

第3章

80x86的指令系统和寻址方式

- 3.1 80x86的寻址方式
- 3.2 程序占有的空间和执行时间
- 3.3 80x86的指令系统

3.1 80x86的寻址方式

- 3.1.1 与数据有关的寻址方式
- 3.1.2 有转移地址有关的寻址方式

指令格式

机器指令格式

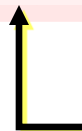
操作码 操作数

机器执行什么操作



执行对象

(具体数、存放位置)



例如：

MOV AX,1234H

MOV BX,AX

ADD AX,[1234H]

JMP ACTION1

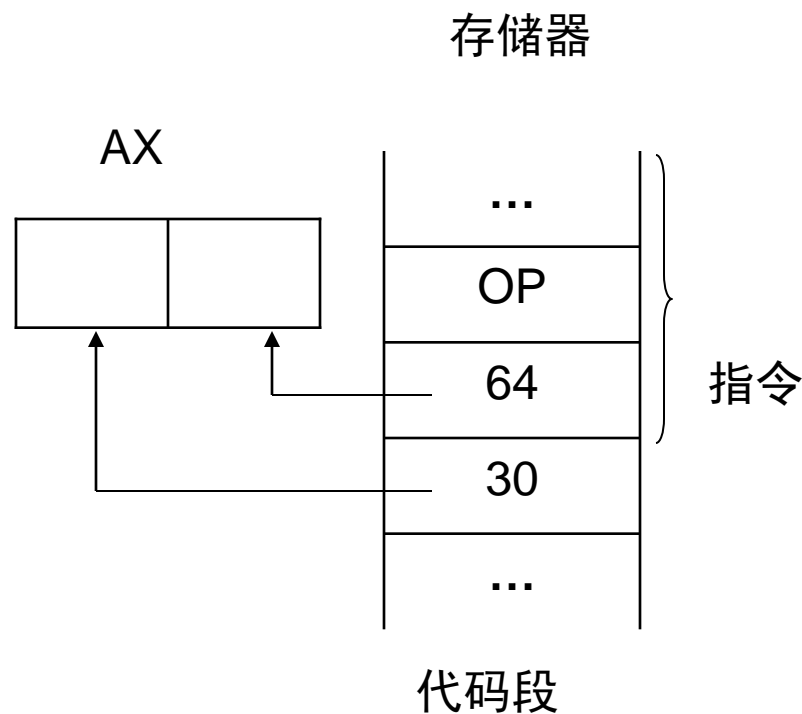
3.1.1 与数据有关的寻址方式

- 1. 立即寻址
- 2. 寄存器寻址
- 存储器寻址
 - 3. 直接寻址方式
 - 4. 寄存器间接寻址方式
 - 5. 寄存器相对寻址方式
 - 6. 基址变址寻址方式

- 8086/80286的字长是16位，一般情况下只处理8位和16位数
- 80386及后继机型字长为32位，可处理8、16和32位
- 数据传送指令MOV
格式：MOV DST, SRC
执行操作：(DST) \leftarrow (SRC)

1. 立即寻址方式

- 操作数直接存放在指令中，紧跟在操作码之后，作为指令的一部分存放在代码段中
- 例3.2 **MOV AL, 5**
 MOV AX, 3064H
- 常用于给寄存器赋初值
- 立即数只能做源操作数



2. 寄存器寻址方式

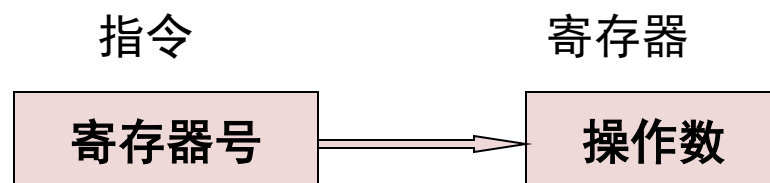
- 操作数在寄存器中，指令指定寄存器号

- 例如 `MOV AX, BX`

如指令执行前 $(AX)=3064H$ ， $(BX)=1234H$

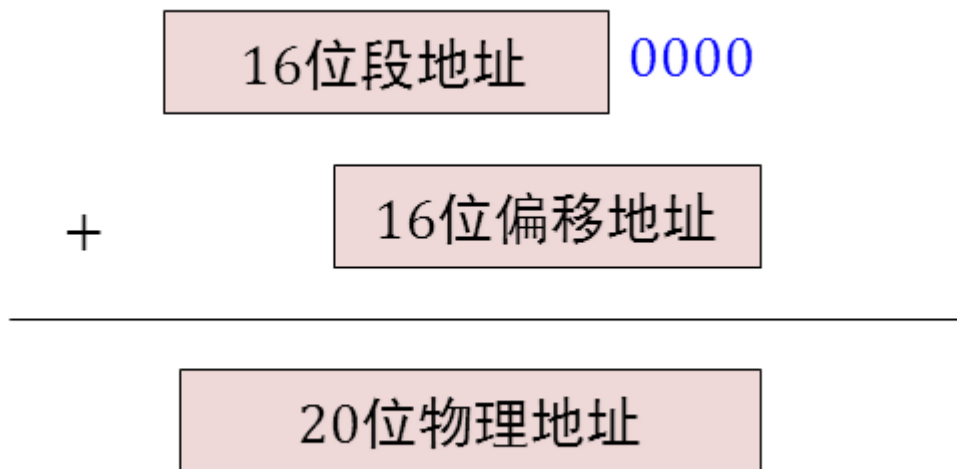
则指令执行后， $(AX)=?$ ， $(BX)=?$

- 这种寻址方式由于操作数就在寄存器中，不需要访问存储器，因而可以获得较高的运算速度



有效地址

- 实模式存储器寻址中，



物理地址 = 段地址左移4位 + 偏移地址

- 段基地址存储在段寄存器中
- 有效地址（EA）：操作数的偏移地址

数据段定义

data segment

```

    DB 7 dup(?)
    age DB 10
    value DW 60H
    array DW 1234H , 5678H ,
           0ABCDH, 76H

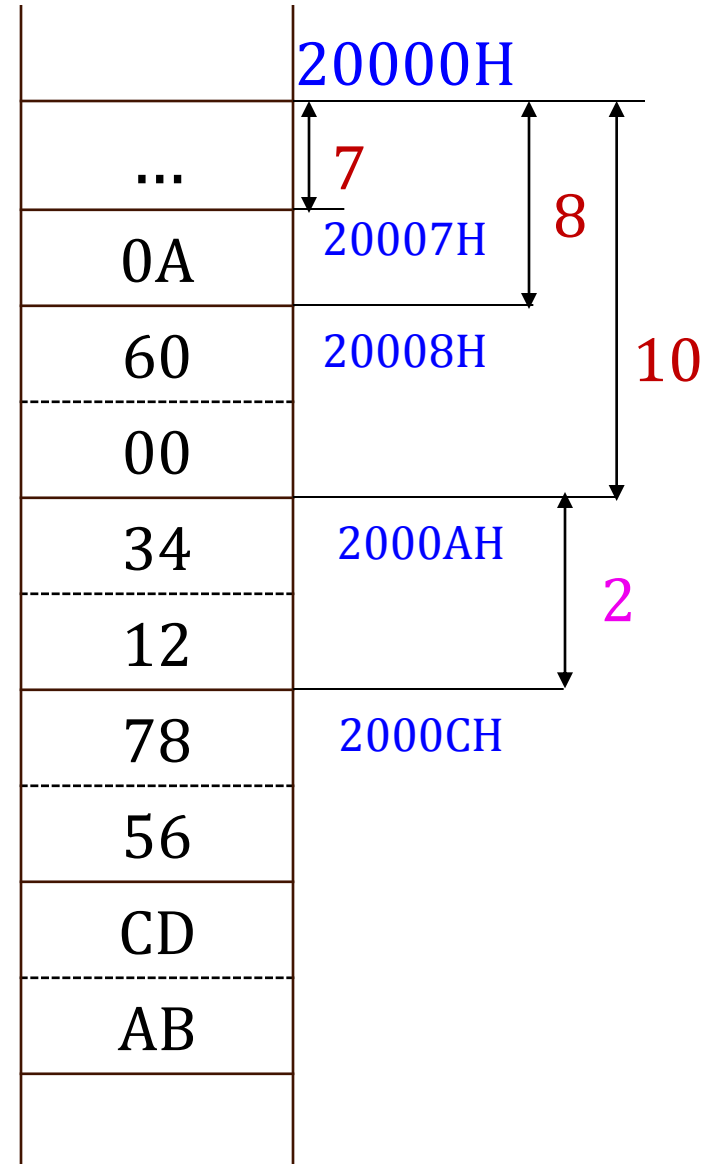
```

data ends

数据段

age
value

array



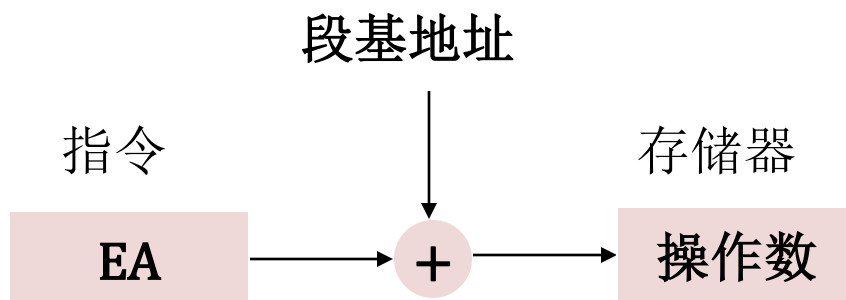
- 1) **位移量**：存放在指令中的一个8位、16位或32位数（386及后继机型），但不是立即数，而是一个地址
- 2) **基址**：存放在基址寄存器中，通常用来指向数据段中数组或字符串的首地址
- 3) **变址**：存放在变址寄存器中，通常用来访问数组中的某个元素或字符串中的某个字符

段寄存器和相应存放偏移地址的寄存器之间的默认组合

段	偏移
CS	IP
SS	SP或BP
DS	BX、DI、SI或一个16位数
ES	DI（用于串指令）

3. 直接寻址方式

操作数的有效地址只包含位移量一种成份，其值就存放在代码段中指令的操作码之后。位移量的值即操作数的有效地址

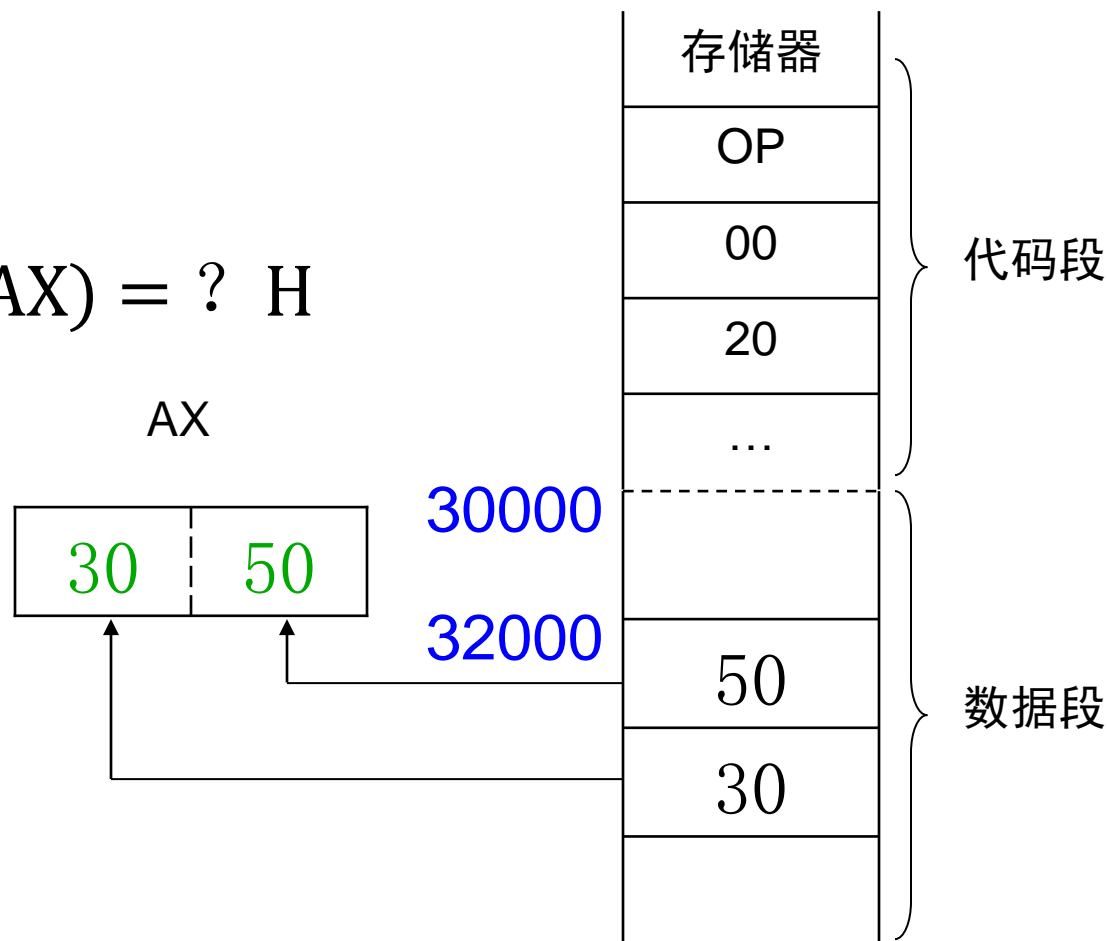


例3.5:

