



# 汇编语言程序设计

*Assembly Language Programming*

主讲：徐娟

计算机与信息学院 计算机系 分布智能与物联网  
研究所

E-mail: [xujuan@hfut.edu.cn](mailto:xujuan@hfut.edu.cn),

Mobile: 18055100485

- 数据传送指令
- 算术运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- CPU控制指令

# 什么是指令系统

- ❖ 计算机的指令系统就是指该计算机能够执行的全部指令的集合
- ❖ 每种计算机都有它支持的指令集合
- ❖ 16位8086指令系统是Intel 80x86系列微处理器指令系统的基础
- ❖ Intel 80x86系列微处理器指令系统：
  - 整数指令
  - 浮点指令
  - 多媒体指令

# 8086指令系统概述

- ❖ Intel 8086指令系统共有117条基本指令
- ❖ 可分成6个功能组
  - ① 数据传送类指令
  - ② 算术运算类指令
  - ③ 位操作类指令
  - ④ 串操作类指令
  - ⑤ 控制转移类指令
  - ⑥ 处理机控制类指令



# 学习指令的注意事项

- ❖ 指令的功能——该指令能够实现何种操作。通常指令助记符就是指令功能的英文单词或其缩写形式
- ❖ 指令支持的寻址方式——该指令中的操作数可以采用何种寻址方式
- ❖ 指令对标志的影响——该指令执行后是否对各个标志位有影响，以及如何影响
- ❖ 其他方面——该指令其他需要特别注意的地方，如指令执行时的约定设置、必须预置的参数、隐含使用的寄存器等

# 汇编语言指令格式

## ❖ 指令的一般格式

【标号：】 指令助记符 操作数1 操作数2 【；注释】

- 有些指令不需要操作数，通常的指令都有一个或两个操作数，个别指令有3个甚至4个操作数

● 标号表示该指令在主存中的逻辑地址

● 每个指令助记符就代表一种指令

● 目的和源操作数表示参与操作的对象

● 注释是对该指令或程序段功能的说明

## 2. 1 数据传送类指令

- ❖ 数据传送是计算机中最基本、最重要的一种操作
- ❖ 传送指令也是最常使用的一类指令
- ❖ 传送指令把数据从一个位置传送到另一个位置
- ❖ 除标志寄存器传送指令外，均不影响标志位
- ❖ 重点掌握
  - MOV XCHG
  - PUSH POP
  - LEA

# 传送指令MOV (move)

❖ 把一个字节或字的操作数从源地址传送至目的地址

## 1、立即数传送

mov al, 4 ; al  $\leftarrow$  04h, 字节传送

mov cx, 0ffh ; cx  $\leftarrow$  00ffh, 字传送

mov si, 200h ; si  $\leftarrow$  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

