位不同（有变化），则OF = 1；否则OF = 0。当移位次数大于1时，OF不确定

中山大學

例2.35：32位数移位

```
; 将DX.AX中32位数值左移一位
shl  ax,1
rcl  dx,1
```



第5章 微机CPU的指令系统

中山大學

5.2.6 位操作指令

1. 位扫描指令(Bit Scan Instruction)

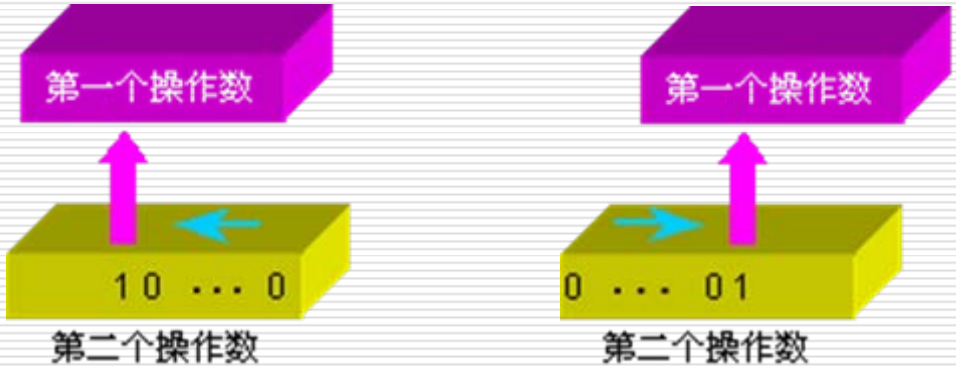
BSF/BSR Reg, Reg/Mem ;80386+

第一操作数不能是段寄存器和8位寄存器，第二操作数的类型与前者相同。

位扫描指令是在第二个操作数中找第一个“1”的位置。如果找到，则该“1”的置保存在第一操作数中，并置标志位ZF为1，否则，置标志位ZF为0。

根据位扫描的方向不同，指令分二种：正向扫描指令和逆向扫描指令。

- 正向扫描指令**BSF(Bit Scan Forward)**从右向左扫描，从低位向高位扫描
- 逆向扫描指令**BSR(Bit Scan Reverse)**从左向右扫描，从高位向低位扫描



第5章 微机CPU的指令系统

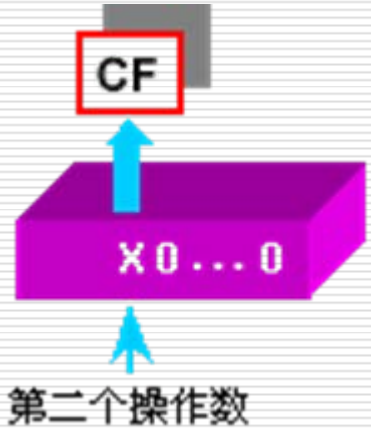
2. 位检测指令(Bit Test Instruction)

BT/BTC/BTR/BTS Reg/Mem, Reg/Imm ;80386+

指令中的操作数不能是段寄存器和8位寄存器。

位检测指令是把第一个操作数中某一位的值传送给标志位CF, 具体的哪一位由指令的第二操作数来确定, 若第二操作数的值超过第一操作数的位数, 那么, 该指令不作任何操作。根据指令中对具体位的处理不同, 又分一下几种指令:

- **BT:** 把指定的位传送给CF
- **BTC:** 把指定的位传送给CF后, 还使该位变反
- **BTR:** 把指定的位传送给CF后, 还使该位变为0
- **BTS:** 把指定的位传送给CF后, 还使该位变为1



第5章 微机CPU的指令系统

例如: 假设(AX) = 1234H, 分别执行下面指令。

```
BT    AX,  2      ;指令执行后, CF = 1, (AX) = 1234h
BTC   AX,  6      ;指令执行后, CF = 0, (AX) = 1274h
BTR   AX,  10     ;指令执行后, CF = 0, (AX) = 1234h
BTS   AX,  14     ;指令执行后, CF = 0, (AX) = 5234h
```

第5章 微机CPU的指令系统

3. 检测位指令TEST (Test Bits Instruction)

检测位指令是把二个操作数进行逻辑“与”操作，并根据运算结果设置相应的标志位，但并不保存该运算结果，所以，不会改变指令中的操作数。在该指令后，通常用JE、JNE、JZ和JNZ等条件转移指令。

TEST Reg/Mem, Reg/Mem/Imm

例如：

```
TEST   AX, 1          ;测试AX的第0位
TEST   CL, 10101B     ;测试CL的第0、2、4位
```

TEST指令通常用于检测一些条件是否满足，但又不希望改变原操作数的情况

第5章 微机CPU的指令系统

5.2.7 比较运算指令

在程序中，我们要时经常根据某个变量或表达式的取值去执行不同指令，从而使程序表现出有不同的功能。为了配合这样的操作，在CPU的指令系统中提供了各种不同的比较指令。通过这些比较指令的执行来改变有关标志位，为进行条件转移提供依据。

1. 比较指令CMP (Compare Instruction)

