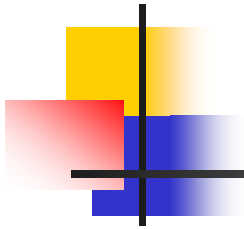
5 80x86的指令系统

- 数据传送指令
- 算术指令
- 逻辑指令
- 串处理指令
- 控制转移指令
- 处理机控制指令



重点关注：

- 指令的汇编格式
- 指令的基本功能
- 指令支持的寻址方式
- 指令的执行对标志位的影响
- 指令的特殊要求
- 对寻址方式或寄存器使用的限制和隐含使用的情况

§3.3.1 数据传送指令

1. 通用数据传送指令

- MOV DST，SRC
- PUSH SRC
- POP DST
- XCHG OPR1，OPR2

2. 累加器专用传送指令

- IN AX，PORT
- OUT PORT，AX
- XLAT

3. 地址传送指令

- LEA REG，SRC
- LDS REG，SRC
- LES REG，SRC

4. 标志寄存器传送指令

- POPF，PUSHF
- LAHF，SAHF

通用数据传送指令

传送指令：　MOV　DST, SRC

执行操作：　(DST) ← (SRC)

注意:

* DST不能是CS，也不能是立即数
* 不影响标志位
* DST,SCR不能同时为内存单元数据
* DST、SRC不同时为段寄存器　×　MOV　DS, ES
* 立即数不能直接送段寄存器　×　MOV　DS, 2000H

对段寄存器的赋值只能通过AX来完成（见例子3.20）

mov指令的常见几种形式

- **mov 寄存器，数据　　(mov ax, 8)**
- **mov 寄存器，寄存器　　(mov ax, bx)**
- **mov 寄存器，内存单元　　(mov ax, [0])**
- **mov 内存单元，寄存器　　(mov [0], ax)**
- **mov 段寄存器，寄存器　　(mov ds, ax)**

验证（Debug）

- mov 段寄存器，寄存器

  ➔ mov 寄存器，段寄存器

```
C:\>debug
-a
1389:0100 mov ax,ds
1389:0102
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1389  ES=1389  SS=1389  CS=1389  IP=0100   NV UP EI PL NZ NA PO NC
1389:0100 8CD8          MOV     AX,DS
-t

AX=1389  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1389  ES=1389  SS=1389  CS=1389  IP=0102   NV UP EI PL NZ NA PO NC
1389:0102 0000          ADD     [BX+SI],AL                      DS:0000=CD
-_
```