# 传送指令MOV (move)

## 2、寄存器传送

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

# 传送指令MOV (move)

## 3：存储器(内存)传送

```
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!

# 传送指令MOV (move)

## 4：段寄存器传送

mov [si], ds

{ mov ax, es ; ax ← es  
  mov ds, ax ; ds ← ax ← es

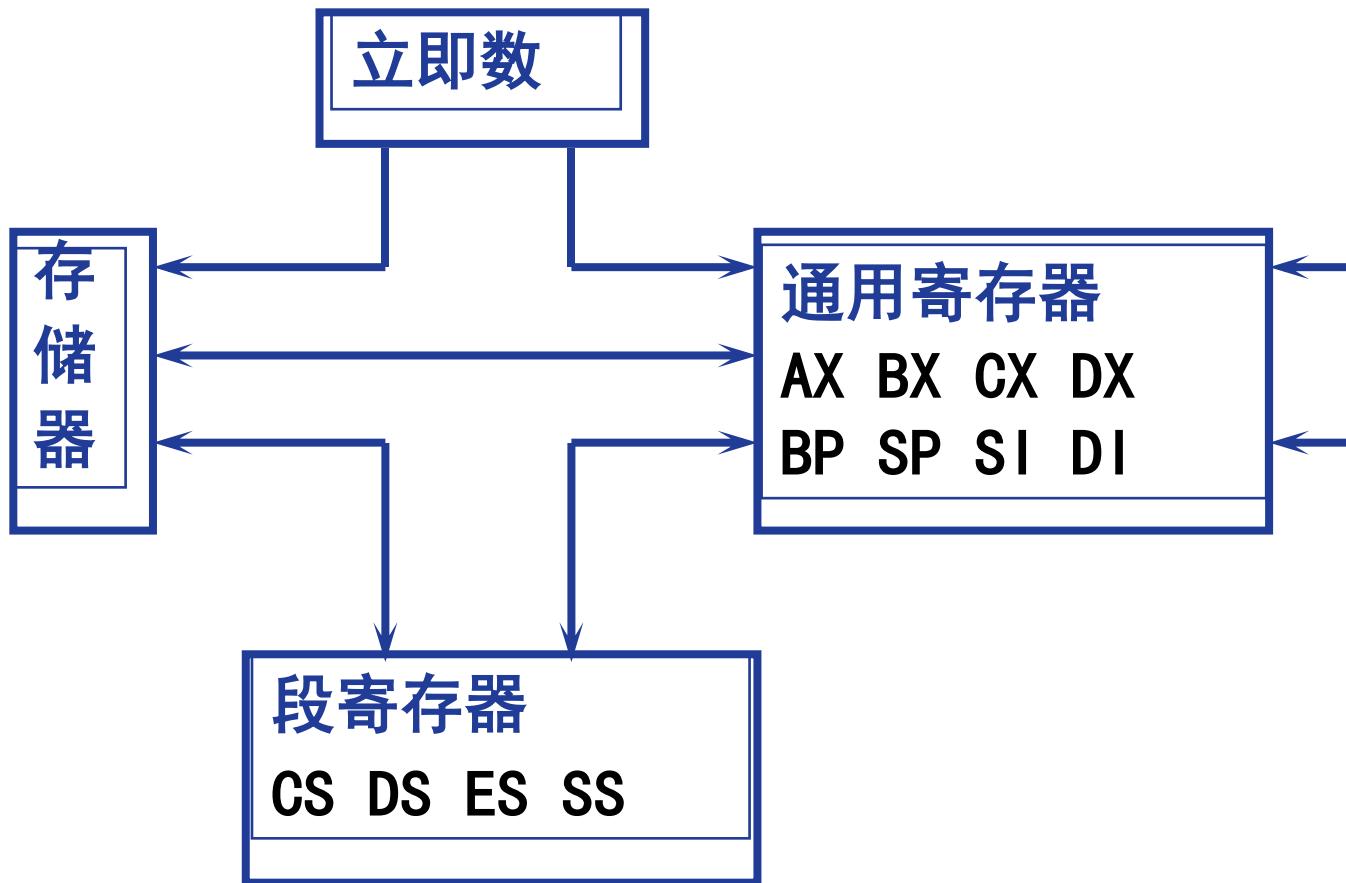


对段寄存器的操作有一些限制



不允许段寄存器之间的直接数据传送

# MOV指令传送功能



**MOV并非任意传送, 注意事项!**

# 两个操作数的类型要一致

- ❖ 绝大多数双操作数指令，除非特别说明，目的操作数与源操作数必须类型一致，否则为非法指令

MOV AL, 050AH ; 非法指令：050Ah为字，而AL为字节

- ❖ 寄存器有明确的字节或字类型，有寄存器参与的指令其操作数类型就是寄存器的类型
- ❖ 对于存储器单元与立即数同时作为操作数的情况，必须显式指明；byte ptr指示字节类型，word ptr指示字类型

mov byte ptr [si],0ah

# 两个操作数不能都是存储器

- ❖ 8086指令系统不允许两个操作数都是存储单元（除串操作指令），要实现这种传送，可通过寄存器间接实现