CMP Reg/Mem, Reg/Mem/Imm

指令的功能：用第二个操作数去减第一个操作数，并根据所得的差设置有关标志位，为随后的条件转移指令提供条件。但并不保存该差，所以，不会改变指令中的操作数。

第5章 微机CPU的指令系统

2. 比较交换指令(Compare And Exchange Instruction)

前面介绍了交换指令XCHG, 不管二个操作数的值是什么, 都无条件地进行交换。而比较交换指令, 是先进行比较, 再根据比较的结果决定是否进行操作数的交换操作。

比较交换指令的功能: 当二个操作数相等时, 置标志位ZF为1;否则, 把第一操作数的值赋给第二操作数, 并置标志位ZF为0。

- 8位/16位/32位比较交换指令

```
CMPXCHG  Reg/Mem,  AL/AX/EAX          ;80486+
```

- 64位比较交换指令

该指令只有一个操作数, 第二个操作数EDX: EAX是隐含的。

```
CMPXCHG8B  Reg/Mem                    ;Pentium+
```

3. 字符串比较指令(Compare String Instruction)

参见后面第5.2.11节字符串操作类指令中的叙述。

第5章 微机CPU的指令系统

5.2.8 循环指令

循环结构是程序的三大结构之一。汇编语言提供了多种循环指令，这些循环指令的循环次数都是保存在计数器CX或ECX中。除了CX或ECX可以决定循环是否结束外，有的循环指令还可由标志位ZF来决定是否结束循环。

汇编语言中，CX或ECX只能递减，所以，循环计数器只能从大到小。在程序中，必须先把循环次数赋给循环计数器。

汇编语言的循环指令放在循环体的下面。循环时，首先执行一次循环体，然后把循环计数器CX或ECX减1。当循环终止条件达到满足时，该循环指令下面的指令将是下一条被执行的指令，否则，程序向上转到循环体的第一条指令。

在循环未终止，而向上转移时，规定：该转移只能是一个短转移，即偏移量不能超过128，也就是说循环体中所有指令码的字节数之和不能超过128。如果循环体过大，可以用后面介绍的“转移指令”来构造循环结构。

循环指令本身的执行不影响任何标志位。

第5章 微机CPU的指令系统

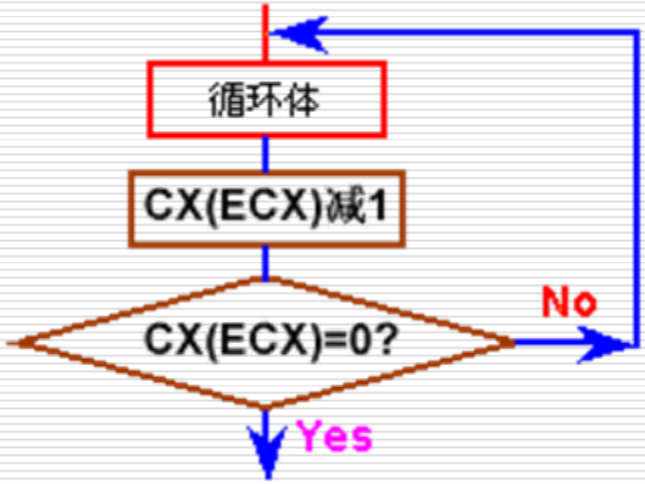
1. 循环指令(Loop Until Complete)

循环指令LOOP的一般格式:

```
LOOP   标号
LOOPW  标号          ;CX作为循环计数器, 80386+
LOOPD  标号          ;ECX作为循环计数器, 80386+
```

循环指令的功能如下图所示, 其文字描述如下:

- (CX) = (CX)-1或(ECX) = (ECX)-1;
- 如果(CX)≠0或(ECX)≠0, 转向“标号”所指向的指令, 否则, 终止循环, 执行该指令下面的指令。



第5章 微机CPU的指令系统

例5.13: 编写一段程序, 求1+2+…+100之和, 并把结果存入AX中。

解:

方法1: 因为计数器CX只能递减, 可把求和式子改变为: 100+99+…+2+1。

```
        …
        XOR     AX,  AX
        MOV     CX,  100D
again:  ADD     AX,  CX          ;计算过程: 100+99+…+2+1
        LOOP    again
        …
```

方法2: 不用循环计数器进行累加, 求和式子仍为: 1+2+…+99+100。

```
        …
        XOR     AX,  AX
        MOV     CX,  100D
        MOV     BX,  1
again:  ADD     AX,  BX          ;计算过程: 1+2+…+99+100
        INC     BX
        LOOP    again
```

从程序段的效率来看: 方法1要比方法2好。

第5章 微机CPU的指令系统

2. 相等或为零循环指令(Loop While Equal or Loop While Zero)

相等或为零循环指令的一般格式:

```
LOOPE/LOOPZ  标号
LOOPEW/LOOPZW  标号          ;CX作为循环计数器, 80386+
LOOPED/LOOPZD  标号          ;ECX作为循环计数器, 80386+
```

这是一组有条件循环指令, 它们除了要受CX或ECX的影响外, 还要受标志位ZF的影响。其具体规定如下:

- (CX) = (CX)-1或(ECX) = (ECX)-1; (不改变任何标志位)
- 如果循环计数器 ≠ 0且ZF = 1, 则程序转到循环体的第一条指令, 否则, 序将执行该循环指令下面的指令。