验证

mov 内存单元，寄存器

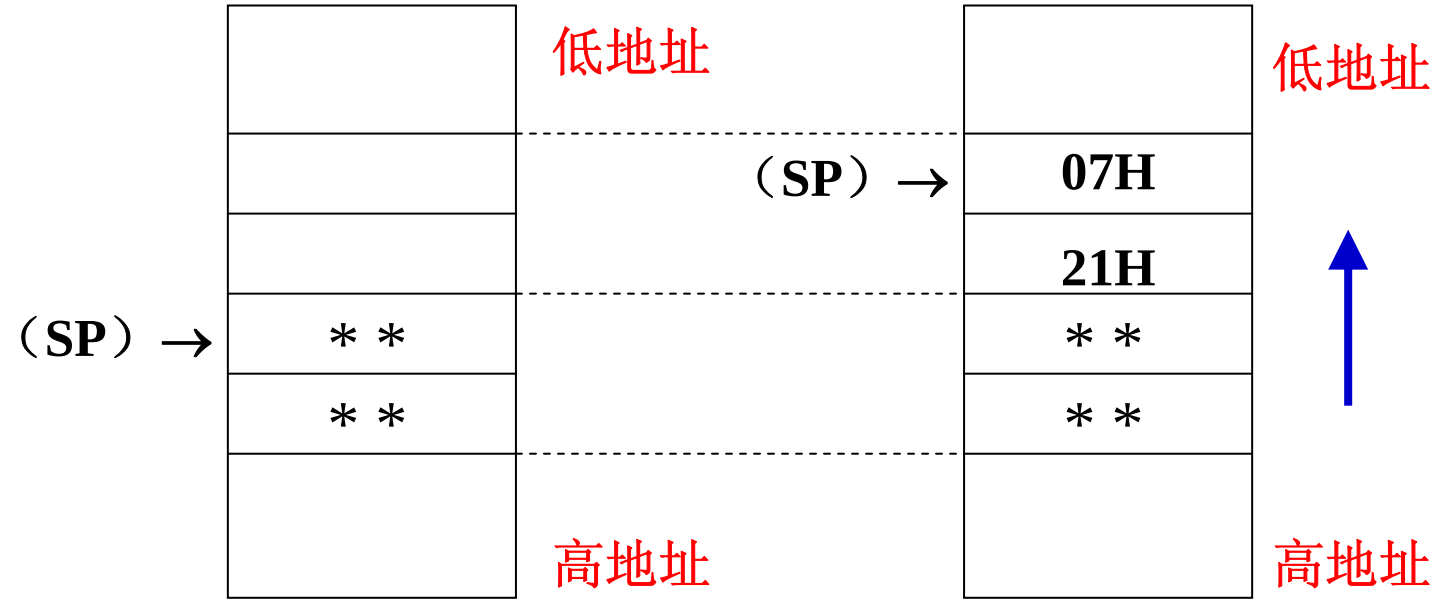
mov ax,1000h

mov ds,ax

mov [0],cs

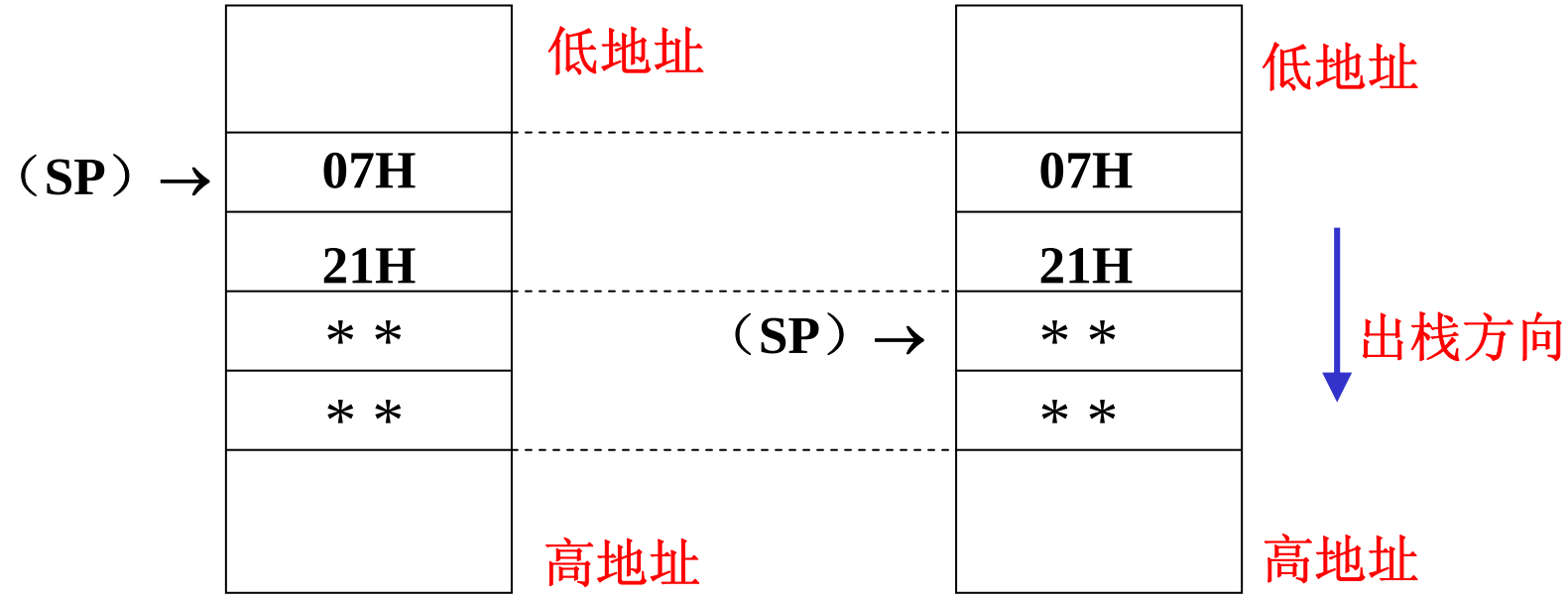
```
C:\>debug
-a
1389:0100 mov ax,1000
1389:0103 mov ds,ax
1389:0105 mov [0],cs
1389:0109
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1389  ES=1389  SS=1389  CS=1389  IP=0100   NV UP EI PL NZ NA PO NC
1389:0100 B80010        MOV     AX,1000
-t

AX=1000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1389  ES=1389  SS=1389  CS=1389  IP=0103   NV UP EI PL NZ NA PO NC
1389:0103 8ED8          MOV     DS,AX
-t

AX=1000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1000  ES=1389  SS=1389  CS=1389  IP=0105   NV UP EI PL NZ NA PO NC
1389:0105 8C0E0000      MOV     [0000],CS                       DS:0000=E8C0
-t

AX=1000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=1000  ES=1389  SS=1389  CS=1389  IP=0109   NV UP EI PL NZ NA PO NC
1389:0109 0000          ADD     [BX+SI],AL                      DS:0000=89
-d 1000:0
1000:0000  89 13 84 00 B0 2C AA F6-06 A6 56 FF 75 1E E8 F3    .....,....V.u...
1000:0010  FE EB 28 57 BF 9D 56 E8-12 00 5F E8 0E 00 B0 3A    ..(W..V..._.....:
1000:0020  AA BE 9D 56 B9 04 00 AC-AA E2 FC C3 E8 D5 FE 8A    ...V............
1000:0030  D0 E8 D0 FE 8A F0 E8 02-00 8A C2 8A E0 D0 E8 D0    ................
1000:0040  E8 D0 E8 D0 E8 E8 02 00-8A C4 24 0F 04 90 27 14    ..........$...'.
1000:0050  40 27 AA C3 E8 AD FE 3C-0A 75 E0 C3 BB 67 48 E8    @'.....<.u...gH.
1000:0060  E1 03 E8 AD 03 B0 2C AA-E8 99 FE 98 8B D0 8A E0    ......,.........
1000:0070  B0 2B 0A E4 79 04 B0 2D-F6 DC AA 8A C4 EB BC E8    .+..y..-........
-
```