```
mov ax, buffer1 ; ax←buffer1 (将buffer1内容送ax)
```

```
mov buffer2, ax ; buffer2←ax
```

- ❖ buffer1和buffer2是两个字变量；实际表示直接寻址方式

# 要小心段寄存器的操作

- ❖ 不允许立即数传送给段寄存器
  - `MOV DS, 100H ; 非法指令：立即数不能传送段寄存器`
- ❖ 不允许直接改变CS值
  - `MOV CS, [SI] ; 不允许使用的指令`
- ❖ 不允许段寄存器之间的直接数据传送
  - `MOV DS, ES ; 非法指令：不允许段寄存器间传送`



## ❖ [习题2.2] 指出下列指令的错误

- ❖ (1) **mov cx, dl** 两操作数类型不匹配
- ❖ (2) **mov ip, ax** IP指令指针禁止用户访问
- ❖ (3) **mov es, 1234h** 立即数不允许传给段寄存器
- ❖ (4) **mov es, ds** 段寄存器之间不允许直接传送
- ❖ (5) **mov al, 300** 两操作数类型不匹配
- ❖ (6) **mov [sp], ax** 目的操作数应为[ SI ]
- ❖ (7) **mov ax, bx+di** 源操作数应为 [BX+DI]
- ❖ (8) **mov 20h, ah** 立即数不能作目的操作数

## 2. 交换指令XCHG (exchange)

- ❖ 把两个地方的数据进行互换

XCHG reg, reg/mem ; reg  $\leftrightarrow$  reg/mem

- ❖ 寄存器与寄存器之间对换数据

mov ax, 1234h ; ax=1234h

mov bx, 5678h ; bx=5678h

xchg ax, bx ; ax=5678h, bx=1234h

xchg ah, al ; ax=7856h

- ❖ 寄存器与存储器之间对换数据

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

- ❖ 不能在存储器与存储器之间对换数据

### 3. 换码指令XLAT (translate)

- ❖ 将BX指定的缓冲区中、AL指定的位移处的一个字节数据取出赋给AL

XLAT

; al $\leftarrow$ ds:[bx+al]

- 换码指令执行前：  
主存建立一个字节量表格，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

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)



❖ 通用数据传送指令

❖ 堆栈操作指令

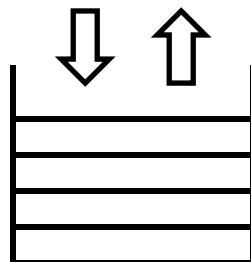
❖ 标志传送指令

❖ 地址传送指令

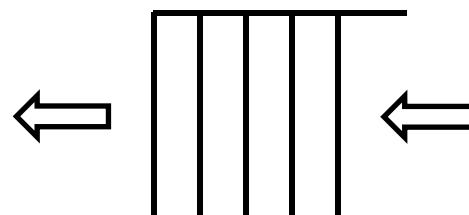
## 2.1.2 堆栈操作指令

- ❖ 堆栈：具有特殊访问方式的存储空间；特殊之处：按照后进先出(LIFO)的原则组织
- ❖ SS段寄存器记录其段地址；用堆栈指针寄存器SP指定栈顶（唯一出口）

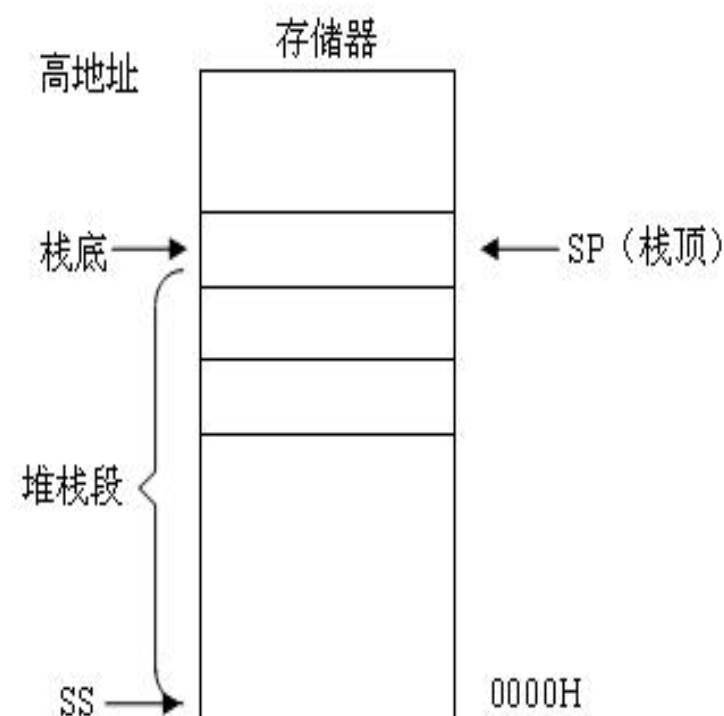
把一段内存当作栈来使用



堆栈:LIFO



队列:FIFO



# 堆栈的操作

- ❖ 堆栈只有两种基本操作：进栈 (PUSH) 和出栈 (POP)
- ❖ PUSH；进栈指令先使堆栈指针SP减2，然后把一个字操作数存入堆栈顶部

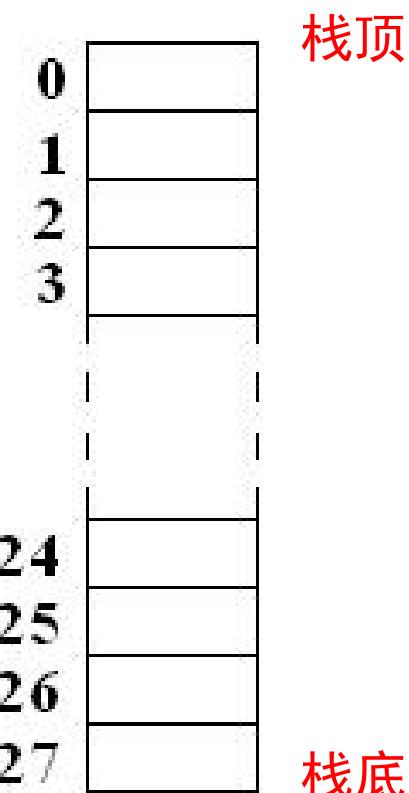
PUSH r16/m16/seg

- 1)  $SP \leftarrow SP - 2$
- 2)  $SS:[SP] \leftarrow r16/m16/seg$

- ❖ POP ；出栈指令把栈顶的一个字传送至指定的目的操作数，然后堆栈指针SP加2

POP r16/m16/seg

- 1)  $r16/m16/seg \leftarrow SS:[SP]$
- 2)  $SP \leftarrow SP + 2$



# 堆栈的操作

❖ 例:将10000H-1000FH这段内存当作栈来使用