第5章 微机CPU的指令系统

指令LOOPE/LOOPZ所组成的循环功能如下图所示，指令LOOPEW/LOOPZW、LOOPED/LOOPZD与之功能相一致，所不同就在于循环寄器是CX，还是ECX。

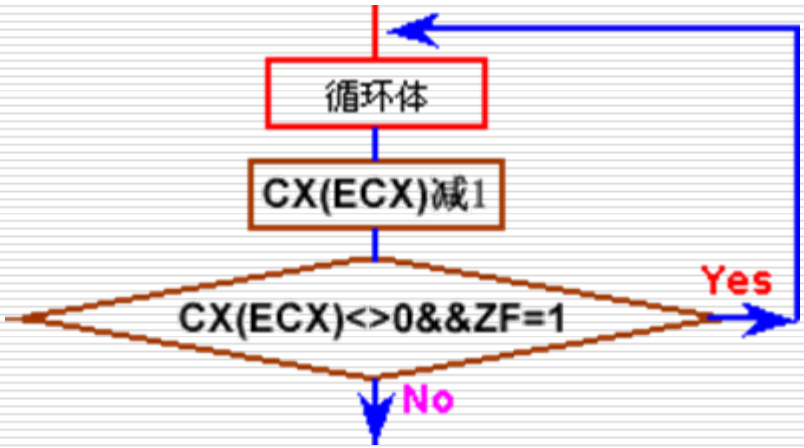


图5.13 LOOPE的功能示意图

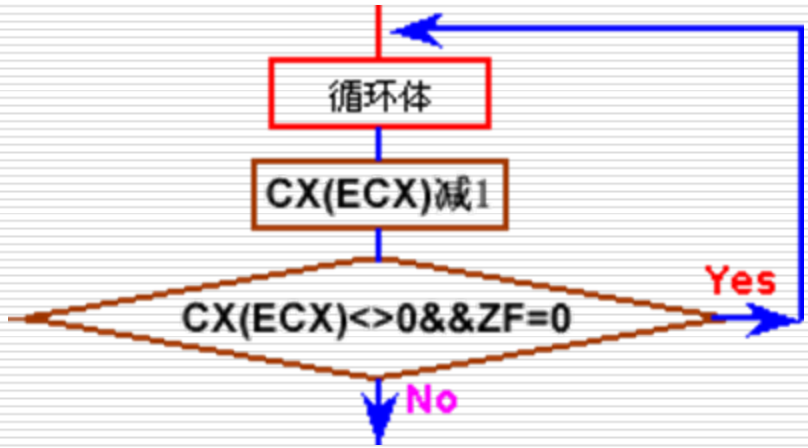


图5.14 LOOPNE的功能示意图

第5章 微机CPU的指令系统

3. 不等或不为零循环指令(Loop While Not Equal or Loop While Not Zero)

不等或不为零循环指令的一般格式:

```
LOOPNE/LOOPNZ  标号
LOOPNEW/LOOPNZW  标号        ;CX作为循环计数器, 80386+
LOOPNED/LOOPNZD  标号        ;ECX作为循环计数器, 80386+
```

这也是一组有条件循环指令, 它们与相等或为零循环指令在循环结束条件上点不同。其具体规定如下:

- (CX) = (CX)-1或(ECX) = (ECX)-1; (不改变任何标志位)
- 如果循环计数器 ≠ 0且ZF = 0, 则程序转到循环体的第一条指令, 否则, 程序将执行该循环指令下面的指令。

第5章 微机CPU的指令系统

4. 循环计数器为零转指令(Jump if CX/ECX is Zero)

在前面的各类循环指令中，不管循环计数器的初值为何，循环体至少会被执行一次。当循环计数器的初值为0时，通常的理解应是循环体被循环0次，即循环体一次也不被执行。其实不然，循环体不是不被执行，而是会被执行65536次(用CX计或4294967296次(几乎是死循环，用ECX计数)。

为了解决指令的执行和常规思维之间差异，指令系统又提供了一条与循环计器有关的指令——循环计数器为零转指令。

该指令一般用于循环的开始处，其指令格式如下：

```
JCXZ  标号      ;当(CX) = 0时，则程序转移标号处执行。
JECXZ 标号      ;当(ECX) = 0时，则程序转移标号处执行。80386+
```

第5章 微机CPU的指令系统

例5.14: 编写一段程序, 求1+2+…+k(K⩾0)之和, 并把结果存入AX中。

解:

```
            …
            K       DW      ?       ;变量定义
            …
            XOR     AX,  AX         ;AX置初值0
            MOV     CX,  K
            JCXZ    next
again:      ADD     AX,  CX         ;计算过程:  K+(K-1)+…+2+1
            LOOP    again
next:       …
```

第5章 微机CPU的指令系统

5.2.9 转移指令

转移指令是汇编语言程序员经常使用的一组指令。

在高级语言中，时常有“尽量不要使用转移语句”的劝告，但如果在汇编语言的程序中也尽量不用转移语句，那么该程序要么无法编写，要么没有多少功能，以，在汇编语言中，不但要使用转移指令，而且还要灵活运用。

转移指令分无条件转移指令和有条件转移指令两大类。

第5章 微机CPU的指令系统

1、无条件转移指令(Transfer Unconditionally)