MOV AX, [2000H]

若 (DS) = 3000H

则执行结果为: (AX) = ? H



- 在汇编语言中，可以用符号地址代替数值地址
— 例：

```
VALUE DB 10
```

```
MOV AH, [ VALUE ]      或
```

```
MOV AH, VALUE
```

- 可使用段跨越前缀

```
MOV AX, ES: [2000H]
```

- 直接寻址方式适用于处理单个变量

— 例：

```
mov  al , [age]
```

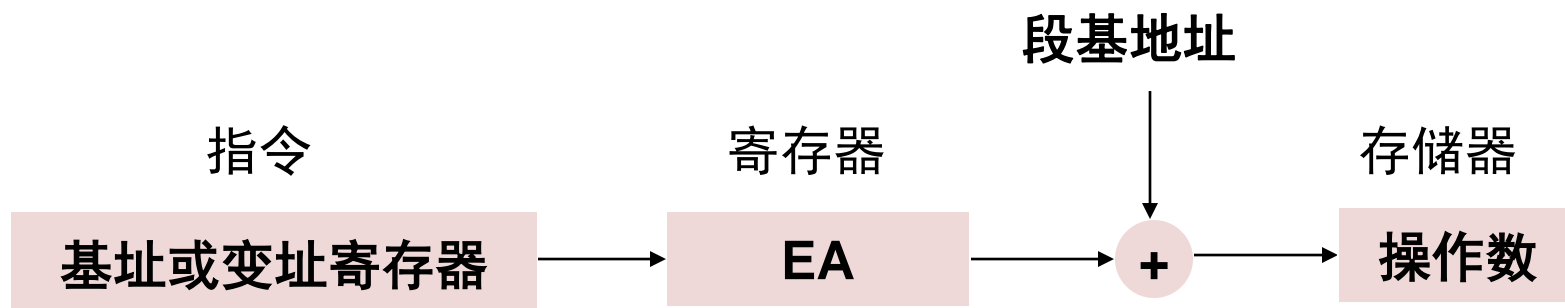
```
mov  al , age
```

```
mov  ax , [value]
```

```
mov  ax , value
```


4. 寄存器间接寻址

寄存器间接寻址方式中，**操作数的有效地址只包含基址寄存器内容或变址寄存器内容一种成份**。该寄存器的内容为操作数的偏移地址EA，操作数在存储器中。



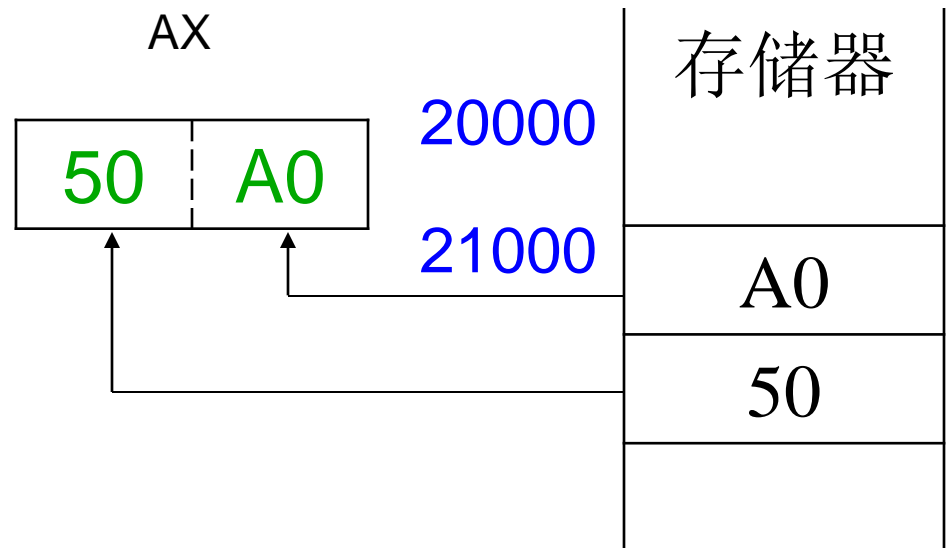
例3.7

如果: $(DS) = 2000H$, $(BX) = 1000H$

执行指令 `MOV AX, [BX]`

$(AX) = ?H$

物理地址 = $20000 + 1000 = 21000H$

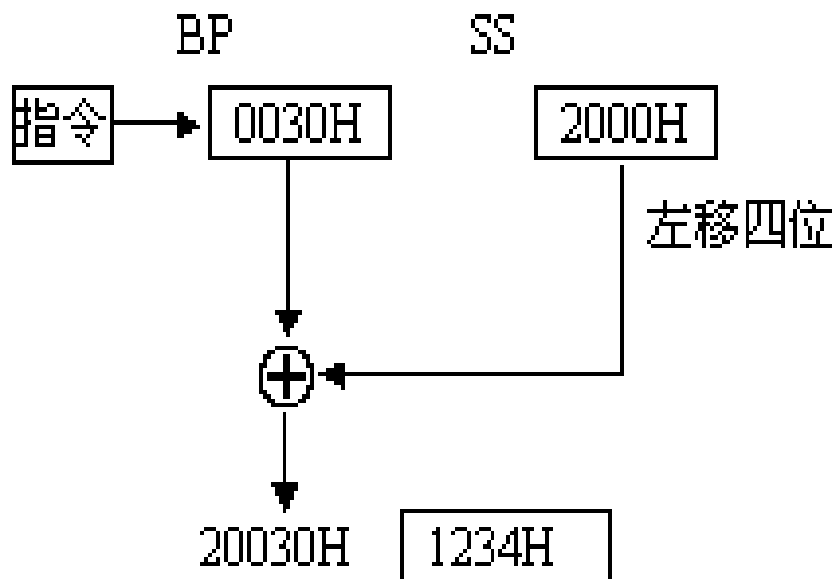


寄存器和存储器内容分别为：

(AX)=0 (BP)=0030H (DS)=3000H (SS)=2000H
(20030H)=1234H (30030H)=5678H

执行指令：MOV AX, [BP]

执行后：(AX)=?, (BP)=?, (SS)=?,
(20030H)=?

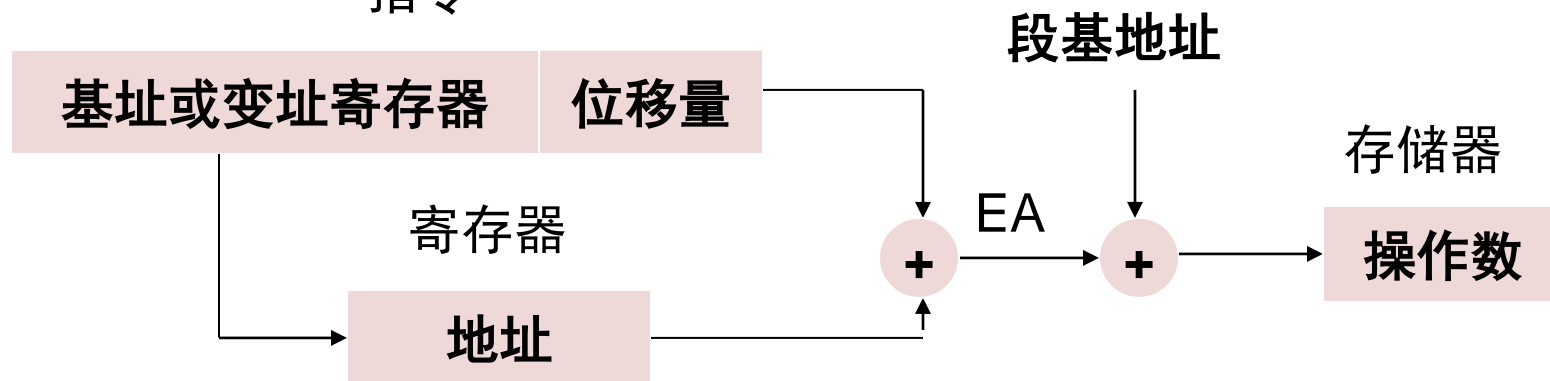


5. 寄存器相对寻址方式(直接变址寻址)

操作数的有效地址为基址寄存器或变址寄存器的内容与指令中的位移量之和。

用途：处理一维数组或字符串

$$\begin{array}{c} \text{有效地址} \\ = \\ \text{指令} \end{array} \left\{ \begin{array}{l} (\text{BX}) \\ (\text{BP}) \\ (\text{SI}) \\ (\text{DI}) \end{array} \right\} + \left\{ \begin{array}{l} 8 \\ 16 \end{array} \right\} \text{位移量}$$



例3.9

执行指令: MOV AX, COUNT[SI]