例：　假设 (AX) = 2107 H，执行 PUSH AX



PUSH AX 执行前　　PUSH AX 执行后

例： POP BX



POP BX 执行前

POP BX 执行后

(BX) = 2107H

交换指令： XCHG

交换指令： **XCHG OPR1, OPR2**
执行操作： **(OPR1) ↔ (OPR2)**

注意：

* 不影响标志位
* 不允许使用段寄存器

例：**XCHG BX, [BP+SI]**

XCHG AL, BH

请分析：第52页 例3.34

§3.3.1 数据传送指令—地址传送指令

- 地址传送指令

有效地址送寄存器指令：　　LEA REG, SRC

执行操作：　　(REG) ← SRC

指针送寄存器和DS指令：　　LDS REG, SRC

执行操作：　　(REG) ← (SRC)

(DS) ← (SRC+2)

4个相继字节 → 寄存器（通常是SI）、DS

指针送寄存器和ES指令：　　LES REG, SRC

执行操作：　　(REG) ← (SRC)

(ES) ← (SRC+2)

4个相继字节 → 寄存器（通常是DI）、ES

注意：* 不影响标志位

* REG不能是段寄存器

* SRC必须为存储器寻址方式

§3.3.1 地址传送指令—LEA

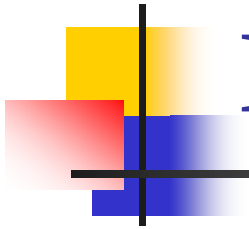
- 地址传送指令

  有效地址送寄存器指令：　　**LEA REG, SRC**

  执行操作：　　**(REG) ← SRC**

将偏移地址送指定的寄存器

注意：
* 不影响标志位
* REG不能是段寄存器
* SRC必须为存储器寻址方式



LEA

- 16位送16位
- 32位送16位，取低16位
- 16位送32位，零扩展
- 32位送32位

§3.3.1 地址传送指令—LDS

- 指针送寄存器和DS指令：　LDS REG, SRC
  执行操作：　(REG) ← (SRC)
  (DS) ← (SRC+2)