无条件转移指令: JMP、子程序的调用和返回指令、中断调用和返回指令等。

无条件转移指令JMP(Unconditional Jump)的一般形式:

JMP 标号/Reg/Mem

JMP指令是从程序当前执行的地方无条件转移到另一个地方执行。这种转移可以是一个短(short)转移(偏移量在[-128, 127]范围内), 近(near)转移(偏移量在[-32K, 32K]范围内)或远(far)转移(在不同的代码段之间转移)。

短和近转移是段内转移, JMP指令只把目标指令位置的偏移量赋值指令指针寄存器IP, 从而实现转移功能。但远转移是段间转移, JMP指令不仅会改变指令指针寄存器IP的值, 而且还会改变代码段寄存器CS的值。

第5章 微机CPU的指令系统

例如:

```
        …
next1:  …
        JMP next1                ;向前转移, 偏移量之差为负数
        …
        JMP next2                ;向后转移, 偏移量之差为正数
        …
next2:  …
```

当段内转移时, 有些软件把该转移指令默认为近转移, 从而使指令的偏移量用一个字来表示, 于是生成3个字节的指令代码。如果程序员清楚转移的幅度在一个短转移的范围之内, 那么, 可用前置short的办法来产生2个字节的指令代码。

比如: 如果程序员知道偏移量不会超过127, 那么, 可产生省掉一个字节的指令代码。

```
        …
        JMP short next2          ;生成2个字节的转移指令, 从而节省一个字节
        …
next2:  …
```

第5章 微机CPU的指令系统

2、条件转移指令(Transfer Conditionally)

条件转移指令是一组极其重要的转移指令，它根据标志寄存器中的一个(或个)标志位来决定是否需要转移，这就为实现多功能程序提供了必要的手段。微的指令系统提供了丰富的条件转移指令来满足各种不同的转移需要，在编程序时，要对它们灵活运用。

条件转移指令分三大类：基于无符号数的条件转移指令、基于有符号数的条件转移指令和基于特殊算术标志位的条件转移指令。

第5章 微机CPU的指令系统

◆ 无符号数的条件转移指令
(Jumps Based on Unsigned (Logic) Data)

指令的助忆符	检测的转移条件	功能描述
JE/JZ	ZF=1	若相等或为0, 则转移
JNE/JNZ	ZF=0	若不等或不为0, 则转移
JA/JNBE	CF=0 and ZF=0	若“高于”或“不低于、等于”, 则转移
JAE/JNB	CF=0	若“高于、等于”或“不低于”, 则转移
JB/JNAE	CF=1	若“低于”或“不高于、等于”, 则转移
JBE/JNA	CF=1 or AF=1	若“低于、等于”或“不高于”, 则转移

第5章 微机CPU的指令系统

◆有符号数的条件转移指令
(Jumps Based on Signed (Arithmetic) Data)

指令的助忆符	检测的转移条件	功能描述
JE/JZ	ZF=1	若相等或为0, 则转移
JNE/JNZ	ZF=0	若不等或不为0, 则转移
JG/JNLE	ZF=0 and SF=OF	若“大于”或“不小于、等于”, 则转移
JGE/JNL	SF=OF	若“大于、等于”或“不小于”, 则转移
JL/JNGE	SF≠OF	若“小于”或“不大于、等于”, 则转移
JLE/JNG	ZF=1 or SF≠OF	若“小于、等于”或“不大于”, 则转移

第5章 微机CPU的指令系统

◆特殊算术标志位的条件转移指令
(Jumps Based on Special Arithmetic Tests)

指令的助忆符	检测的转移条件	功能描述
JC	CF=1	若有进位, 则转移
JNC	CF=0	若无进位, 则转移
JO	OF=1	若有溢出, 则转移
JNO	OF=0	若无溢出, 则转移
JP/JPE	PF=1	若有偶数个'1', 则转移
JNP/JPO	PF=0	若有齐数个'1', 则转移
JS	SF=1	若为负数, 则转移
JNS	SF=0	若为正数, 则转移

第5章 微机CPU的指令系统

例5.15: 编写一程序段, 它把寄存器AX-BX的绝对值存入BX中。

解:

```
        …
        SUB     BX,  AX
        JNS     next
        NEG     BX
next:   …
```

例5.16: 已知字节变量char, 编写程序段把其所存的大写字母变成小写字母。

解:

```
        …
        char  DB  'F'                ;变量说明
        …
        MOV     AL,  char
        CMP     AL,  'A'
        JB      next                 ;注意: 字符是无符号数, 不要使用指令JL
        CMP     AL,  'Z'
        JA      next
        ADD     char,  20H           ;小写字母比大写字母的ASCII码大20H
next:   …
```

第5章 微机CPU的指令系统

例5.17: 编写一段程序, 完成下面计算公式, 其中: 变量X和Y都是字类型。

$$Y=\begin{cases} X & X\le 0 \\ X+100 & 0<X\le 500 \\ X-50 & X>500 \end{cases}$$

解:

```
            ...
            X  DW  ?                ;变量说明
            Y  DW  ?
            ...
            MOV     AX,  X
            MOV     BX,  AX         ;用BX来临时存放计算结果
            CMP     AX,  0
            JLE     setdata
            CMP     AX,  500
            JG      case3
            ADD     BX,  100D       ;BX = X+100
            JMP     setdata
case3:      SUB     BX,  50D        ;BX = X-50
setdata:    MOV     Y,  BX          ;把计算结果赋给变量Y
            ...
```