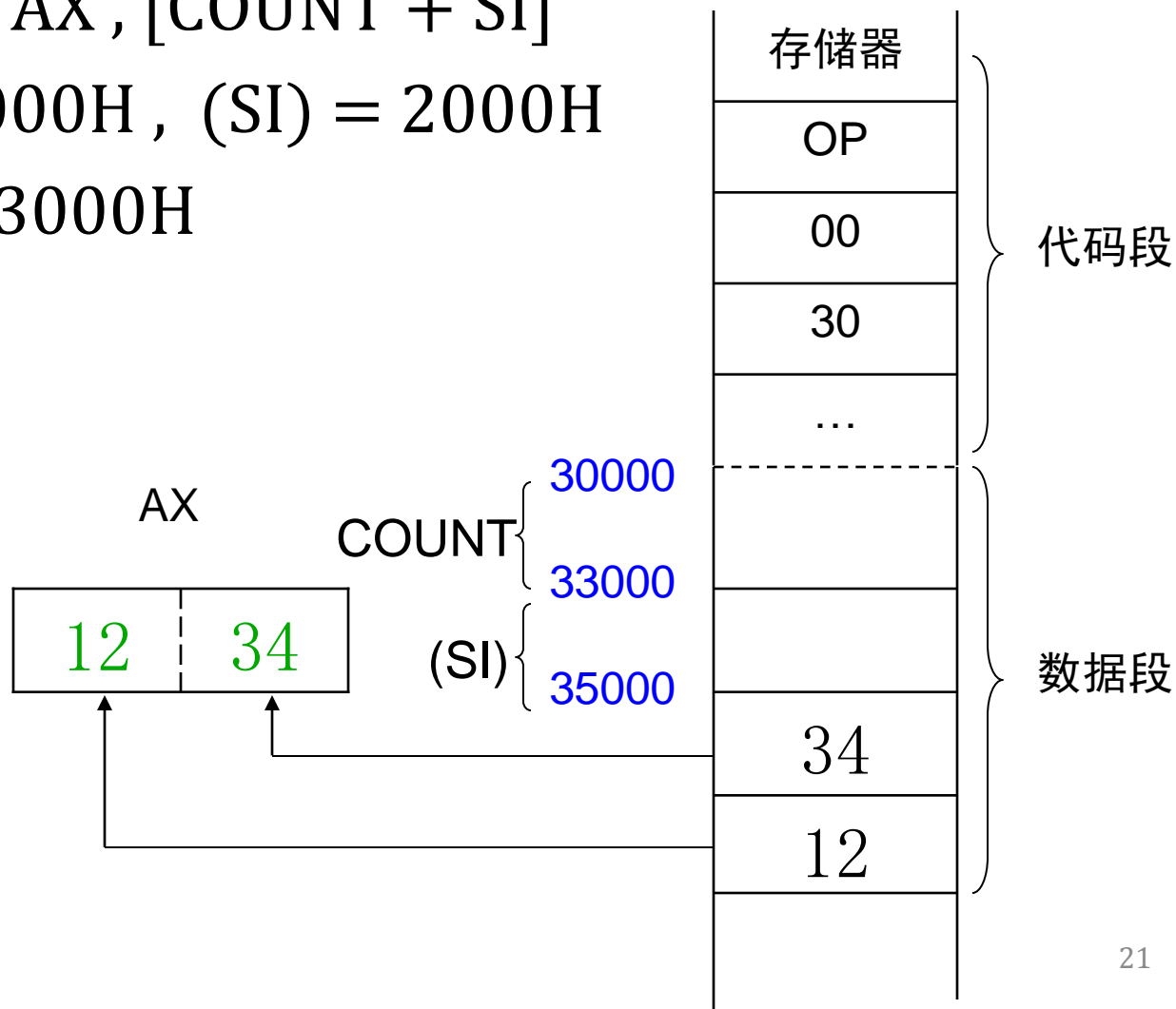
或 MOV AX, [COUNT + SI]

如果: (DS) = 3000H, (SI) = 2000H

COUNT = 3000H

则物理地址为:

(AX) = ?

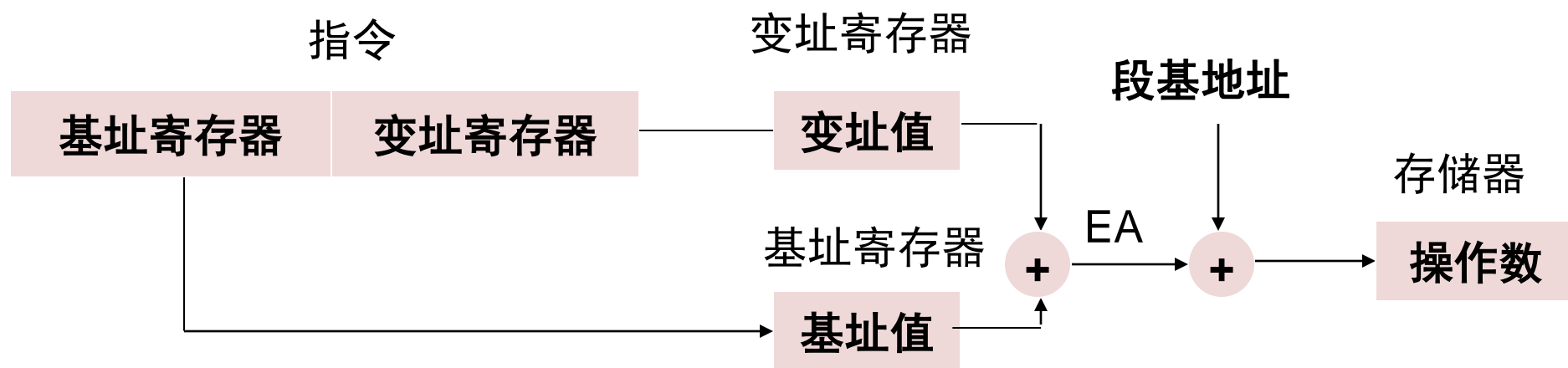


6. 基址变址寻址方式

操作数的有效地址是一个基址寄存器和一个变址寄存器的内容之和。

用途：数组和字符串

$$\text{有效地址} = \left\{ \begin{array}{l} (\text{BX}) \\ (\text{BP}) \end{array} \right\} + \left\{ \begin{array}{l} (\text{SI}) \\ (\text{DI}) \end{array} \right\}$$



例3.10

MOV AX, [BX][DI]

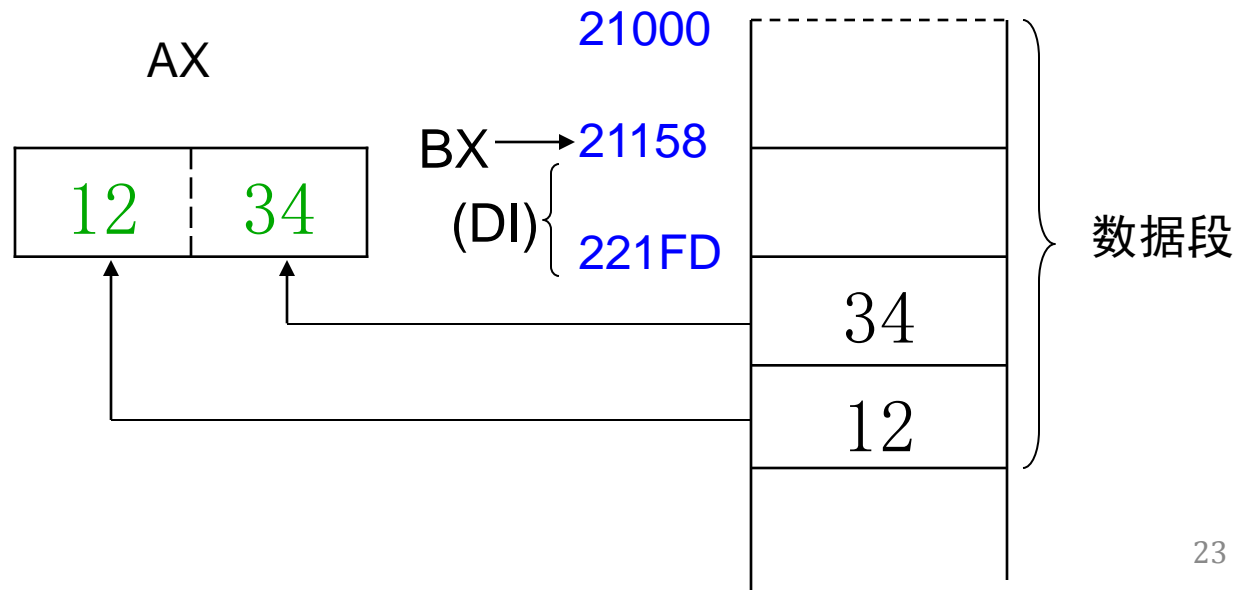
或 MOV AX, [BX + DI]

若 (DS) = 2100H, (BX) = 0158H, (DI) = 10A5H

则 EA = ?

物理地址 = ?

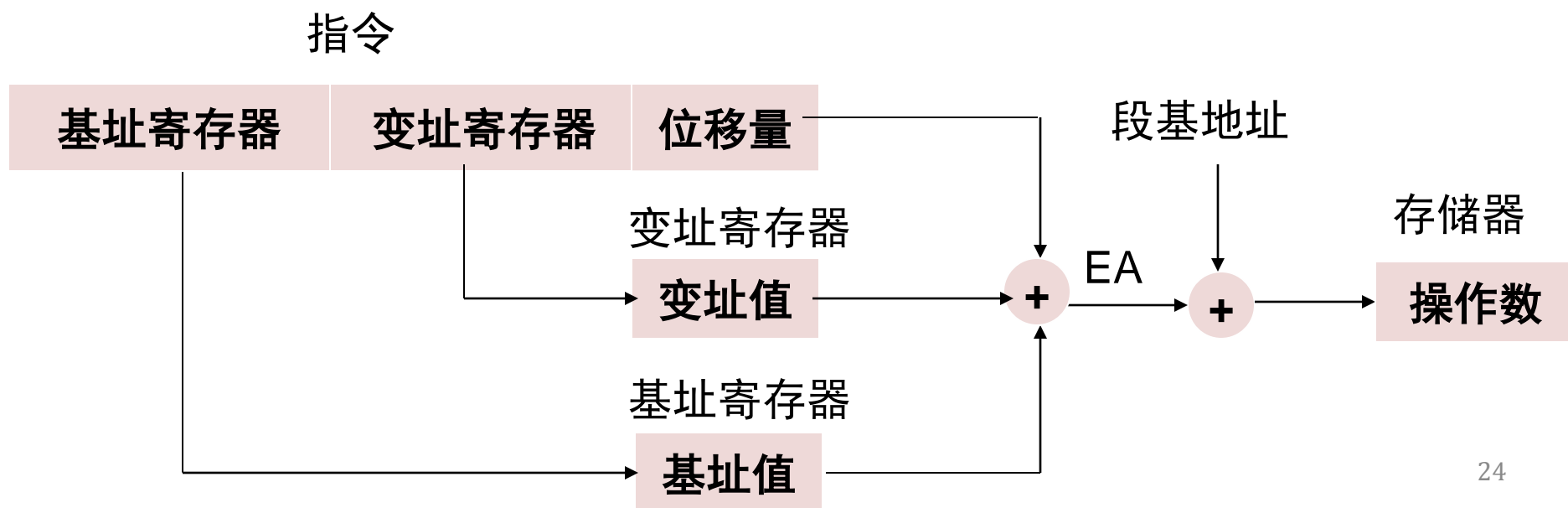
(AX) = ?