4个相继字节 → 寄存器（通常是SI）、DS

- 例：LDS SI, [10H]
- LDS DI,[BX]

注意：* 不影响标志位
* REG不能是段寄存器
* SRC必须为存储器寻址方式

LDS示例

- 利用**debug**查看的内存情况如下：

  1000:10F0 78 56 34 12

  执行下列的指令序列后，

  MOV AX, 1000

  MOV DS, AX

  LDS BX, [10F0]

- **DS =** 1234H　　**BX =** 5678H

§3.3.1 地址传送指令—LES

- 指针送寄存器和ES指令： LES REG, SRC
  执行操作： (REG) ← (SRC)
  (ES) ← (SRC+2)

  4个相继字节 → 寄存器（通常是DI）、ES

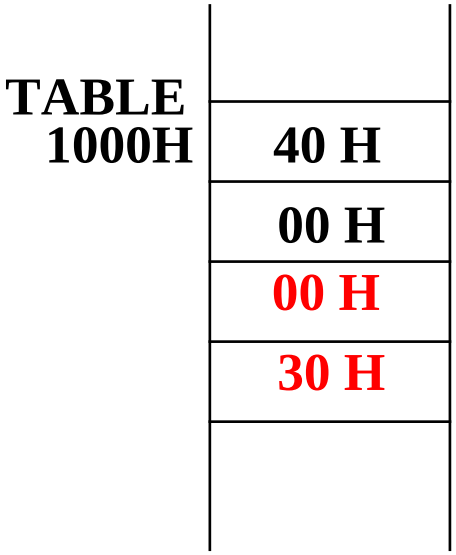
- 例：LES DI, [BX]

注意：
* 不影响标志位
* REG不能是段寄存器
* SRC必须为存储器寻址方式

- (BX)=0400H, (SI)=003CH
- LEA BX, [BX+SI+0F62H]
- MOV AX, [BX+SI+0F62H]
- AX = 0040H
- BX = 139EH

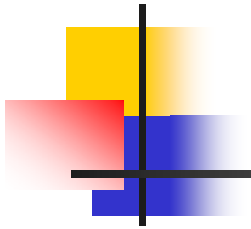
内存分配情况

内存地址	内存单元
2000:139E	40 H
2000:139F	00 H
2000:1340	15 H
2000:1341	30 H
2000:1342	2CH
⋮	




```
MOV  BX, TABLE              ; (BX)= 0040H
MOV  BX, OFFSET TABLE       ; (BX)= 1000H
LEA  BX, TABLE              ; (BX)= 1000H
LDS  BX, TABLE              ; (BX)= 0040H
                            ; (DS)= 3000H
LES  BX, TABLE              ; (BX)= 0040H
                            ; (ES)= 3000H
```

说明：OFFSET 功能----将其后随符号的偏移地址送指定寄存器。（参见：48页例3.22）



§ 3.3.2 算术指令

- 加法指令
- 减法指令
- 乘法指令
- 除法指令
- 十进制调整指令

§ 加法指令

- 加法指令

加法指令：　**ADD DST, SRC**
执行操作：　**(DST) ← (SRC) + (DST)**

带进位加法指令：　**ADC DST, SRC**
执行操作：　**(DST) ← (SRC) + (DST) + CF**

加1指令：　**INC OPR**
执行操作：　**(OPR) ← (OPR) + 1**

注意：
除INC指令不影响CF标志外，均对条件标志位 有影响。

加法指令对条件标志位（CF/OF/ZF/SF）的影响：

$$SF=\begin{cases}1 & \text{结果为负}\\ 0 & \text{否则}\end{cases}\qquad ZF=\begin{cases}1 & \text{结果为0}\\ 0 & \text{否则}\end{cases}$$

$$CF=\begin{cases}1 & \text{和的最高有效位有向高位的进位}\\ 0 & \text{否则}\end{cases}$$

$$OF=\begin{cases}1 & \text{两个操作数符号相同，而结果符号与之相反}\\ 0 & \text{否则}\end{cases}$$

CF位表示无符号数相加的溢出。

OF位表示带符号数相加的溢出。

OF的判定：
双符号位补码参与运算结果的符号位相同则OF为0，否则为1。
（01上溢，10下溢）

n=8bit 带符号数(-128~127) 无符号数(0~255)