第5章 微机CPU的指令系统

例5.18: 下面循环体的指令代码字节数超过128, 试改写该循环。

```
            …
            MOV     CX,  COUNT      ;给循环计数器赋初值(>0)
again:      循环体指令序列          ;循环体的首地址偏移量大于128
            LOOP    again
```

解:

```
            …
            MOV     CX,  COUNT
again:      循环体指令序列
            DEC     CX
            JNZ     again           ;把LOOP指令改为条件转移指令
```

第5章 微机CPU的指令系统

5.2.10 条件设置字节指令

条件设置字节指令(Set Byte Conditionally)是80386及其以后CPU所具有的一组指令。它们在测试条件方面与条件转移是一致的,但在功能方面,它们不是转移,而是根据测试条件的值来设置其字节操作数的内容为1或0。

条件设置字节指令的一般格式如下:

```
SETnn  Reg/Mem                    ;80386+
```

其中: nn是表示测试条件的(见后表),操作数只能是8位寄存器或一个字节元。

这组指令的执行不影响任何标志位。

第5章 微机CPU的指令系统

条件设置字节指令列表

指令的助忆符	操作数和检测条件之间的关系
SETZ/SETE	Reg/Mem = ZF
SETNZSETNE	Reg/Mem = not ZF
SETS	Reg/Mem = SF
SETNS	Reg/Mem = not SF
SETO	Reg/Mem = OF
SETNO	Reg/Mem = not OF
SETP/SETPE	Reg/Mem = PF
SETNP/SETPO	Reg/Mem = not PF
SETC/SETB/SETNAE	Reg/Mem = CF
SETNC/SETB/SETAE	Reg/Mem = not CF
SETNA/SETBE	Reg/Mem = (CF or ZF)
SETA/SETNBE	Reg/Mem = not (CF or ZF)
SETL/SETNGE	Reg/Mem = (SF xor OF)
SETNL/SETGE	Reg/Mem = not (SF xor OF)
SETLE/SETNG	Reg/Mem = (SF xor OF) or ZF
SETNLE/SETG	Reg/Mem = not ((SF xor OF) or ZF)

第5章 微机CPU的指令系统

例5.19: 检测寄存器EAX中有几个16进制的0H, 并把统计结果存入BH中。

解:

方法1: 用条件转移指令来实现

```
        XOR     BH, BH
        MOV     CX, 8           ;测试寄存器EAX——8次
again:  TEST    AL, 0FH         ;测试低四位二进制是否为0H
        JNZ     next
        INC     BH
next:   ROR     EAX, 4          ;循环向右移四位, 为测试高四位作准备
        LOOP    again
```

方法2: 用条件设置字节指令来实现

```
        XOR     BH, BH
        MOV     CX, 8           ;测试寄存器EAX——8次
again:  TEST    AL, 0FH         ;测试低四位二进制是否为0H
        SETZ    BL              ;AL低四位是0, 则BL置为1, 否则, BL为0
        ADD     BH, BL
        ROR     EAX, 4
        LOOP    again
```

第5章 微机CPU的指令系统

5.2.11 字符串操作指令

字符串操作指令的实质是对一片连续存储单元进行处理，这片存储单元是由隐含指针DS: SI或ES: DI来指定的。

字符串操作指令可对内存单元按字节、字或双字进行处理，并能根据操作对象的字节数使变址寄存器SI(和DI)增减1、2或4。

具体规定如下:

(1)、当DF=0时，变址寄存器SI(和DI)增加1、2或4;

(2)、当DF=1时，变址寄存器SI(和DI)减少1、2或4。

在后面的其它字符串指令中，变址寄存器都按上述规定进行增减，并不再说明。

第5章 微机CPU的指令系统

中山大學

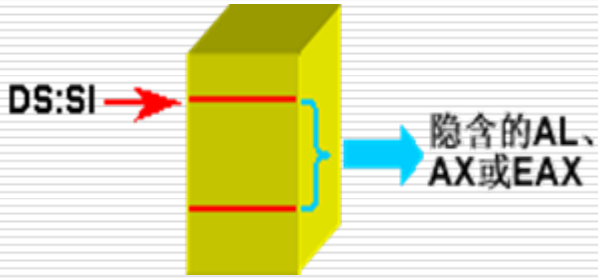
1. 取字符串数据指令(Load String Instruction)

从由指针DS: SI所指向的内存单元开始, 取一个字节、字或双字进入AL、AX或EAX中, 并根据标志位DF对寄存器SI作相应增减。

指令格式: LODS 地址表达式

LODSB/LODSW

LODSD ;80386+



指令LODSB是从指针DS: SI处读一个字节到AL中;

指令LODSW是从指针DS: SI处读一个字到AX中;

指令LODSD是从指针DS: SI处读一个双字到EAX中;

指令LODS将根据其地址表达式的属性来决定读取一个字节、字或双字。即: 当该地址表达式的属性为字节、字或双字时, 将从指针DS: SI处读一个字节, 或一个字, 或一个双字, 与此同时, SI还将分别增减1, 2或4。