7. 相对基址变址寻址方式

操作数的有效地址是一个基址寄存器和一个变址寄存器的内容与指令中指定的位移量之和。

用途：二维数组



例3.11

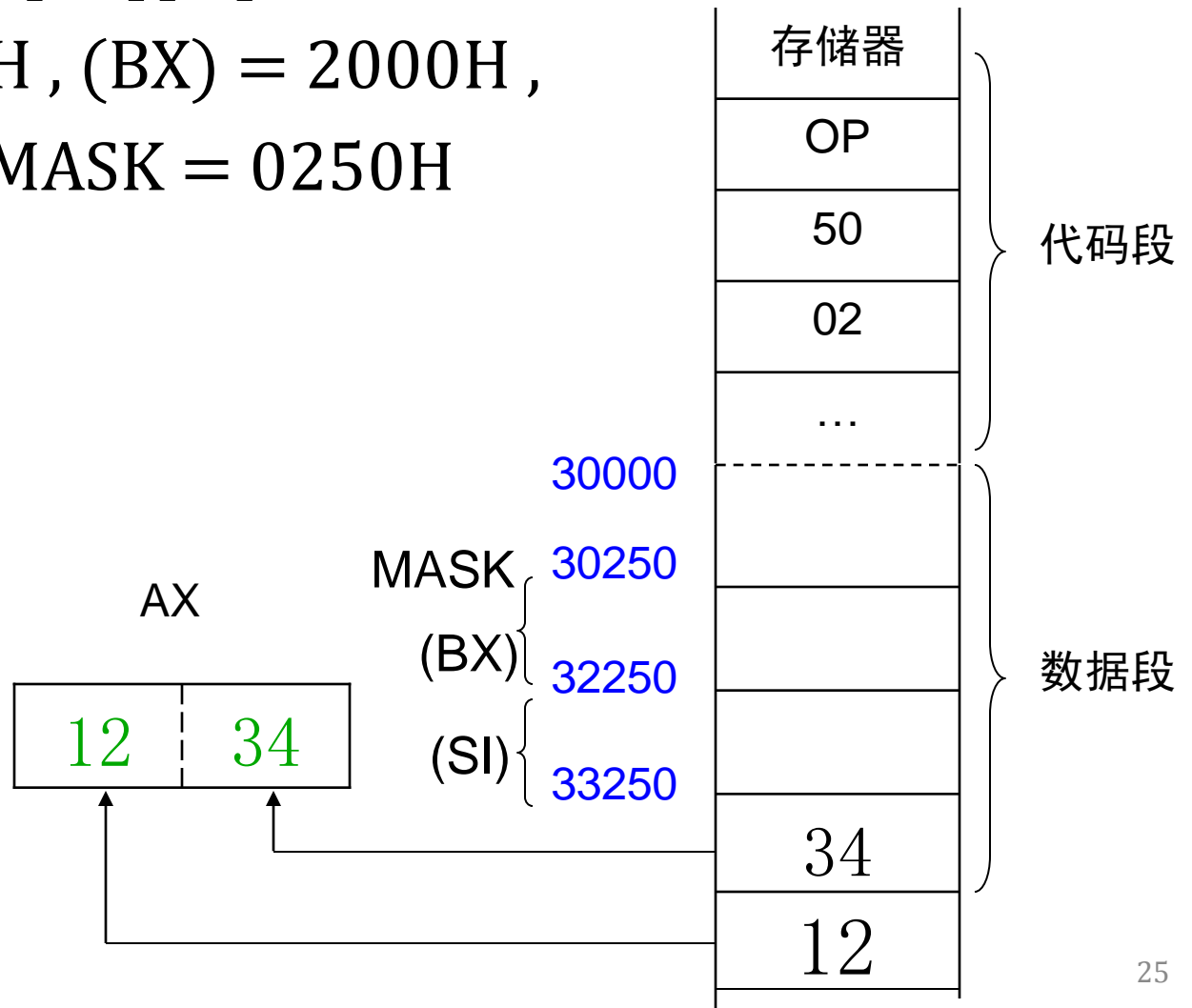
MOV AX, MASK[BX][SI]

若(DS) = 3000H, (BX) = 2000H,

(SI) = 1000H, MASK = 0250H

则物理地址=?

(AX) = ?



3.1.1 与数据有关的寻址方式

- 1. 立即寻址 `MOV AX, 1234H`
- 2. 寄存器寻址 `MOV AX, BX`
- 存储器寻址
 - 3. 直接寻址方式 `MOV AX, [VALUE]`
 - 4. 寄存器间接寻址方式 `MOV AX, [BX]`
 - 5. 寄存器相对寻址方式 `MOV AX, ARRAY[BX]`
 - 6. 基址变址寻址方式 `MOV AX, [BX][SI]`

3.3 80x86的指令系统

- 3.3.1 数据传送指令
- 3.3.2 算术指令
- 3.3.3 逻辑指令
- 3.3.4 串处理指令
- 3.3.5 控制转移指令

3.3.1 数据传送指令

- 1. 通用数据传送指令
- 2. 累加器专用传送指令
- 3. 地址传送指令
- 4. 标志寄存器传送指令
- 5. 类型转换指令

1.通用数据传送指令

- MOV传送指令

(1) MOV传送指令

- 格式: **MOV DST, SRC**
- 执行操作: **(DST) ← (SRC)**

DST:目的操作数, 可以为: mem、reg、segreg

SRC:源操作数, 可以为: mem、reg、segreg或data

- 说明:
 - 双操作数指令不允许两个操作数都使用存储器
 - 立即数不能直接送段寄存器 **MOV DS, 2000H** ×
 - DST 不能是立即数和CS
 - DST、SRC 不能同时为段寄存器 **MOV DS, ES** ×
 - 不影响标志位

(1) MOV传送指令

- 例3.20 `MOV AX, DATA_SEG`
 `MOV DS, AX`
- 例3.21 `MOV AL, 'E'`
- 例3.22 `MOV BX, OFFSET TABLE`
- 例3.23 `MOV AX, Y[BP][SI]`

1. 通用数据传送指令

- 堆栈操作指令p49
 - **PUSH** 进栈
 - **POP** 出栈

堆栈

- 堆栈是以“**后进先出**”方式工作的一个存储区，必须存在于堆栈段中，段地址存放于**SS**寄存器中
- **堆栈只有一个出入口**，所以只有一个堆栈指针寄存器，当堆栈地址长度为16或32时，分别使用SP或ESP
- **SP或ESP的内容在任何时候都指向当前的栈顶**，所有进栈和出栈指令都根据SP或ESP的内容来确定进栈或出栈的存储单元，而且必须及时修改指针，以保证SP或ESP指向当前的栈顶

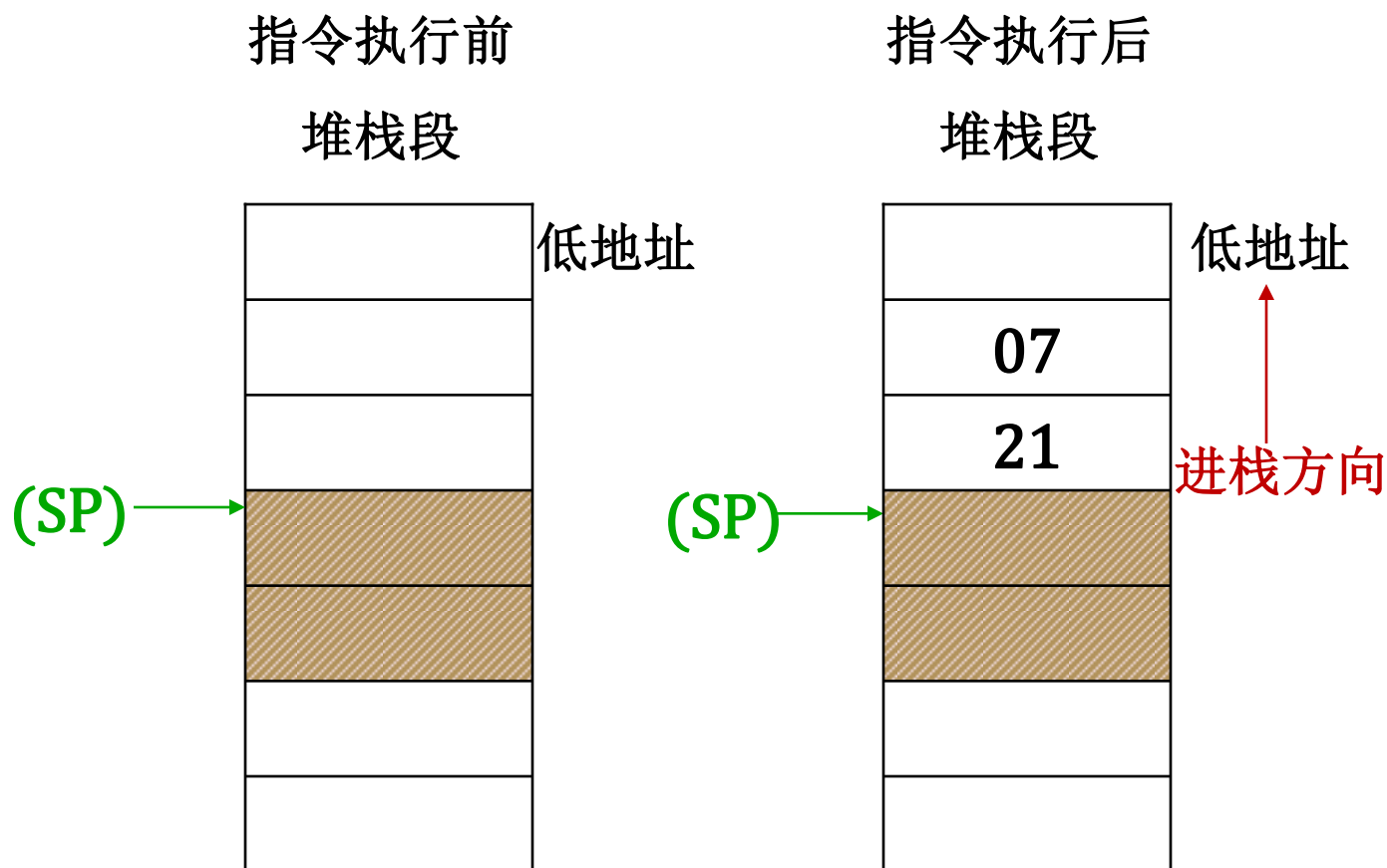
(2) PUSH 进栈指令

- 格式: **PUSH SRC**
隐含目的操作数: **SS:SP**
- 执行操作:
16位操作数: $(SP) \leftarrow (SP)-2$
 $((SP)+1, (SP)) \leftarrow SRC$
32位操作数: $(ESP) \leftarrow (ESP)-4$
 $((ESP)+3, (ESP)+2, (ESP)+1, (ESP)) \leftarrow SRC$
- 有4种格式: **PUSH reg / mem / data / segreg**
- 功能: 将寄存器、段寄存器或存储器中的数据或立即数压入堆栈, 堆栈指针减2或4, 指向栈顶

例3.29

MOV AX, 2107H

PUSH AX

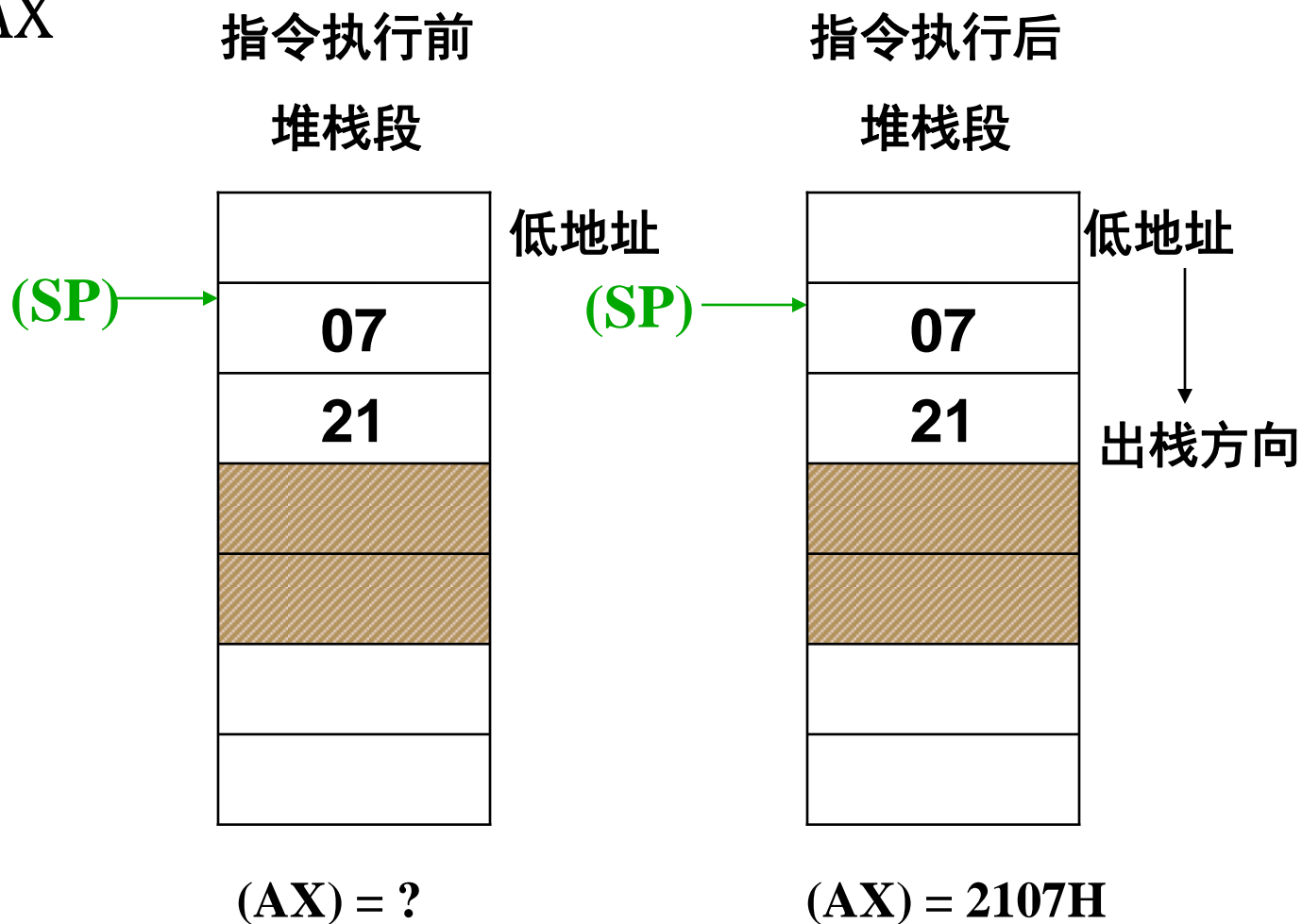


(3) POP 出栈指令

- 格式: **POP DST**
隐含源操作数: **SS:SP**
- 执行操作:
16位操作数: $(DST) \leftarrow ((SP)+1, (SP))$
 $(SP) \leftarrow (SP)+2$
32位操作数:
 $(DST) \leftarrow ((ESP)+3, (ESP)+2, (ESP)+1, (ESP))$
 $(ESP) \leftarrow (ESP)+4$
- 有3种格式: **POP reg / mem / segreg**(除CS寄存器以外)
- 功能: 将栈顶元素弹出送至某一寄存器、段寄存器(除CS外)或存储器, 堆栈指针加2或4