带符号数和无符号数都不溢出

00 0 0 0 0 1 0 0
+ 00 0 0 0 1 0 1 1
00 0 0 0 1 1 1 1

带：(+4)+(+11)=+15 OF=0
无：4+11=15 CF=0

无符号数溢出

00 0 0 0 0 1 1 1
+ 11 1 1 1 1 0 1 1
00 0 0 0 0 0 1 0

带：(+7)+(-5)=+2 OF=0
无：7+251=2 CF=1

带符号数和无符号数都溢出

11 0 0 0 0 1 1 1
+ 11 1 1 1 0 1 0 1
10 1 1 1 1 1 0 0

带：(-121)+(-11)=+124 OF=1
无：135+245=124 CF=1

带符号数溢出

00 0 0 0 1 0 0 1
+ 00 1 1 1 1 1 0 0
01 0 0 0 0 1 0 1

带:(+9)+(+124)=-123 OF=1
无:9+124=133 CF=0

例：双精度数的加法

设双精度数α：（DX）= 0002H （AX）= 0F365H

双精度数β：（BX）= 0005H （CX）= 0E024H

求α＋β？

指令序列 (1) ADD AX, CX

(2) ADC DX, BX

(1) 执行后，（AX）= 0D389H CF=1 OF=0

SF=1 ZF=0

(2) 执行后，（DX）= 0008H CF=0 OF=0

SF=0 ZF=0

带符号的双精度数的溢出，应该根据ADC指令的OF位来判别。而判别低位加法ADD指令的溢出OF是无意义的。

注：考虑内存中的两个双精度数mem1,mem2相加

加1指令INC（单操作数指令）

格式：**INC reg/mem**

功能：类似于C语言中的++操作：对指定的操作数加1

例：
```
INC AL
INC SI
INC BYTE PTR[BX+4]
```

注：本指令不影响CF标志。

练习

- 任务：编程计算2^3。
- 分析：

汇编语言用什么指令来实现？

- 2^3 =2×2×2，若设(ax)=2，可计算：(ax)=(ax)×2×2，最后(ax)中为2^3的值，N×2可用N+N 实现。
- 请写出程序代码

任务：编程计算2^3

```
mov  ax, 2        ; 2 →ax
add  ax, ax       ;ax+ax →ax
add  ax, ax       ;ax+ax →ax
```

思考：写几条指令，累加数据段中的前3个字型数据，和存入AX

- 注意：一个字型数据占两个单元，所以偏移地址是0、2、4。

内存分配情况

内存地址	内存单元
123B:0000	40 H
123B:0001	00 H
123B:0002	15 H
123B:0003	30 H
123B:0004	2CH
123B:0005	21H
123B:0006	5CH
⋮	

思考：写几条指令，累加数据段中的前3个字型数据

```
MOV  AX, 123BH   ;将123BH送入DS，作为数据段的段地址
MOV  DS,  AX
MOV AX,  0       ;用AX存放累加结果，先清0
ADD  AX, [0]     ;将数据段第一个字（偏移地址为0），加入AX中
ADD  AX, [2]     ;将数据段第二个字（偏移地址为2），加入AX中
ADD  AX, [4]     ;将数据段第三个字（偏移地址为4），加入AX中
```

- 注意：一个字型数据占两个单元，所以偏移地址是0、2、4。

内存分配情况

内存地址	内存单元
123B:0000	40 H
123B:0001	00 H
123B:0002	15 H
123B:0003	30 H
123B:0004	2CH
123B:0005	21H
123B:0006	5CH
⋮	

§ 减法指令

- 减法指令
  减法指令：　　SUB DST, SRC
  执行操作：　　(DST) ← (DST) - (SRC)
- 带借位减法指令：　SBB DST, SRC
  执行操作：　　(DST) ← (DST) - (SRC) - CF
- 减1指令：　　DEC OPR
  执行操作：　　(OPR) ← (OPR) - 1
- 求补指令：　　NEG OPR
  执行操作：　　(OPR) ← - (OPR)
- 比较指令：　　CMP OPR1, OPR2
  执行操作：　　(OPR1) - (OPR2)