mov ax,0123H

push ax

mov bx,2266H

push bx

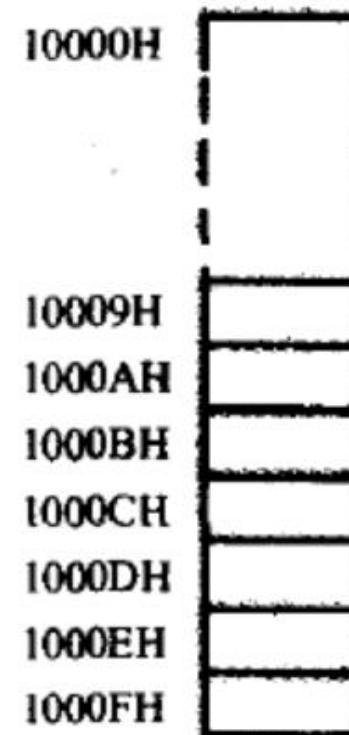
mov cx, 1122H

push cx

pop ax

pop bx

pop cx



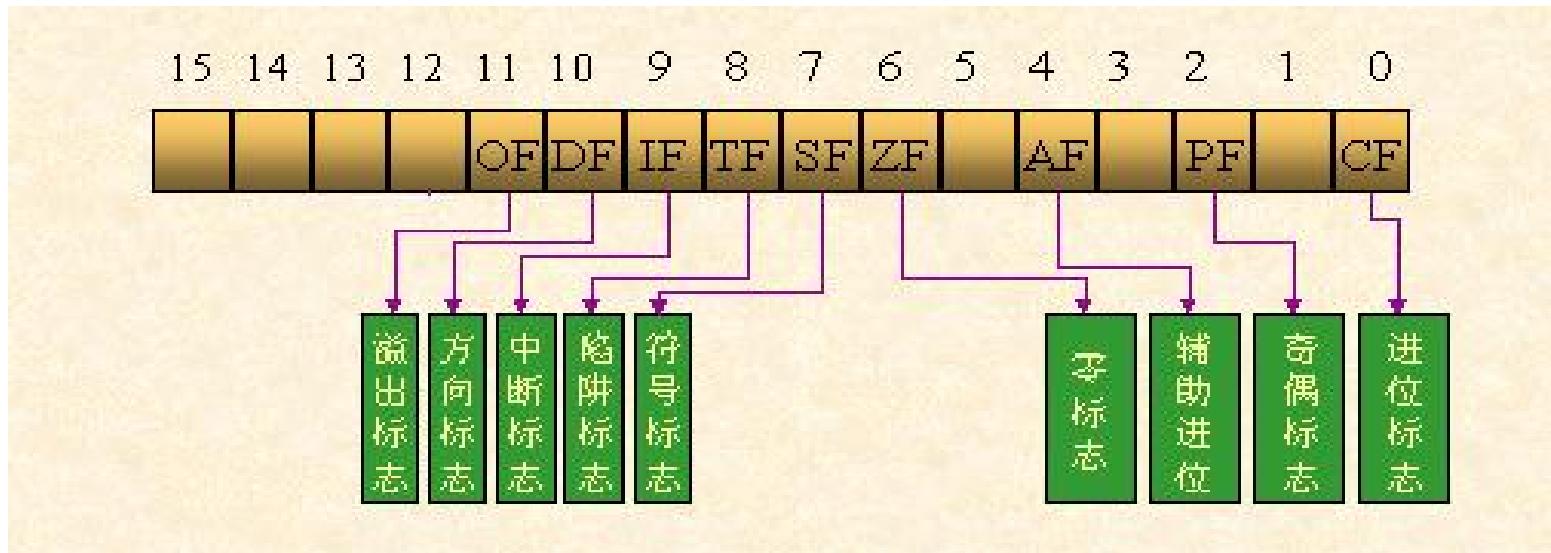
看动画：8086CPU的栈操作

# 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

## 2.1.3 标志传送指令

- ❖ 标志寄存器传送指令用来传送标志寄存器FLAGS的内容，方便进行对各个标志位的直接操作
- ❖ 有2对4条指令
  - 低8位传送：LAHF和SAHF
  - 16位传送：PUSHF和POPF



# 标志低字节进出AH指令

## LAHF

- ； AH←FLAGS的低字节
- ❖ 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

# 标志寄存器进出堆栈指令

## 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

# 标志寄存器进出堆栈指令

例：置位单步标志

pushf ; 保存全部标志到堆栈

pop ax ; 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

## 2.1.3 标志传送指令

- ❖ **标志位操作指令：**直接对CF、DF、IF标志进行复位或置位，常用于特定的情况(不影响其他标志)
- ❖ **对标志位进行设置的指令**
  - CLC STC CMC
  - CLD STD
  - CLI STI

## 2.1.3 标志传送指令

❖ 进位标志操作指令, 用于任意设置进位标志

CLC ; 复位进位标志:  $CF \leftarrow 0$

STC ; 置位进位标志:  $CF \leftarrow 1$

CMC ; 求反进位标志:  $CF \leftarrow \sim CF$

❖ 方向标志操作指令, 串操作指令中需要使用

CLD ; 复位方向标志:  $DF \leftarrow 0$

STD ; 置位方向标志:  $DF \leftarrow 1$

❖ 中断标志操作指令, 在编写中断服务程序时, 需要控制可屏蔽中断的允许和禁止

CLI ; 复位中断标志:  $IF \leftarrow 0$

STI ; 置位中断标志:  $IF \leftarrow 1$

## 2.1.4 地址传送指令

- ❖ 地址传送指令将存储器单元的逻辑地址送至指定的寄存器
  - 有效地址传送指令 LEA
  - 指针传送指令 LDS和LES
- ❖ 注意不是获取存储器单元的内容

# 有效地址传送指令LEA (load EA)

- ❖ 将存储器操作数的有效地址传送至指定的16位寄存器中

**LEA r16,mem ; r16←mem的有效地址EA**

**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

# 指针传送指令

**LDS r16,mem**

;  $r16 \leftarrow mem,$   
;  $DS \leftarrow mem+2$

❖ LDS指令将主存中mem指定的字送至r16，并将mem的下一字送**DS寄存器**

**LES r16,mem**

;  $r16 \leftarrow mem,$   
;  $ES \leftarrow mem+2$

❖ LES指令将主存中mem指定的字送至r16，并将mem的下一字送**ES寄存器**

# 指针传送指令

## 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**

D<sub>7</sub> D<sub>0</sub>

00H
01H
50H
14H

3060H

3061H

3062H

3063H

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

## 2. 算术运算指令

- ❖ 加法指令
- ❖ 减法指令