例3.30

POP AX



保存寄存器的值:

PUSH AX

PUSH BX

PUSH CX

..... ;其间用到AX、BX、CX

POP CX ; 后进先出

POP BX

POP AX

.....

(4) XCHG(exchange)交换指令(p52)

- 格式: **XCHG OPR1 , OPR2**
执行操作: **(OPR1) ↔ (OPR2)**
- 可以在寄存器之间或寄存器与存储器之间交换信息, 但不允许使用段寄存器

例3.34

XCHG BX, [BP + SI]

如指令执行前: (BX)=6F30H, (BP)=0200H,
(SI)=0046H, (SS)=2F00H, (2F246H)=4154H

则OPR2的物理地址=2F000+0200+0046=
2F246H

指令执行后:

(BX)= 4154H, (2F246H)= 6F30H

2. 地址传送指令(p55)

- LEA 有效地址送寄存器

(1) LEA有效地址送寄存器

- 格式: **LEA REG, SRC**
操作: **(REG) ← SRC的有效地址**
- 目的操作数可使用16位或32位寄存器, 但不能使用段寄存器
- 源操作数可使用除立即数和寄存器外的任一种存储器寻址方式

例3.40

若 $(BX) = 0400H$ $(SI) = 003CH$
 $(DS) = 2000H$ $(2139E) = 2195H$

则执行指令

LEA BX, [BX + SI + 0F62H] 后, $(BX) = ?$

$$(BX) = 0400 + 003C + 0F62 = 139EH$$

执行指令

MOV BX, [BX + SI + 0F62H] 后, $(BX) = ?$

$$(BX) = 2195H$$

- 例3.41 LEA BX, LIST
 MOV BX, OFFSET LIST

OFFSET **variable或label**

回送标号或变量的偏移地址

TABLE (DS):1000H		MOV BX, TABLE	; (BX)=0040H
	40	MOV BX, OFFSET TABLE	; (BX)=1000H
	00	LEA BX, TABLE	; (BX)=1000H
	00		
	30		

3. 类型转换指令(p58)

- **CBW**-字节转换为字
 $(AX) \leftarrow (AL)$ 符号扩展

执行操作:

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)= FFH

- **CWD**-字转换为双字
 $(DX:AX) \leftarrow (AX)$ 符号扩展

执行操作:

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)= FFFFH

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

3.3.2 算术指令

- 加法指令
- 减法指令
- 乘法指令
- 除法指令

- 双操作数的两个操作数中除源操作数为立即数的情况外，必须有一个操作数在寄存器中。
- 单操作数指令不允许使用立即数方式。

标志位

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF			SF	ZF		AF		PF		CF

条件标志中最主要的是CF、ZF、SF和OF

- 1) SF(符号标志): 运算结果为负时置1, 否则置0
- 2) ZF(零标志): 运算结果为零时置1, 否则置0
- 3) CF(进位或借位)
- 4) OF(溢出)

1 加法指令

- **ADD** 加法
- **ADC** 带进位加法
- **INC** 加1

加法指令中CF位和OF位的设置

- CF位
 - 执行加法指令时，**CF位根据最高有效位是否有进位设置，有进位时CF=1, 无进位时CF=0**
 - 表示无符号数的溢出；
双字长数运算中，利用CF位的值把低位字的进位计入高位字中。
- OF位
 - **若两个操作数的符号相同，而结果的符号与之相反时OF=1，否则OF=0**
 - 表示有符号数的溢出


(1) ADD(add) 加法

- 格式: **ADD DST, SRC**

操作: **$(DST) \leftarrow (DST) + (SRC)$**

- 例3.45: **ADD DX, 0F0F0H**

如指令执行前 **$(DX) = 4652H$**

则	4652		0100 0110 0101 0010
	+F0F0		+ 1111 0000 1111 0000
	<hr/>		<hr/>
			1 0011 0111 0100 0010

执行指令后:

$(DX) = 3742H$, **$ZF = 0$** , **$SF = 0$** , **$CF = 1$** , **$OF = 0$**

(2) ADC(add with carry) 带进位加法

- 格式: ADC DST, SRC

操作: $(DST) \leftarrow (DST) + (SRC) + CF$

- 例3.46:

下列指令可在8086/80286中实现两个双精度数的加法。

设目的操作数存放在DX和AX寄存器中，其中DX存放高位字。源操作数存放在BX和CX寄存器中，BX存放高位字。

如指令执行前：

(DX)=0002H, (AX)=0F365H,

(BX)=0005H, (CX)=0E024H

指令序列为 **ADD AX, CX**

ADC DX, BX

则第一条指令执行后

(AX)=0D389H, SF=1, ZF=0, **CF=1**, OF=0

第二条指令执行后

(DX)=0008H, SF=0, ZF=0, CF=0, **OF=0**

该指令序列执行完后

(DX)=0008H, (AX)=0D389H

(3) INC(increment) 加1

- 格式: **INC OPR**
操作: **$(\text{OPR}) \leftarrow (\text{OPR}) + 1$**
- 不影响CF 标志位
- 常用于对计数器进行加1的操作

MOV CX, 1

LOOP1:

... .. 循环指令

INC CX

CMP CX, 10

JNZ LOOP1

2. 减法指令

- SUB

减法

- SBB

带借位减法

- DEC

减1

- NEG

求补

- CMP

比较

减法指令中CF位和OF位的设置

- CF位
 - 若减数 > 被减数，此时有借位，则CF=1, 否则CF=0
 - 无符号数相减的溢出；
被减数的最高有效位向高位的借位。
- OF位
 - 若两个操作数的符号相反，而结果的符号与减数相同，则OF=1，否则OF=0
 - 有符号数的减法溢出

(1) SUB(subtract) 减法

- 格式: **SUB DST, SRC**
操作: **(DST) ← (DST) - (SRC)**
- 例3.48: SUB [SI+14H], 0136H

如指令执行前

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

则执行指令:

4336	→	0100 0011 0011 0110	→	0100 0011 0011 0110
-0136		-0000 0001 0011 0110		+1111 1110 1100 1010
		0100 0010 0000 0000		≠ 0100 0010 0000 0000

(30054H)=4200H, SF=0, ZF=0, CF=0, OF=0

例3.49

SUB DH, [BP+4]

如指令执行前

(DH)=41H, (SS)=0000H, (BP)=00E4H, (000E8)=5AH

则执行指令

$$\begin{array}{r} 41 \\ -5A \\ \hline \end{array} \quad \Rightarrow \quad \begin{array}{r} 0100\ 0001 \\ -0101\ 1010 \\ \hline \end{array} \quad \Rightarrow \quad \begin{array}{r} 0100\ 0001 \\ +1010\ 0110 \\ \hline 1110\ 0111 \end{array}$$

(DH)=0E7H, SF=1, ZF=0, CF=1, OF=0

-19H的补码

(2) SBB(subtract with borrow) 帶借位減法

- 格式: **SBB DST, SRC**
操作: **$(DST) \leftarrow (DST) - (SRC) - CF$**
- 8086/80286实现双字减法, 80386后继机型实现4字减法

例3.50

设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，存放时高位字在高地址中，低位字在低地址中。在8086/80286中实现 $w \leftarrow x + y + 24 - z$ ，并用w和w+2存放结果

; X+Y

MOV AX, X

MOV DX, X+2

ADD AX, Y

ADC DX, Y+2

; +24

ADD AX, 24

ADC DX, 0

; -Z

SUB AX, Z

SBB DX, Z+2

;结果存入W, W+2

MOV W, AX

MOV W+2, DX

(3) DEC(decrement) 减1

- 格式: **DEC OPR**
操作: **$(\text{OPR}) \leftarrow (\text{OPR}) - 1$**
- 减1指令一般用于对计数器和地址指针的调整

MOV CX, 10

SUM: ... ;循环指令

DEC CX

JNZ SUM

(4) NEG(negate) 求补

- 格式: **NEG OPR**
操作: **$(\text{OPR}) \leftarrow -(\text{OPR})$**
操作数按位求反后加1
- 例如:

MOV AL, 0FEH	;(AL)=0FEH=[-2] _补
NEG AL	;(AL)=2
MOV CL, 3	;(CL)=3
NEG CL	;(CL)=0FDH=[-3] _补

(5) CMP(COMPARE) 比较

- 格式: **CMP OPR1, OPR2**
操作: **(OPR1) - (OPR2)**
该指令执行减法操作, 但并不保存运算结果, 只是根据结果设置条件标志位。
- **CMP**指令后往往跟着条件转移指令, 根据比较结果产生不同的程序分支。

CMP AL, 50 ;(AL)-50

JB BELOW ;(AL)<50

SUB AL, 50 ;(AL)>=50

INC AH ;(AH)+1→AH

BELOW: ...