减法指令对条件标志位（CF/OF/ZF/SF）的影响：

CF= { 1 被减数的最高有效位有向高位的借位
      0 否则

或

CF= { 1 减法转换为加法运算时无进位
      0 否则

OF= { 1 两个操作数符号相反，而结果的符号与减数相同
      0 否则

CF位表示无符号数减法的溢出。
OF位表示带符号数减法的溢出。

NEG(求补指令)指令对CF/OF的影响：
CF位：操作数为0时，求补的结果使CF=0，否则CF=1。
OF位：字节运算对-128求补或字运算对-32768求补时OF=1，
否则OF=0。

例：求SUB [SI+14H], 0136H

其中，(DS) = 3000H，(SI) = 0040H

分析：先求出物理地址，找到被减数

物理地址 = 段地址补0 + 偏移地址

= 30000H+0040H+14H

= 30054H

取一个字型数据为：4336H

内存分配情况

内存地址	内存单元
30050H	40 H
30051H	00 H
30052H	15 H
30053H	30 H
30054H	36H
30055H	43H
30056H	50H
⋮	

计算机实现是加补码

```
  4336H
- 0136H
--------
  4200H
```

-0136H的补码？0000 0001 0011 0110 原码

1111 1110 1100 1001 按位求反

+1 末位加1

1111 1110 1100 1010 补码

例　SUB [SI+14H], 0136H

(DS)=3000H, (SI)=0040H, (30054H)=4336H

```
    0100 0011 0011 0110
  + 1111 1110 1100 1010
  ---------------------
    0100 0010 0000 0000
1
```

SF=0, ZF=0, CF=0, OF=0

例3.49　SUB DH, [BP+4]

(DH)=41H, (SS)=0000H, (BP)=00E4H,(00E8)=5AH

41H
- 5AH

-5AH的补码？ 0101 1010　　原码

1010 0101　　按位求反

+1 末位加1

1010 0110　5A补码

+ 0100 0001　41H

1110 0111

SF=1, ZF=0, CF=1, OF=0

例：x、y、z均为双精度数，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 ← x+y+24-z，并用W, W+2单元存放w。

```
MOV  AX, X        ;因为双精度所以低位放入AX
MOV  DX, X+2      ;高位放DX
ADD  AX, Y        ;低位加
ADC  DX, Y+2      ;高位加，x+y
ADD  AX, 24       ;低位加，24
ADC  DX, 0        ;高位加，没有，实现了x+y+24
SUB  AX, Z        ;低位减
SBB  DX, Z+2      ;高位减，实现了 x+y+24-z
MOV  W, AX        ;存放低位结果，存入w单元
MOV  W+2, DX      ;存放高位结果，存入W+2单元
```

减1指令DEC

作用类似于C语言中的”——”操作符。

格式：**DEC opr**

操作：**opr←(opr)-1**

指令例：

DEC CL

DEC BYTE PTR[DI+2]

DEC SI

求补指令NEG

格式： **NEG opr**

操作： **opr← 0-(opr)**

对一个操作数取补码相当于用**0**减去此操作数，故利用**NEG**指令可得到负数的绝对值。

例：若**(AL)=0FCH**，则执行 **NEG AL**后，

(AL)=04H，**CF=1**

本例中，**0FCH**为**-4**的补码,执行求补指令后,即得到**4(-4**的绝对值)。

mov、add、sub指令

- add和sub指令同mov一样，都有两个操作对象。

ADD	寄存器, 数据	比如： ADD AX, 8
ADD	寄存器,寄存器	比如： ADD AX, BX
ADD	寄存器, 内存单元	比如： ADD AX, [0]
ADD	内存单元, 寄存器	比如： ADD [0], AX
SUB	寄存器, 数据	比如： SUB AX, 9
SUB	寄存器,寄存器	比如： SUB AX, BX
SUB	寄存器, 内存单元	比如： SUB AX, [0]
SUB	内存单元, 寄存器	比如： SUB [0], AX

- 将123B0H~123BAH的内存单元定义为数据段，累加这个数据段中的前3个单元中的数据，代码如下：

```
MOV  AX, 123BH    ;将123BH送入DS，作为数据段的段地址
MOV  DS,  AX
MOV AL,  0    ;用AL存放累加结果，先清0
ADD  AL, [0]  ;将数据段第一个单元（偏移地址为0），加入AL中
ADD  AL, [1]  ;将数据段第二个单元（偏移地址为1），加入AL中
ADD  AL, [2]  ;将数据段第三个单元（偏移地址为1），加入AL中
```