- ❖ 乘法指令
- ❖ 除法指令
- ❖ 十进制/BCD码调整指令

# 加法指令ADD

- ❖ ADD指令将源与目的操作数相加，结果送到目的操作数
- ❖ ADD指令按状态标志的定义相应设置

**ADD reg,imm/reg/mem** ;  $\text{reg} \leftarrow \text{reg} + \text{imm/reg/mem}$

**ADD mem,imm/reg** ;  $\text{mem} \leftarrow \text{mem} + \text{imm/reg}$

**mov al,0fbh**

**add al,07h**

**mov word ptr [200h],4652h**

**mov bx,1feh**

**add al,bl**

**add word ptr [bx+2],0f0f0h**

？ 调试看看各标志位的情况

# 带进位加法指令ADC

- ◆ ADC指令将源与目的操作数相加，再加上进位CF标志，结果送到目的操作数
- ◆ ADC指令按状态标志的定义相应设置
- ◆ ADC指令主要与ADD配合，实现多字节加法运算

**ADC reg,imm/reg/mem ; reg $\leftarrow$ reg+imm/reg/mem+CF**

**ADC mem,imm/reg ; mem $\leftarrow$ mem+imm/reg+CF**

**mov ax,4652h ; ax=4652h**

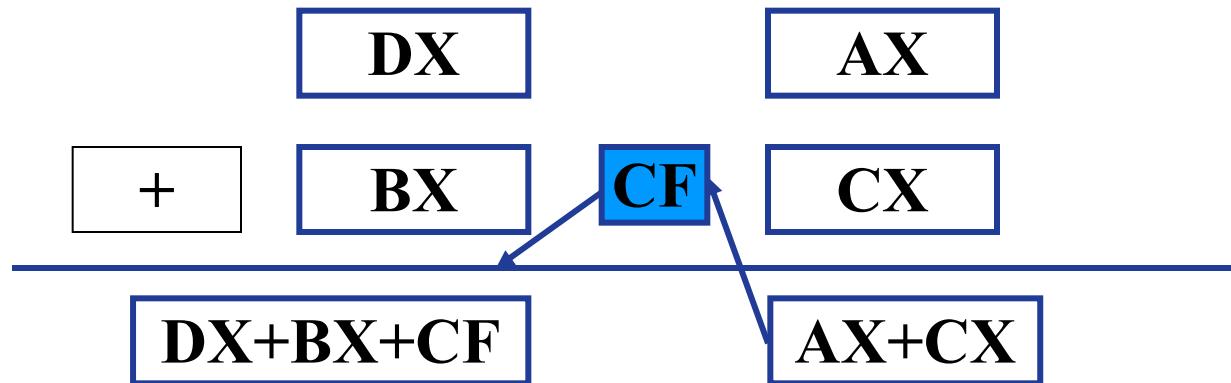
**add ax,0f0f0h ; ax=3742h, CF=1**

**mov dx,0234h ;**

**adc dx,0f0f0h ;**

# 多字节数相加

- ❖ **DX= 0002H AX= 0F365H**
- ❖ **BX= 0005H CX= 0E024H**
- ❖ (1) **ADD AX, CX**
  - 执行后, AX= 0D389H CF=1
- ❖ (2) **ADC DX, BX**
  - 执行后, DX= 0008H CF=0



# 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 $\leftarrow$ reg/mem + 1**

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

# 减法指令SUB (subtract)

- ❖ SUB指令将目的操作数减去源操作数，结果送到目的操作数
- ❖ SUB指令按照定义相应设置状态标志

**SUB reg,imm/reg/mem** ;  $\text{reg} \leftarrow \text{reg} - \text{imm/reg/mem}$

**SUB mem,imm/reg** ;  $\text{mem} \leftarrow \text{mem} - \text{imm/reg}$

**mov al,0fbh**

**sub al,07h**

**mov word ptr [200h],4652h**

**mov bx,1feh**

**sub al,bl**

**sub word ptr [bx+2],0f0f0h**

# 带借位减法指令SBB

- ❖ SBB指令将目的操作数减去源操作数，再减去借位CF（进位），结果送到目的操作数。
- ❖ SBB指令按照定义相应设置状态标志
- ❖ SBB指令主要与SUB配合，实现多精度减法运算

**SBB reg,imm/reg/mem ; reg $\leftarrow$ reg-imm/reg/mem-CF**

**SBB mem,imm/reg ; mem $\leftarrow$ mem-imm/reg-CF**

**mov ax,4652h ; ax=4652h**

**sub ax,0f0f0h ; ax=5562h, CF=1**

**mov dx,0234h ; dx=0234h**

**sbb dx,0f0f0h ; dx=1143h, CF=1**

# 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
- ❖ 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