3 乘法指令

- **MUL** 无符号数乘法
- **IMUL** 带符号数乘法

MUL(unsigned multiple) 无符号数乘法

- 格式: MUL SRC
- 执行的操作:
 - 字节操作数: $(AX) \leftarrow (AL) * (SRC)$
 - 字操作数: $(DX, AX) \leftarrow (AX) * (SRC)$
 - 双字操作数: $(EDX, EAX) \leftarrow (EAX) * (SRC)$
- 目的操作数必须是累加器(AL、AX、EAX), 两个8、16、32位数相乘, 分别得到16、32、64位乘积

- 源操作数可以使用除立即数以外的任何一种寻址方式
- 标志位：如果乘积的高一半为0，则CF和OF位均为0，否则为1。用以检查相乘的结果是字节、字、双字还是4字

例3.52

MUL BL

如指令执行前 $(AL) = 0B4H$, $(BL) = 11H$

则指令执行后

$(AX) = 0BF4H$

$CF = OF = 1$

如指令执行前 $(AL) = 04H$, $(BL) = 11H$

则指令执行后

$(AX) = 0044H$

$CF = OF = 0$

IMUL(signed multiple) 带符号数乘法

- 格式: IMUL SRC
- 执行的操作、目的操作数和源操作数与MUL指令相同
- 标志位: 如果乘积的高一半是低一半的符号扩展, 则CF位和OF位均为0, 否则均为1

例3.52

如 $(AL) = 0B4H$, $(BL) = 11H$

- `IMUL BL` ;带符号数乘法

$(AL) = 0B4H$, 其原码为 $0CCH$, 无符号数为 $180D$, 带符号数为 $-76D$;

$(BL) = 11H$, 其原码为 $11H$, 无符号数为 $17D$, 带符号数为 $17D$

则执行指令后

$(AX) = 0FAF4H$, 其原码为 $850CH$, 即 $-1292D$

$CF = OF = 1$

- `MUL BL` ;无符号数乘法

则指令执行后

$(AX) = 0BF4H = 3060D$ $CF = OF = 1$

4 除法指令

- **DIV** 无符号数除法
- **IDIV** 带符号数除法

DIV和IDIV

- 格式: DIV SRC
IDIV SRC

- 执行的操作

字节操作: $(AL) \leftarrow (AX) / (SRC)$ 的商

$(AH) \leftarrow (AX) / (SRC)$ 的余数

字操作: $(AX) \leftarrow (DX, AX) / (SRC)$ 的商

$(DX) \leftarrow (DX, AX) / (SRC)$ 的余数

双字操作: $(EAX) \leftarrow (EDX, EAX) / (SRC)$ 的商

$(EDX) \leftarrow (EDX, EAX) / (SRC)$ 的余数

- 除数、商、余数分别为8、16、32位时，被除数则为16、32、64位
- 目的操作数（被除数）必须存放在AX 或DX, AX或 EDX, EAX中，源操作数可以用除立即数以外的任何一种寻址方式
- DIV指令中商和余数均为无符号数
IDIV指令中商和余数均为带符号数，且余数的符号和被除数的符号相同
- 除法指令对所有条件码均无定义

- 例3.56: 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 → (dx:ax)
mov	cx,ax	
mov	bx,dx	;(dx:ax) → (bx:cx)即x*y → (bx:cx)
mov	ax,z	
cwd		;z符号扩展 → (dx:ax)
add	cx,ax	
adc	bx,dx	;+z → (bx:cx)
sub	cx,540	
sbb	bx,0	;-540 → (bx:cx)
mov	ax,v	
cwd		;v符号扩展 → (dx:ax)
sub	ax,cx	
sbb	dx,bx	;v- → (dx:ax)
idiv	x	;/x → 商ax, 余数dx

3.3.3 逻辑指令

- 1 逻辑运算指令
- 2 移位指令

1 逻辑运算指令

- **AND** 逻辑与
- **OR** 逻辑或
- **NOT** 逻辑非
- **XOR** 异或
- **TEST** 测试

逻辑运算

与运算(AND)

A	B	$A \wedge B$	作用
0	0	0	置0
1	0	0	
0	1	0	保留
1	1	1	

或运算(OR)

A	B	$A \vee B$	作用
0	0	0	保留
1	0	1	
0	1	1	置1
1	1	1	

异或运算(XOR)

A	B	$A \nabla B$	作用
0	0	0	保留
1	0	1	
0	1	1	取反
1	1	0	

非运算(NOT)

A	$\neg A$	作用
0	1	取反
1	0	

(1) 逻辑与指令AND

- 格式: **AND DST , SRC**
操作: $(DST) \leftarrow (DST) \wedge (SRC)$
- 功能: **屏蔽某些位（将这些位置0）**
- 例3.57: 屏蔽数0BFH的0、1两位，其它位保持不变

MOV AL, 0BFH ;1011 1111

AND AL, 0FCH ;1111 1100

则 (AL) = 1011 1100

(2) 逻辑或指令

- 格式: **OR** **DST, SRC**
操作: $(DST) \leftarrow (DST) \vee (SRC)$
- 功能: **将某些位置1**
- 例3.58: 将数43H第5位置1, 其它位保持不变

MOV AL, 43H ;0100 0011

OR AL, 20H ;0010 0000

则(AL)=0110 0011

(3) 逻辑非指令NOT

- 格式: NOT OPR
- 操作: (OPR) \leftarrow (OPR)按位求反
- 例如:

MOV AL, 8BH ;1000 1011

NOT AL ;0111 0100

则(AL) = 74H

(4) 异或指令XOR

- 格式: XOR DST , SRC
操作: $(DST) \leftarrow (DST) \vee (SRC)$
- 功能: 某些位变反
- 例3.61: 数11H低0、1位变反, 其它位保持不变

MOV AL, 11H ;0001 0001

XOR AL, 03H ;0000 0011

(AL) = 0001 0010

(5) 测试指令TEST

- 格式: TEST OPR1 , OPR2
操作: $(OPR1) \wedge (OPR2)$
- 功能: 测试某些位是否为0
- 例3.59: 测试数40H的第0、1、2、3、5、7位是否为0

MOV AL, 40H ;0100 0000

TEST AL, 0AFH ;1010 1111

;0000 0000

SF = 0 , ZF = 1, 说明所需测试的位均为0

2 移位指令

- 算术移位: **SAL SAR**
- 逻辑移位: **SHL SHR**
- 循环移位: **ROL ROR**
- 带进位循环移位: **RCL RCR**

移位指令的格式

- 格式： 移位指令 OPR, CNT
- 操作数OPR可用除立即数外的任何寻址方式
- CNT指明移位次数，移位次数为1时，可直接写在指令中；移位次数大于1时，则需要把次数置于CL寄存器中，指令中写CL
- CF位根据各条指令的规定设置
OF只有当CNT=1时才有效，移位后最高有效位的值发生变化，OF=1,否则OF=0

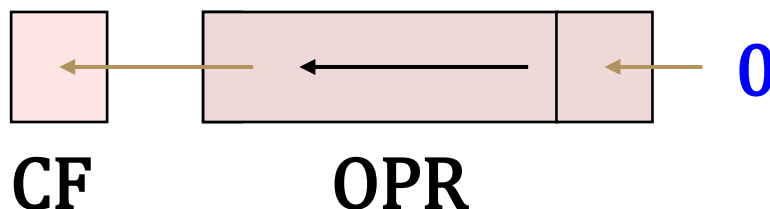
(1) 移位指令

- SHL
- SHR
- SAL
- SAR

① 逻辑左移指令SHL(shift logical left)

- 格式: **SHL** **OPR, CNT**

操作: 将(OPR)向左移动CNT指定的次数, **空出的低位补入相应个数的0**, CF的内容为最后移入位的值。



- 例3.66:

MOV CL, 2

SHL SI, CL

如指令执行前(SI) = 1450H; 0001 0100 0101 0000

则指令执行后 ~~00~~01 0100 0101 0000 **00**

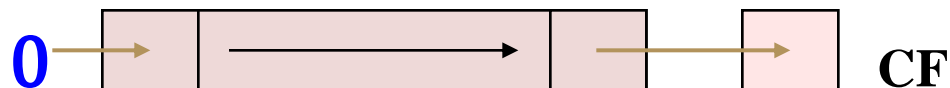
(SI) = 5140H CF = 0

计算: $1450H * 2^2 = ?$

② 逻辑右移指令SHR(shift logic right)

- 格式: **SHR** **OPR, CNT**

操作: 将(OPR)向右移动CNT指定的次数, **空出的高位补入相应个数的0**, CF的内容为最后移入位的值。



- 例:

MOV BL, 0A9H

SHR BL, 1

指令执行前: BL = 0A9H ; 1010 1001

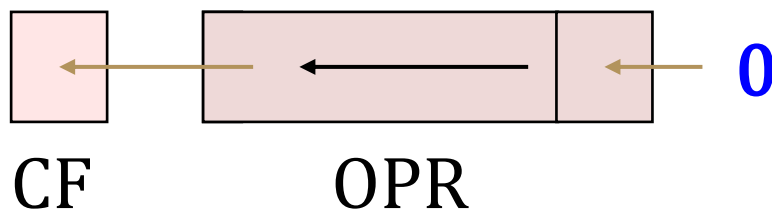
指令执行后: ; **0**1010 100**1**

BL = 54H CF = 1 OF = 1

计算: $0A9H / 2^1 = ?$

③ 算术左移SAL(shift arithmetic left)

- 格式: **SAL** **OPR, CNT**
- 执行的操作: 与SHL相同



- 例:

```
mov     cl,2
```

```
mov     al,0f0h      ;或mov     al,-16d
```

```
sal     al,cl
```

指令执行前: (al) = 0f0h ;1111 0000

指令执行后: ~~1~~11 0000 00

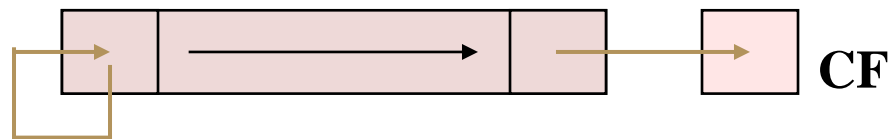
al = 0c0h CF = 1

计算: $-16 \times 2^2 = -64$ $(-64)_{\text{补}} = 0c0h$

④ 算术右移SAR(shift arithmetic right)

- 格式: **SAR** **OPR, CNT**

操作: 将(OPR)向右移动CNT指定的次数, **空出的高位补入相应个数的符号位**, CF的内容为最后移入位的值。



- 例3.65:

```
MOV     CL, 5
SAR     [DI], CL
```

指令执行前:

(DS)=0F800H (DI)=180AH (0F980A)=0064H

指令执行后:

0000 0000 0110 0100
00000 0000 0000 0110 0100

则(0F980A) = 0003H, CF=0

计算: $100D / 2^5 = ?$

- 算术移位运算适用于带符号数运算
 - SAL用来乘以 2^n （ n 指移位次数）
 - SAR用来除以 2^n
- 逻辑移位运算适用于无符号数运算
 - SHL用来乘以 2^n
 - SHR用来除以 2^n

- 例1: $(BX) = 84F0H$

(1) (BX) 为无符号数, 求 $(BX) / 2$

SHR BX, 1 ; $(BX) = 4278H$

(2) (BX) 为带符号数, 求 $(BX) \times 2$

SAL BX, 1 ; $(BX) = 09E0H, OF=1$

(3) (BX) 为带符号数, 求 $(BX) / 4$

MOV CL, 2

SAR BX, CL ; $(BX) = 0E13CH$

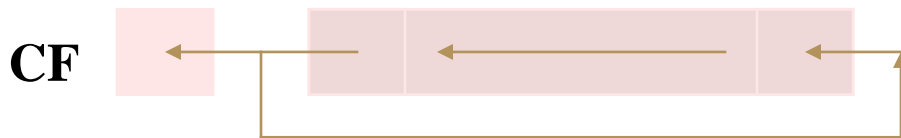
(2) 循环移位指令

- ROL
- ROR
- RCL
- RCR

① 循环左移指令 ROL(rotate left)

- 格式: **ROL** **OPR, CNT**

操作: 将操作数向左移动CNT指定的位数, **移出**
的位不仅要进入CF, 而且还要填补空出的位



- 例3.67:

如(A_X)=0012H, (B_X)=0034H, 要求把它们装配在一起形成(A_X)=1234H。

MOV CL, 8

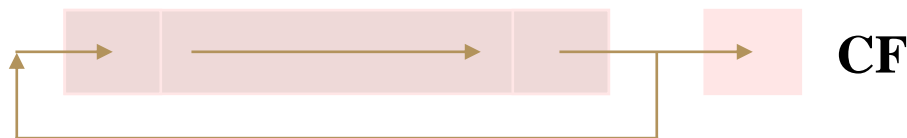
ROL AX, CL

ADD AX, BX

② 循环右移指令ROR(rotate right)

- 格式: **ROR** **OPR, CNT**

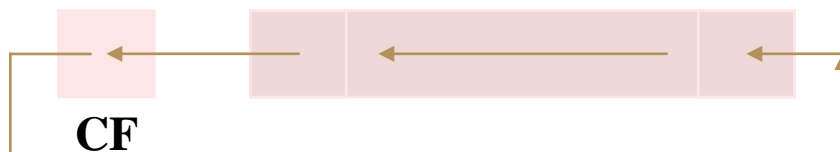
操作: 将操作数向右移动CNT指定的位数, 移出的位不仅要进入CF, 而且还要填补空出的位