§ 3.3.2 算术指令—乘法指令

- 乘法指令

无符号数乘法指令：　　MUL　SRC
执行操作：字节操作数　(AX) ← (AL) * (SRC)
　　字操作数　(DX, AX) ← (AX) * (SRC)

带符号数乘法指令：　　IMUL　SRC

注意：
* AL(AX)为隐含的乘数寄存器。
* AX(DX,AX)为隐含的乘积寄存器。
* SRC不能为立即数。
* 除CF和OF外，对条件标志位无定义。
* 结果与乘数和被乘数的数据宽度不同。
* 先判断数据是什么数据（有无符号），再选取指令
→ (Why?)

(1) 无符号数的乘法指令 MUL

格式： **MUL src**

操作：字节操作数 **(AX) ← (AL) × (src)**

字操作数 **(DX, AX) ← (AX) × (src)**

指令示例：

MUL BL；(AL)×(BL),乘积在AX中

MUL CX；(AX)×(CX),乘积在DX,AX中

(2)有符号数乘法指令IMUL

格式与MUL指令类似，只是要求两操作数均为有符号数。

指令示例：

```
IMUL  BL        ；(AX)←(AL)×(BL)
IMUL  WORD PTR[SI]
                ；(DX,AX)←(AX)×([SI+1][SI])
```

注意：MUL/IMUL指令中

- AL(AX)为隐含的乘数寄存器；
- AX(DX,AX)为隐含的乘积寄存器；
- SRC不能为立即数；
- 除CF和OF外，对其它标志位无定义。

mul 指令（乘法指令）

- 格式：

  mul reg

  mul 内存单元 { mul byte ptr ds:[0]

  mul word ptr [bx+si+8]

- 相乘的两个数：要么都是8位，要么都是16位

  8 位： AL中和 8位寄存器或内存字节单元中

  16 位： AX中和 16 位寄存器或内存字单元中

- 结果

  8位：AX中；

  16位：DX（高位）和AX（低位）中。

乘法指令对CF/OF的影响：

MUL指令: CF/OF = { 0/0 乘积的高一半为零；1/1 否则

IMUL指令: CF/OF = { 0/0 乘积的高一半是低一半的符号扩展；1/1 否则

例：(AL) = A5H(-5B)，(BL) = 11H

```
(1)   IMUL  BL      ; (AX) ← (AL)×(BL)
                    ; A5×11 ⇒ -5B×11=-060B ⇒ F9F5
                    ; (AX) = F9F5H    CF=OF=1
(2)   MUL  BL       ; (AX) ← (AL)×(BL)
                    ; A5×11= 0AF5
                    ; (AX) = 0AF5H   CF=OF=1
```

mul 指令

- 例：计算100*10
  - 100和10小于255，可以做8位乘法，程序如下：

```
mov al,100
mov bl,10
mul bl
```

结果： (ax)=1000（03E8H）

mul 指令

- 例：计算100*10000
  - 100小于255，可10000大于255，所以必须做16位乘法，程序如下：

```
mov ax,100
mov bx,10000
mul bx
```

结果： (ax)=4240H，(dx)=000FH

（F4240H=1000000）

§ 3.3.2 算术指令—除法指令

- 除法指令

无符号数除法指令：　**DIV SRC**

执行操作：　字节操作　**(AL) ← (AX) / (SRC) 的商**

(AH) ← (AX) / (SRC) 的余数

字操作　**(AX) ← (DX, AX) / (SRC) 的商**

(DX) ← (DX, AX) / (SRC) 的余数

带符号数除法指令：　**IDIV SRC**

注意：

* **AX(DX,AX)为隐含的被除数寄存器。**
* **AL(AX)为隐含的商寄存器。**
* **AH(DX)为隐含的余数寄存器。**
* **SRC不能为立即数。**
* **对所有条件标志位均无定义。**

除法指令

进行除法时：16位/8位→8位商

32位/16位→16位商

对被除数、商及余数存放有如下规定：

	被除数	商	余数
字节除法	AX	AL	AH
字除法	DX:AX	AX	DX

div 指令

- **div**

除法指令，使用div作除法的时候：

- 除数：8位或16位，在寄存器或内存单元中
- 被除数：在AX 或 DX和AX中（默认）

(16位) (32位，高16位，低16位)