**DEC reg/mem ; reg/mem $\leftarrow$ reg/mem-1**

**dec cx**

**dec word ptr [si]**

# 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，**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**

**mov ax,0ff64h**

**neg al ; ax=ff9ch, OF=0、SF=1、ZF=0、PF=1、CF=1**

**sub al,9dh ; ax=fffffh, 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**

# 比较指令CMP (compare)

- ◆ CMP指令执行的功能与SUB指令，但结果不回送目的操作数
- ◆ CMP指令将目的操作数减去源操作数，执行CMP之后，可以根据标志位判断两个数是否相等、大小关系等

**CMP reg, imm/reg/mem ; reg—imm/reg/mem**

**CMP mem, imm/reg ; mem—imm/reg**

**cmp al,100 ; al-100**

**jb below ; al<100, 跳转到below执行**

**sub al,100 ; al $\geq$ 100, al $\leftarrow$ al-100**

**inc ah ; ah $\leftarrow$ ah+1**

**比较AL与100**

**below: ...**

# 习题

例：x、y、z均为双精度无符号数，分别存放在地址为X, X+2; Y, Y+2; Z, Z+2的存储单元中，用指令序列实现  $w \leftarrow x+y+24-z$ ，并用W, W+2单元存放w。

## 2.2.4 乘法指令 (Multiplication )

**MUL r8/m8**

; 无符号字节乘法

; AX←AL×r8/m8

**MUL r16/m16**

; 无符号字乘法

; DX.AX←AX×r16/m16

**IMUL r8/m8**

; 有符号字节乘法

; AX←AL×r8/m8

**IMUL r16/m16**

; 有符号字乘法

; DX.AX←AX×r16/m16

❖ 乘法指令的源操作数显式给出，**隐含使用另一个操作数AX和DX**

- 字节量相乘： AL与r8/m8相乘，得到16位的结果，**存入AX**
- 字量相乘： AX与r16/m16相乘，得到32位的结果，**其高字存入DX，低字存入AX**

❖ 乘法指令利用**OF**和**CF**判断乘积的高一半是否具有有效数值

# 乘法指令对标志的影响

## ❖ 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

## 2.2.4 乘法指令

例:  $(AX) = 16A5H, (BX) = 0611H$

(1) **IMUL BL** ;  $(AX) \leftarrow (AL) * (BL)$

;  $A5 * 11 \Rightarrow 5B * 11 = 060B \Rightarrow F9F5$   
;  $(AX) = 0F9F5H$

(2) **MUL BX** ;  $(DX, AX) \leftarrow (AX) * (BX)$

;  $16A5 * 0611 = 0089\ 5EF5$   
;  $(DX) = 0089H\ (AX) = 5EF5H$

## 2.2.5 除法指令 (division)

DIV r8/m8 ; 无符号字节除法:

AL $\leftarrow$ AX  $\div$  r8/m8的商, Ah $\leftarrow$ AX  $\div$  r8/m8的余数

DIV r16/m16 ; 无符号字除法:

; AX $\leftarrow$ DX. AX  $\div$  r16/m16的商, DX $\leftarrow$ DX. AX  $\div$  r16/m16的余数

IDIV r8/m8 ; 有符号字节除法:

AL $\leftarrow$ AX  $\div$  r8/m8的商, Ah $\leftarrow$ AX  $\div$  r8/m8的余数

IDIV r16/m16 ; 有符号字除法:

; AX $\leftarrow$ DX. AX  $\div$  r16/m16的商, DX $\leftarrow$ DX. AX  $\div$  r16/m16的余数

# 除法指令的功能

- ❖ 除法指令分**无符号和有符号**除法指令
- ❖ 除法指令的除数显式给出，隐含使用另一个操作数**AX和DX作为被除数**
  - 字节量除法： AX除以r8/m8， 8位商存入AL， 8位余数存入AH
  - 字量除法： DX. AX除以r16/m16， 16位商存入AX， 16位余数存入DX
- ❖ 除法指令对标志没有定义
- ❖ 除法指令会产生结果溢出

# 除法错中断

- ❖ 当被除数远大于除数时，所得的商就有可能超出它所能表达的范围。如果存放商的寄存器AL/AX不能表达，便产生溢出，8086CPU中就产生编号为0的内部中断——除法错中断

DIV指令：除数为0，在字节除时商超过8位，或者在字除时商超过16位

IDIV指令：除数为0，在字节除时商不在-128～127范围内  
或者在字除时商不在-32768～32767范围内

## 例：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

## 2.2.6 符号扩展指令

- ❖ 符号扩展：用一个操作数的符号位（即最高位）形成另一个操作数，**后一个操作数的各位是全0（正数）或全1（负数）**  
符号扩展不改变数据大小
  - 对于数据64H（表示数据100），其最高位D7为0，符号扩展后高8位都是0，成为0064H（仍表示数据100）
  - 对于数据FF00H，其最高位D15为1，符号扩展后高16位都是1，成为FFFFFF00H（仍表示有符号数-256）



## 2.2.6 符号扩展指令

➤ 符号扩展指令常用于获得倍长的数据

**CBW (Byte-Word)** ; AL的符号扩展至AH

; 如AL的最高有效位是0，则AH=00

; AL的最高有效位为1，则AH=FFH。AL不变

**CWD (Word-Dword)** ; AX的符号扩展至DX

; 如AX的最高有效位是0，则DX=00

; AX的最高有效位为1，则DX=FFFFH。AX不变

**mov al,80h**

**cbw**

**add al,255**

**cbw**

## 例：AX ÷ BX

cwd

; DX.AX←AX

idiv bx

; AX←DX.AX ÷ BX

- 对有符号数除法：可以利用符号扩展指令得到倍长于除数的被除数
- 对无符号数除法：采用直接使高8位或高16位清0，获得倍长的被除数。这就是零位扩展

## 2.2.7 十进制调整指令

- 十进制数调整指令对二进制运算的结果进行十进制调整，以得到十进制的运算结果
- 分成压缩BCD码和非压缩BCD码调整

□ 压缩BCD码就是通常的8421码；它用**4个二进制位**表示一个十进制位，一个字节可以表示两个十进制位，即00~99

□ 非压缩BCD码用**8个二进制位**表示一个十进制位，只用低4个二进制位表示一个十进制位0~9，高4位任意，通常默认为0

真值	8d	64d
二进制编码	08H	40H
压缩BCD码	08H	64H
非压缩BCD码	08H	0604H

# 十进制/BCD调整指令



## ❖ 压缩BCD码调整指令： DAA/DAS

- “DAA” 和 “DAS” 指令必须紧跟在相应的加法指令和减法指令后面
- 调整的数据只能是AL中的内容
- DAA和DAS指令对OF标志无定义，按结果影响其他标志，例如CF反映压缩BCD码相加或减的进位或借位状态

## ❖ 非压缩BCD码调整指令

- AAA (ASCII Adjust for Addition)
- AAS (ASCII Adjust for Subtraction)
- AAM (ASCII Adjust for Multiplication)
- AAD (ASCII Adjust for Division)

# 压缩BCD码加、减调整指令

## 例：压缩BCD加法