③ 带进位循环左移指令 RCL

- 格式: **RCL** **OPR, CNT**

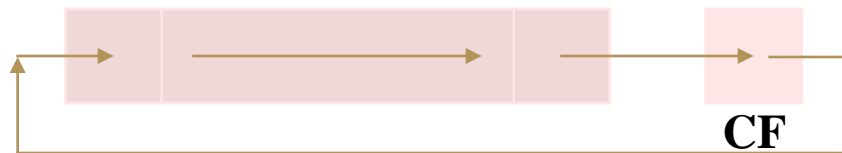
操作: 将操作数向左移动CNT指定的位数, 用原CF的值填补空出的位, 移出的位再进入CF



④ 带进位循环右移指令 RCR

- 格式: **RCR** **OPR, CNT**

操作: 将操作数向右移动CNT指定的位数, 用原CF的值填补空出的位, 移出的位再进入CF

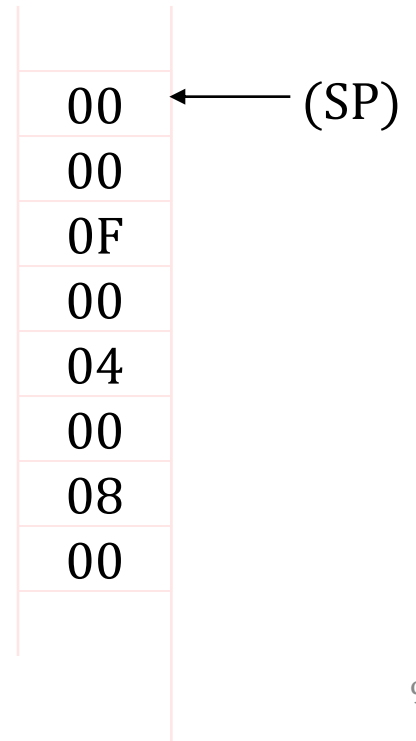


- 例：(BX)=84F0H，把 (BX) 中的 16 位数每 4 位压入堆栈

```
MOV CH, 4      ; 循环次数
MOV CL, 4      ; 移位次数
```

NEXT:

```
ROL BX, CL
MOV AX, BX
AND AX, 0FH
PUSH AX
DEC CH
JNZ NEXT
```



3.3.4 串处理指令

- MOVS 串传送($M \rightarrow M$)
- STOS 存入串($AC \rightarrow M$)
- LODS 从串取($M \rightarrow AC$)
- CMPS 串比较($M-M$)
- SCAS 串扫描($AC-M$)

- 数据传送指令(如mov、in、out)的特点
 - 数据传送指令每次只传送一个数据
 - 数据传送指令不能直接在两个存储单元间传送数据

寄存器

- 源串地址： DS: SI
- 目的串地址： ES: DI
- 累加器： AL AX

方向标志

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF			SF	ZF		AF		PF		CF

- $DF = 1$: 每次操作后使 SI 和 DI 减小
- $DF = 0$: 每次操作后使 SI 和 DI 增大
- 建立方向标志的两条指令:
 - CLD: 该指令使 $DF = 0$
 - STD: 该指令使 $DF = 1$

MOVS(move string) 串传送

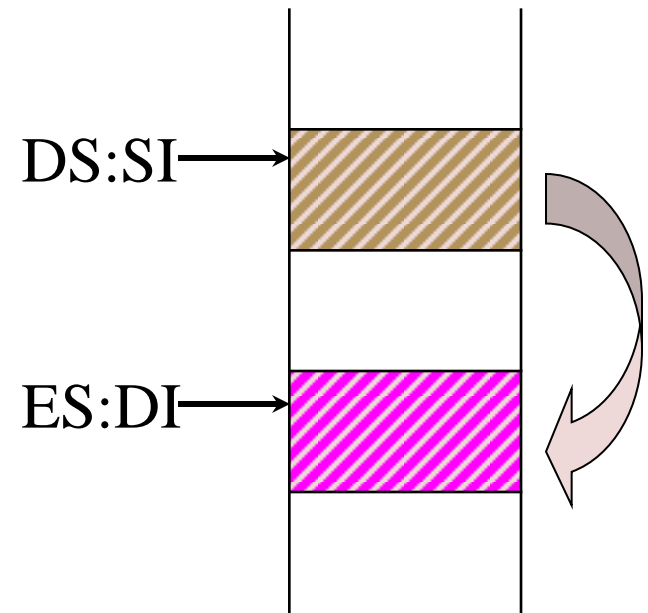
隐含操作数: $(ES:DI) \leftarrow (DS:SI)$

- 格式1: **MOVSB**; 字节操作

操作: $((DI)) \leftarrow ((SI))$
 $(SI) \leftarrow (SI) \pm 1$
 $(DI) \leftarrow (DI) \pm 1$

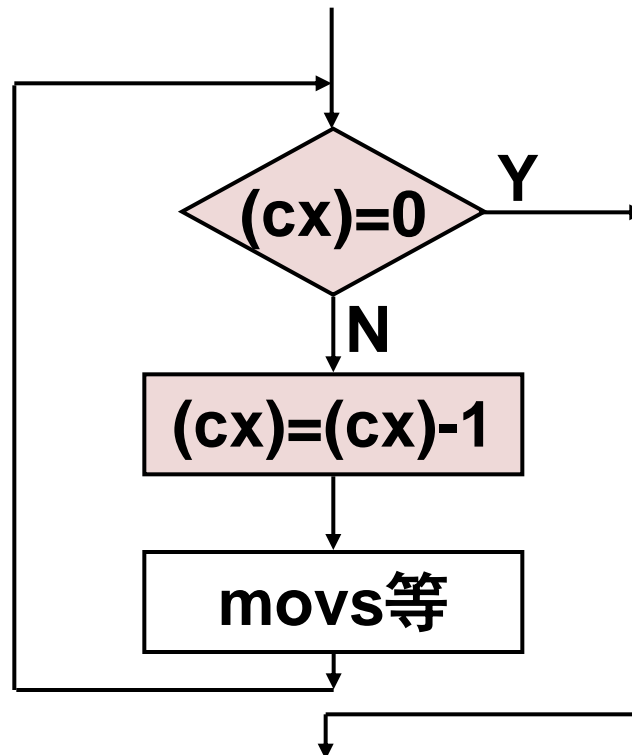
- 格式2: **MOVSW** ; 字操作

操作: $((DI)) \leftarrow ((SI))$
 $(SI) \leftarrow (SI) \pm 2$
 $(DI) \leftarrow (DI) \pm 2$



REP(repeat) 前缀

- REP重复串操作直到计数寄存器 Count Reg (CX)的内容为0为止
- 格式: **REP string primitive**
其中string primitive 可为 MOVs , STOS , LODS , INS , OUTS
- 执行的操作:



串处理过程

- (1) 串处理准备工作
 - 把存放在数据段中的源串首地址（或末地址）放入SI中
 - 把将要存放在附加段中的目的串首地址（或末地址）放入DI中
 - 建立方向标志
 - 把要处理的数据串长度放入计数寄存器CX
- (2) 串处理

p77 例3.70

在数据段中有一字符串，其长度为17，要求把它们传送到附加段中的一个缓冲区中

```
data segment
    mess1 db    'personal_computer'
data ends
```

```
extra segment
    mess2 db    17 dup (?)
extra ends
```

```
code segment
    assume      cs:code,ds:data,es:extra
```

```
start:
    mov  ax,data
    mov  ds,ax
    mov  ax,extra
    mov  es,ax
    lea  si,mess1
    lea  di,mess2
    mov  cx,17
    cld
    rep  movsb
    mov  ax,4c00h
    int  21h
code    ends
end     start
```

2. STOS(store in to string) 存入串

隐含操作数: $(ES:DI) \leftarrow (AC)$

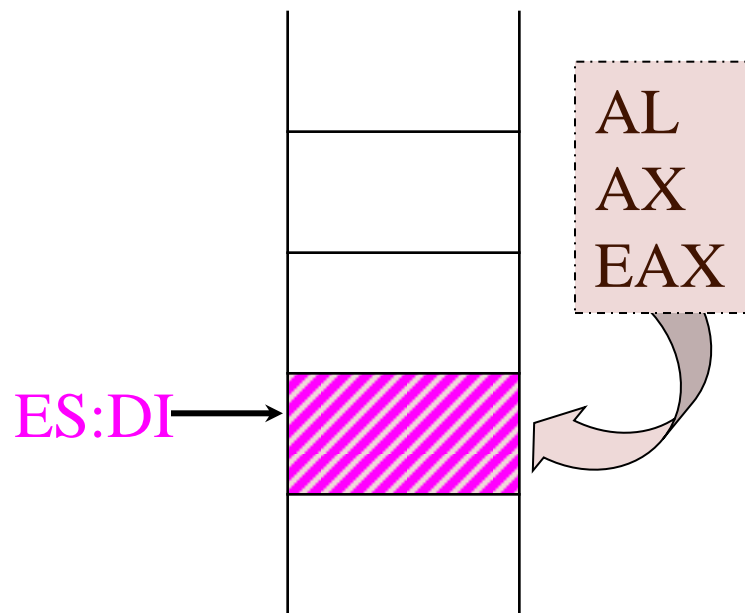
- 格式1: **STOSB** ; 字节

操作: $((DI)) \leftarrow (AL)$
 $(DI) \leftarrow (DI) \pm 1$

- 格式2: **STOSW** ; 字

操作: $((DI)) \leftarrow (AX)$
 $(DI) \leftarrow (DI) \pm 2$

- 常用于初始化某一缓冲区



3. LODS(load from string) 从串取

隐含操作数: $(AC) \leftarrow (DS:SI)$

- 格式1: LODSB ; 字节

操作: $(AL) \leftarrow ((SI))$

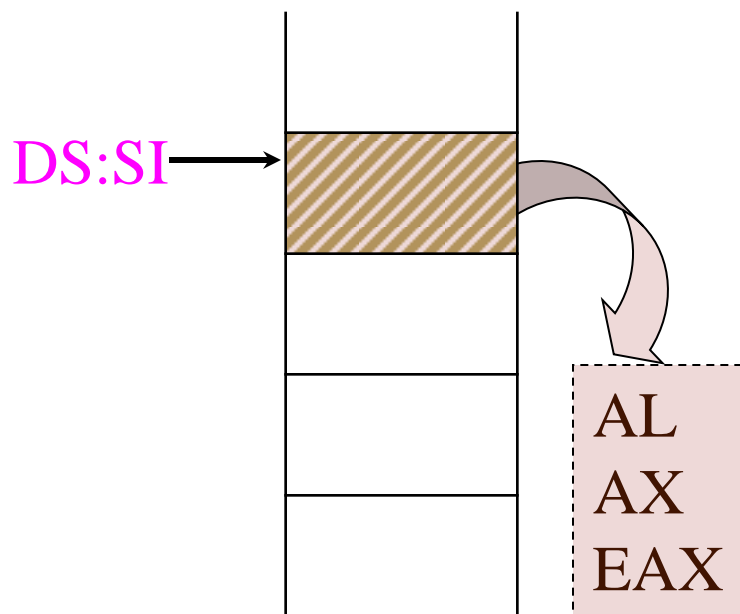
$(SI) \leftarrow (SI) \pm 1$

- 格式2: LODSW ; 字

操作: $(AX) \leftarrow ((SI))$

$(SI) \leftarrow (SI) \pm 2$

- 一般不和REP指令联用



4. CMPS(compare string) 串比较

隐含操作数: (DS:SI) - (ES:DI)

- 格式1: CMPSB ; 字节

操作: $((SI)) - ((DI))$
 $(SI) \leftarrow (SI) \pm 1$
 $(DI) \leftarrow (DI) \pm 1$