- 结果：

运算	除数8位	除数16位
商	AL	AX
余数	AH	DX

div指令格式：

div reg

div 内存单元

无符号数除法指令DIV

格式： **DIV src**

操作：字节操作 **(AL) ← (AX) / (SRC) 的商**

(AH) ← (AX) / (SRC) 的余数

字操作 **(AX) ← (DX, AX) / (SRC) 的商**

(DX) ← (DX, AX) / (SRC) 的余数

指令示例：

DIV CL

DIV WORD PTR[BX]

注：若除数为零或AL中商大于FFH(或AX中商大于FFFFH)，则CPU产生一个类型0的内部中断。

有符号数除法指令IDIV

格式： **IDIV src**

操作与**DIV**类似。商及余数均为有符号数,且余数符号总是与被除数符号相同。

注意： 对于**DIV/IDIV**指令

- **AX(DX,AX)**为隐含的被除数寄存器。
- **AL(AX)**为隐含的商寄存器。
- **AH(DX)**为隐含的余数寄存器。
- **src**不能为立即数。
- 对所有条件标志位均无定义。

除法操作中的字长扩展问题

- 除法运算要求被除数字长是除数字长的两倍,若不满足则需对被除数进行扩展,否则产生错误。
- 对于无符号数除法扩展，只需将**AH**或**DX**清零即可。
- 对有符号数而言,则是符号位的扩展。可使用前面介绍过的符号扩展指令**CBW**和**CWD**

符号扩展指令

CBW

执行操作： **AL → AX**

若(AL)的最高有效位为0，则(AH)= 00H

若(AL)的最高有效位为1，则(AH)= 0FFH

CWD

执行操作： **AX → (DX,AX)**

若(AX)的最高有效位为0，则(DX)= 0000H

若(AX)的最高有效位为1，则(DX)= 0FFFFH

```
例：(AX)=0BA45H
    CBW          ; (AX)=0045H
    CWD          ; (DX)=0FFFFH  (AX)=0BA45H
```

注意：
* 无操作数指令
* 隐含对AL或AX进行符号扩展
* 不影响条件标志位

div 指令用例

- 编程：利用除法指令计算100001/100
- 分析
  - 被除数 100001 大于65535，不能用ax寄存器存放，要用dx和ax两个寄存器联合存放100001，也就是说要进行16位的除法
  - 除数100小于255，可以在一个 8位寄存器中存放，但被除数是32位，除数应为16位，要用一个16位寄存器来存放除数100
  - 为dx和ax赋100001的高16位值和低16位值，应先将100001表示为十六进制形式：186A1H。
  - 程序如下

div 指令用例

- 编程：

利用除法指令计算100001/100。（程序）

```
mov dx,1
mov ax,86A1H  ;   (dx)*10000H+(ax)=100001
mov bx,100
div bx
```

程序执行后，(ax)=03E8H（即1000），(dx)=1（余数为1）

div 指令用例

- 编程：利用除法指令计算1001/100
- 分析
  - 被除数1001可用 ax寄存器存放，除数100可用 8位寄存器存放，也就是说，要进行8位的除法
  - 程序如下：

```
mov ax,1001
mov bl,100
div bl
```

程序执行后，(al)=0AH（即10），(ah)=1（余数为1）

div 指令用例

思考题: 写出34H÷25H的程序段

```
MOV  AL，34H
MOV  BL，25H
CBW                ; AL的符号扩展到AH
IDIV  BL           ; 0034H÷25H,结果为
                   ; (AH)=0FH, (AL)=01H
```

四则混合运算

例：x,y,z,v均为16位带符号数，计算(v-(x×y+z-540))÷x

```
MOV  AX, X
IMUL  Y                    ; x×y
MOV  CX, AX
MOV  BX, DX
MOV  AX, Z
CWD
ADD  CX, AX
ADC  BX, DX                ; x×y+z
SUB  CX, 540
SBB  BX, 0                 ; x×y+z-540
MOV  AX, V
CWD
SUB  AX, CX
SBB  DX, BX                ; v-(x×y+z-540)
IDIV  X                    ; (v-(x×y+z-540))÷x
```