```
mov al,68h ; al=68h, 压缩BCD码表示真值68  
mov bl,28h ; bl=28h, 压缩BCD码表示真值28  
add al,bl ; 二进制加法: al=68h+28h=90h  
daa ; 十进制调整: al=96h  
; 实现压缩BCD码加法: 68+28=96
```

## 例：压缩BCD减法

```
mov al,68h ; al=68h, 压缩BCD码表示真值68  
mov bl,28h ; bl=28h, 压缩BCD码表示真值28  
sub al,bl ; 二进制减法: al=68h-28h=40h  
das ; 十进制调整: al=40h  
; 实现压缩BCD码减法: 68-28=40
```

## 例：压缩BCD减法

**mov ax,1234h**

**mov bx,4612h**

**sub al,bl**

**das**

**xchg al,ah**

**sbb al,bh**

**Das**

**xchg al,ah**

## 习题2.9

- ◆ 设X、Y、Z、V均为16位带符号数，分别存放在X、Y、Z、V存储单元中，阅读如下程序段，得出它的运算公式，并说明运算结果存于何处？

mov ax,X

imul Y

; DX.AX=X×Y

mov cx,ax

mov bx,dx

; BX.CX=X×Y

mov ax,Z

cwd

; DX.AX=Z

add cx,ax

adc bx,dx

; BX.CX=X×Y+Z

65
F3
02
00
24
E0
05
00
⋮

V  
Z  
Y  
X

## 习题2.9

- ◆ 设X、Y、Z、V均为16位带符号数，分别存放在X、Y、Z、V存储单元中，阅读如下程序段，得出它的运算公式，并说明运算结果存于何处？

sub cx,540

sbb bx,0 ; BX.CX = X × Y + Z - 540

mov ax,V

cwd ; DX.AX = V

sub ax,cx

sbb dx,bx ; DX.AX = V - (X × Y + Z - 540)

idiv X

; DX.AX = (V - (X × Y + Z - 540)) ÷ X

## DOSBox Status Window

```
DOSBox 0.74, Cpu speed: 3000 cycles, Fra...
Welcome to DOSBox v0.74
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c: d:\asm
Drive C is mounted as local directory d:\asm\
Z:\>c:
C:\>debug
```

s\xujuan\AppData\Local\DOsBox\dosbox-0.74.conf

谷歌拼音输入法 2 半 :



在正确理解每条指令的功能  
基础上，可以阅读和编写有  
实际意义的程序段



多多阅读程序段