其它字符串指令中的“地址表达式”作用与此类似, 后面将不再说明。

第5章 微机CPU的指令系统

中山大學

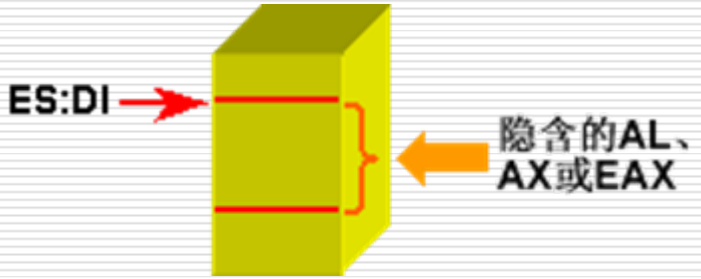
2. 置字符串数据指令(Store String Instruction)

该指令是把寄存器AL、AX或EAX中的值存于以指针ES: DI所指向内存单元为起始的一片存储单元里, 并根据标志位DF对寄存器DI作相应增减。

指令格式: STOS 地址表达式

STOSB/STOSW

STOSD ;80386+



指令STOSB是把AL中的内容存入由指针ES: DI所指向的字节;

指令STOSW是把AX中的内容存入由指针ES: DI所指向的字;

指令STOSD是把EAX中的内容存入由指针ES: DI所指向的双字;

指令STOS将根据其地址表达式的属性(字节、字或双字)来决定把AL、AX还是EAX存入ES: DI所指向的内存单元。其具体的对应关系与指令LODS相一致。

第5章 微机CPU的指令系统

中山大學

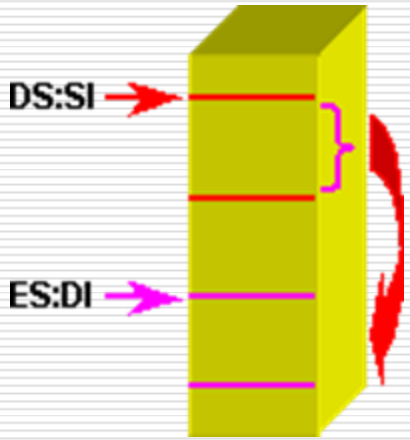
3. 字符串传送指令(Move String Instruction)

该指令是把指针DS: SI所指向的字节、字或双字传送给指针ES: DI所指向内存单元, 并根据标志位DF对寄存器DI和SI作相应增减。

指令格式: MOVS 地址表达式1, 地址表达式2

MOVSB/MOVSW

MOVSD ;80386+



指令功能的文字描述如下:

指令MOVSB是从DS: SI向ES: DI出传送一个字节;

指令MOVSW是从DS: SI向ES: DI出传送一个字;

指令MOVSD是从DS: SI向ES: DI出传送一个双字;

指令MOVS将根据其地址表达式的属性(字节、字或双字)来决定从源字符串送多少个字节到目的字符串位置。其具体的对应关系与指令LODS相一致。

第5章 微机CPU的指令系统

4. 重复前缀指令REP(Repeat String Instruction)

重复前缀指令是重复其后字符串操作指令，重复次数由CX来决定。

REP LODS/LODSB/LODSW/LODSD

REP STOS/STOSB/STOSW/STOSD

REP MOVS/MOVSB/MOVSW/MOVSD

REP INS/INSB/INSW/INSD

REP OUTS/OUTSB/OUTSW/OUTSD

重复前缀指令的执行步骤如下：

- 判断: CX = 0;
- 如果CX = 0, 则结束重复操作, 执行程序中的下一条指令;
- 否则, CX = CX-1(不影响有关标志位), 并执行其后的字符串操作指令, 在该指令执行完后, 再转到步骤(1)。

第5章 微机CPU的指令系统

例5.20: 编写一段程序, 计算字符串“12345abcdefgh”中字符的ASCII之和。

解:

```
        …
        MESS  DB  '12345abcdefgh'          ;在数据段中进行变量说明
        …
        MOV     AX,  SEG MESS
        MOV     DS,  AX
        LEA     SI,  MESS                 ;用DS: SI来指向字符串的首地址
        MOV     CX,  13D                  ;重复次数
        XOR     BX,  BX                   ;置求和的初值为0
        REP     LODSB
        …
```

指令“REP LODSB”能从字符串中取出每个字符, 但程序的其它指令无法处理每次出的数据, 指令的执行结果是: AL只保存最后一次所取出的字符‘h’。

所以, 为了实现本例的要求, 不能使用重复前缀指令, 而要把指令“REP LODSB”改成如下四条指令:

```
        XOR     AH,  AH                   ;为后面的累加作准备
again:  LODSB
        ADD     BX,  AX                   ;AL是被取出的字符, AH已被清0
        LOOP    again
```

第5章 微机CPU的指令系统

5. 条件重复前缀指令(Repeat String Conditionally)

条件重复前缀指令与前面的重复前缀指令功能相类似, 所不同的是: 其重复次数不仅由CX来决定, 而且还会由标志位ZF来决定。

A、相等重复前缀指令

REPE/REPZ SCAS/SCASB/SCASW/SCASD

REPE/REPZ CMPS/CMPSB/CMPSW/CMPSD

该重复前缀指令的执行步骤如下:

(1) 判断条件: CX ≠ 0 且 ZF = 1;

(2) 如果条件不成立, 则结束重复操作, 执行程序中的下一条指令;

(3) 否则, CX = CX-1(不影响有关标志位), 并执行其后的字符串操作指令, 在指令执行完后, 再转到步骤(1)。

第5章 微机CPU的指令系统

B、不等重复前缀指令

REPNE/REPNZ SCAS/SCASB/SCASW/SCASD

REPNE/REPNZ CMPS/CMPSB/CMPSW/CMPSD

该重复前缀指令的执行步骤如下:

(1) 判断条件: CX ≠ 0 且 ZF = 0;

(2) 如果条件不成立, 则结束重复操作, 执行程序中的下一条指令;

(3) 否则, CX = CX-1(不影响有关标志位), 并执行其后的字符串操作指令, 在指令执行完后, 再转到步骤(1)。

第5章 微机CPU的指令系统

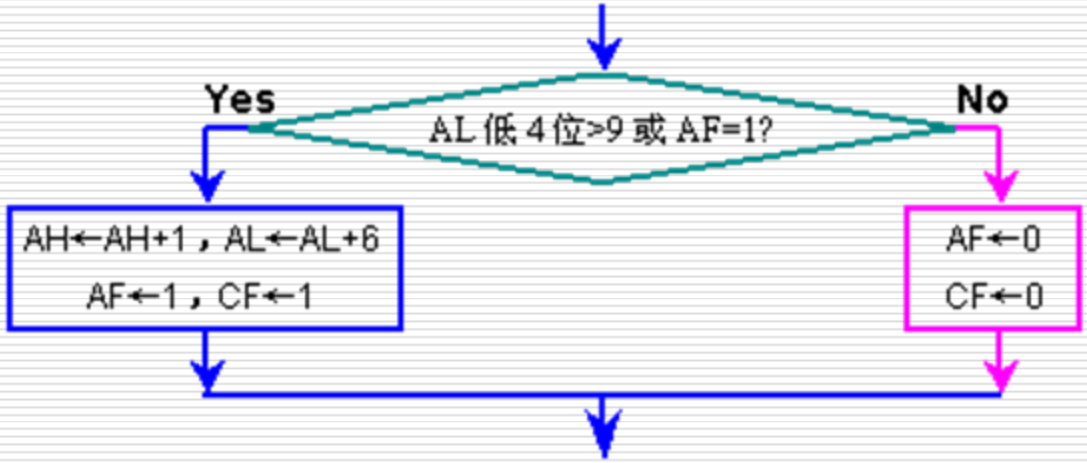
5.2.12 ASCII--BCD码运算调整指令

1. ASCII码加调整指令AAA(Ascii Adjust After Addition)

指令格式: AAA

该指令是用于调整AL的值。其调整规则如下:

- 如果AL的低四位大于9, 或AF为1, 那么, AH = AH+1, AL = AL+6, 并置AF和CF为1, 否则, 只置AF和CF为0;
- 清除AL的高四位。



例5.21: 编写一段程序，完成二个15位十进制数X和Y之和，并把计算结果存入X之中。假设数据X和Y都是以字符串形式表示的。

解:

```
        X  db  "456407983123186"          ;任意假设二个15位的大数
        Y  db  "326676709587211"
        …
        CLC
        MOV     SI,  14                   ;用变址寄存器SI来从字符串的后面向前访问
        MOV     CX,  15                   ;因为它们是二个15位十进制数
loop1:  MOV     AL,  X[SI]
        ADC     AL,  Y[SI]                ;把被加数加上
        AAA
        MOV     X[SI],  AL
        DEC     SI
        LOOP    loop1                     ;15位十进制数相加完毕
        LEA     BX,  X                    ;下面5条指令是把X中的数据变成字符
        MOV     CX,  15
loop2:  ADD     byte ptr [BX],  '0'
        INC     BX
LOOP    loop2
```

从上例来看，其实任意位的十进制数也都是可以的，只要改变CX的值即可。

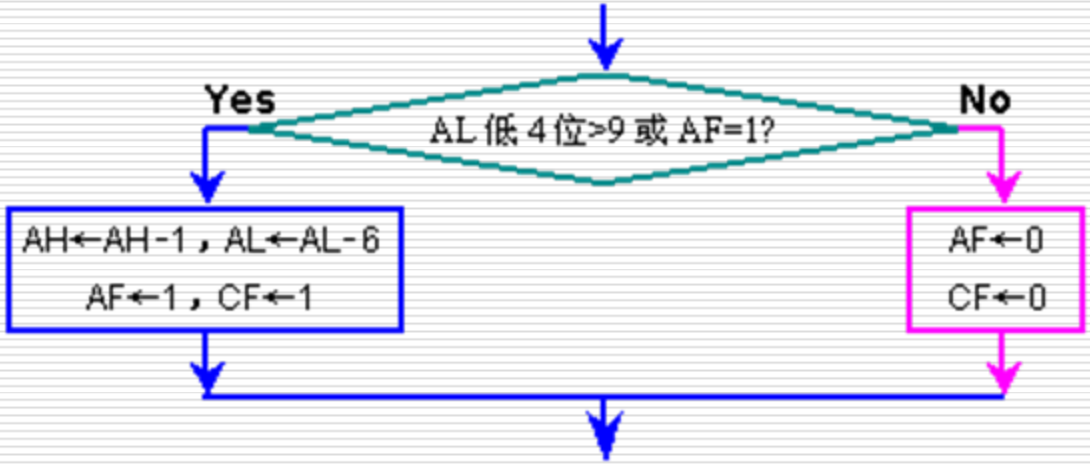
第5章 微机CPU的指令系统

2. ASCII码减调整指令AAS(Ascii Adjust After Subtraction)

指令格式: AAS

该指令是用于调整AL之值。其调整规则如下:

- 如果AL的低四位大于9, 或AF为1, 那么, AH = AH-1, AL = AL-6, 并置AF和CF为1, 否则, 只置AF和CF为0;
- 清除AL的高四位。



第5章 微机CPU的指令系统

3. ASCII码乘调整指令AAM(Ascii Adjust After Multiplication)

指令格式: AAM

该指令是用于调整寄存器AL之值, 该值是由二个单BCD码字节用无符号乘指令MUL所得的积。其调整规则如下:

AH ← AL/10(商), AL ← AL%10(余数)

例如:

```
MOV     AL,  9
MOV     BL,  8
MUL     BL              ;AL = 72D
AAM                     ;AH = 7, AL = 2
```

第5章 微机CPU的指令系统

4. ASCII码除调整指令AAD(Ascii Adjust After Division)

指令格式: AAD

该指令是在作除法前用于调整寄存器AH和AL之值, 它是把二个寄存器中单BCD码组成一个十进制数值, 为下面的除法作准备的。其调整规则如下:

AL ← AH*10+AL, AH ← 0

例如:

```
MOV     AX,  0502H
MOV     BL,  10D
AAD                     ;AH = 0, AL = 52D
DIV     BL              ;AH = 2(余数), AL = 5(商)
```

第5章 微机CPU的指令系统

5. 十进制数加调整指令DAA(Decimal Adjust After Addition)

指令格式: DAA

该指令是用于调整AL的值, 该值是由指令ADD或ADC运算二个压缩型BCD码所得到的结果。其调整规则如下:

- 如果AL的低四位大于9, 或AF为1, 那么, AL = AL+6, 并置AF为1;
- 如果AL的高四位大于9, 或CF为1, 那么, AL = AL+60H, 并置CF为1;
- 如果以上两点都不成立, 则, 清除标志位AF和CF。

经过调整后, AL的值仍然是一个压缩型BCD码, 也就是说, 二个压缩型BCD码相加, 并进行调整后, 得到的结果还是压缩型BCD码。

例如:

```
MOV     AL,  43H
MOV     BL,  29H
ADD     AL,  BL          ;AL = 6BH, 其不是压缩型的BCD码
DAA                      ;调整后, AL = 72H, 这是压缩型的BCD码
```

第5章 微机CPU的指令系统

中山大學

6. 十进制数减调整指令DAS(Decimal Adjust After Subtraction)

指令格式: DAS

该指令也是用于调整AL的值, 该值是由指令SUB或SBB运算二个压缩型BCD码所得到的结果。其调整规则如下:

- 如果AL的低四位大于9, 或AF为1, 那么, AL = AL-6, 并置AF为1;
- 如果AL的高四位大于9, 或CF为1, 那么, AL = AL-60H, 并置CF为1;
- 如果以上两点都不成立, 则, 清除标志位AF和CF。

经过调整后, AL的值仍然是一个压缩型BCD码, 也就是说, 二个压缩型BCD码相减, 并进行调整后, 得到的结果还是压缩型BCD码。

例如:

```
MOV     AL, 43H
MOV     BL, 29H
SUB     AL, BL          ;AL=1AH, 其不是压缩型的BCD码
DAS                     ;调整后, AL=14H, 这是压缩型的BCD码
```

第5章 微机CPU的指令系统

5.2.13 处理器指令

1. 空操作指令NOP(No Operation Instruction)

该指令没有的显式操作数, 主要起延迟下一条指令的执行。通常用执行指令XCHG AX, AX来代表它的执行。

指令格式: NOP

2. 等待指令WAIT(Put Processor in Wait State Instruction)

该指令使CPU处于等待状态, 直到协处理器(Coprocessor)完成运算, 并用一个重启信号唤醒CPU为止。该指令的执行不影响任何标志位。

指令格式: WAIT

第5章 微机CPU的指令系统

3. 暂停指令HLT(Enter Halt State Instruction)

在等待中断信号时，该指令使CPU处于暂停工作状态，CS: IP指向下一条待执行的指令。当产生了中断信号，CPU把CS和IP压栈，并转入中断处理程序。中断处理程序执行完后，中断返回指令IRET弹出IP和CS，并唤醒CPU执行下条指令。

指令格式: HLT

4. 封锁数据指令LOCK(Lock Bus Instruction)

该指令是一个前缀指令形式，在其后面跟一个具体的操作指令。LOCK指令可以保证是在其后指令执行过程中，禁止协处理器修改数据总线上的数据，起到独占总线的作用。

指令格式: LOCK INSTRUCTION

谢　谢

计算机科学系

2003年03月20日

下一章