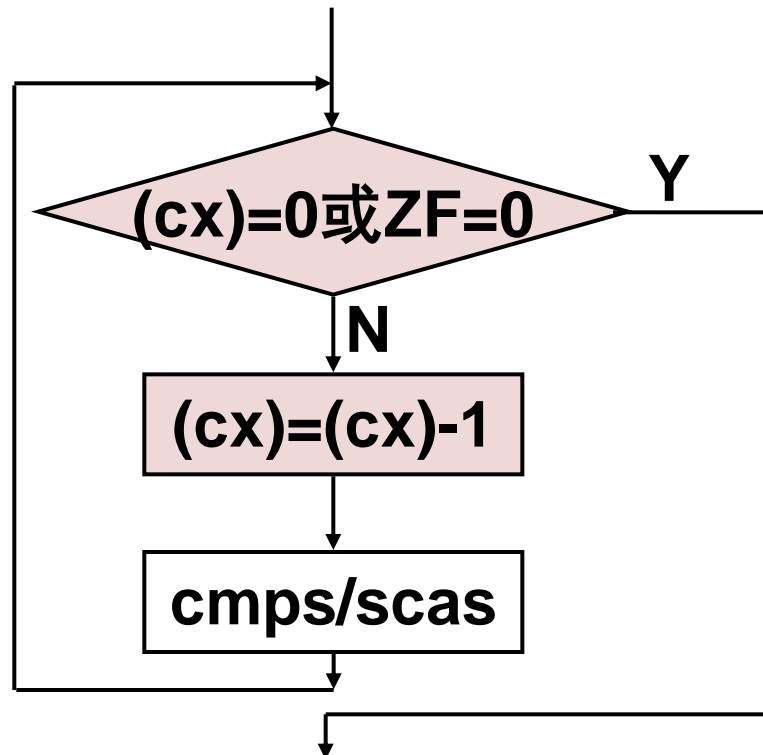
- 格式2: CMPSW ; 字

操作: $((SI)) - ((DI))$
 $(SI) \leftarrow (SI) \pm 2$
 $(DI) \leftarrow (DI) \pm 2$

- 不保存结果, 只根据结果设置条件码

REPE / REPZ 重复前缀

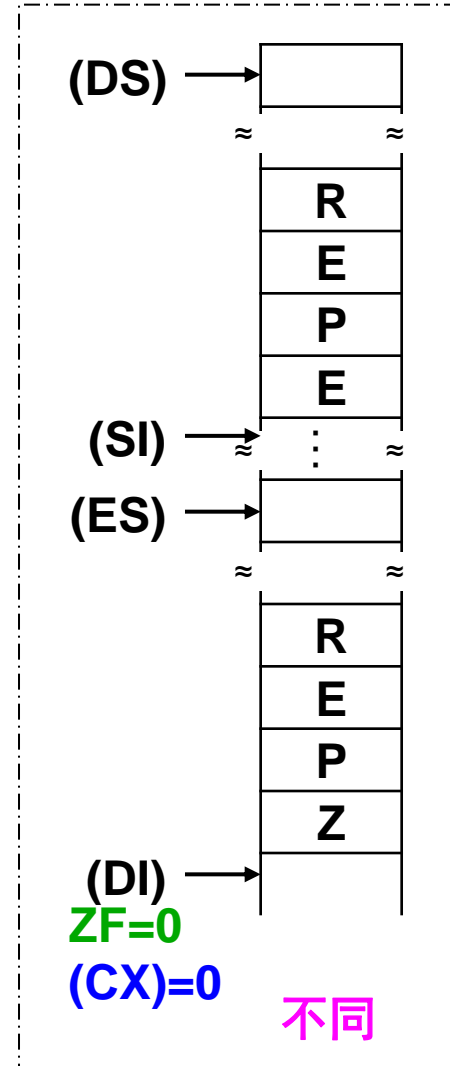
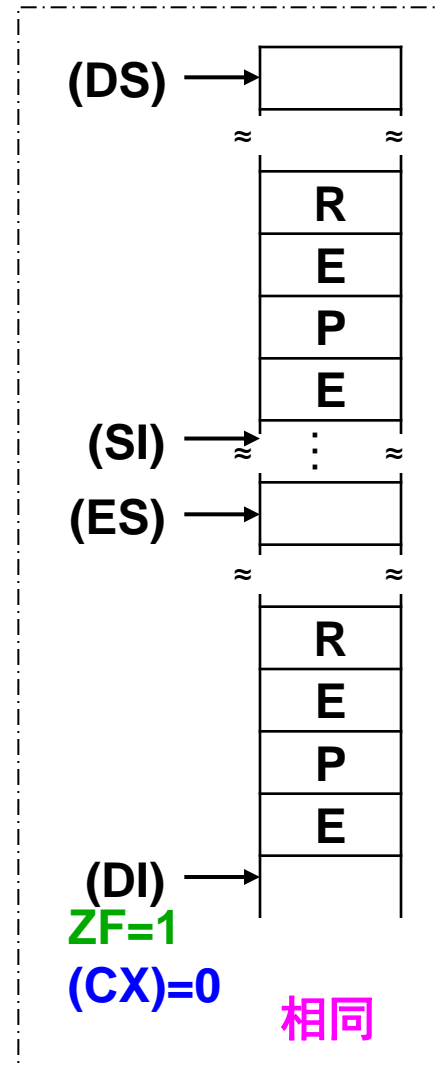
- 当相等/为零时重复串操作
- 格式: **REPE / REPZ** **String Primitive** String
Primitive 可为 CMPS 或 SCAS 指令
- 执行的操作:



例3.73 p83

- 比较两个字符串是否相同

```
lea si, mess1  
lea di, mess2  
mov cx, 8  
cld  
repe cmpsb
```



- 根据ZF判断比较的结果：

ZF=1，相同

ZF=0，不同

- 不能用CX=0判断比较的结果

例3.73 p83

- 比较两个字符串是否相同

```
lea si, mess1
```

```
lea di, mess2
```

```
mov cx, 8
```

```
cld
```

```
repe cmpsb
```

```
jz     match
```

...

match:

...

- 根据ZF判断比较的结果：

ZF=1，相同

ZF=0，不同

- 不能用CX=0判断比较的结果

5. SCAS(scan string) 串扫描

- 隐含操作数: (AC) - (ES:DI)

- 格式1: SCASB ; 字节

操作: (AL) - ((DI))
 (DI) \leftarrow (DI) \pm 1

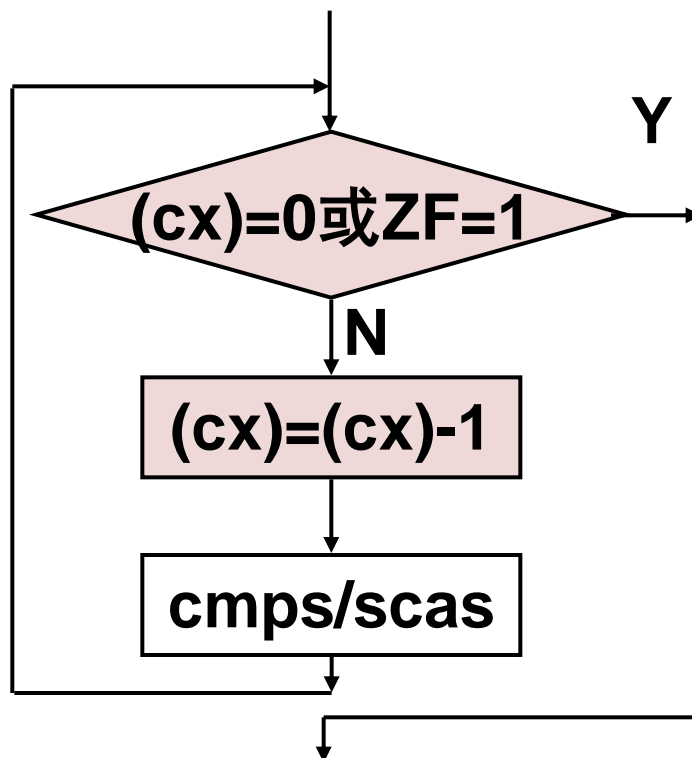
- 格式2: SCASW ; 字

操作: (AX) - ((DI))
 (DI) \leftarrow (DI) \pm 2

- 不保存结果, 根据结果设置条件码

REPNE/REPNZ 重复串操作

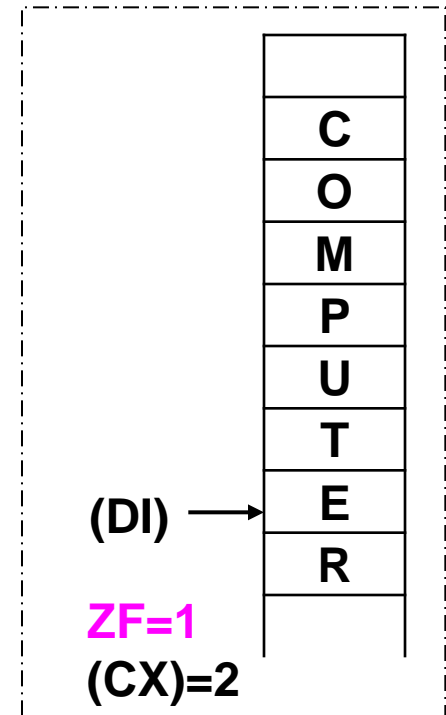
- 当不相等/不为零时重复串操作
- 格式：REPNE / REPNZ String Primitive
- 执行的操作：



例3.72 p83

- 从一个字符串中查找指定的字符
`mess db 'COMPUTER'`

```
lea di, mess
mov al, 'T'
mov cx, 8
cld
repne scasb
```



- 当执行完串搜索指令后，依据ZF的值判别是否找到：
ZF=1为找到，ZF=0，则没有找到。
- (di): 相匹配字符的下一个地址
(cx): 剩下还未比较的字符个数

3.3.5 控制转移指令

- 1. 无条件转移指令
- 2. 条件转移指令
- 3. 条件设置指令
- 4. 循环指令
- 5. 子程序调用指令
- 6. 中断指令

转移的分类

- 段内转移：在同一段的范围内进行转移，只需修改IP寄存器的内容

短转移： **SHORT** 8位位移量 $-128 \sim +127$

近转移： **NEAR PTR** 16位位移量 $-32k \sim +32k$

- 段间转移：转到另一个段去执行程序，不仅要修改IP寄存器的内容，还要修改CS寄存器的内容

远转移： **FAR PTR**

1. 无条件转移指令JMP(p85)

- (1) 段内直接短转移 **JMP SHORT OPR**
- (2) 段内直接近转移 **JMP NEAR PTR OPR**
- (3) 段内间接近转移 **JMP WORD PTR OPR**
- (4) 段间直接远转移 **JMP FAR PTR OPR**
- (5) 段间间接远转移 **JMP DWORD PTR OPR**

(1) 段内直接短转移

格式: **JMP** **SHORT OPR**

说明: **OPR**为符号地址

操作: $(IP) \leftarrow (IP) + 8\text{位位移量}$

(2) 段内直接近转移

格式: **JMP** **NEAR PTR OPR**

说明: **OPR**为符号地址

操作: $(IP) \leftarrow (IP) + 16\text{位位移量}$

(3) 段内间接近转移

格式: **JMP** **WORD PTR OPR**

说明: **OPR**为寄存器或存储器寻址

操作: $(IP) \leftarrow (EA)$

(4) 段间直接远转移

格式: **JMP FAR PTR OPR**

说明: **OPR**为符号地址

操作: $(IP) \leftarrow \text{OPR的段内偏移地址}$
 $(CS) \leftarrow \text{OPR所在段的段地址}$

(5) 段间间接远转移

格式: **JMP DWORD PTR OPR**

说明: **OPR**为存储器寻址

操作: $(IP) \leftarrow (EA)$
 $(CS) \leftarrow (EA+2)$

2 条件转移指令

- (1) 根据单个条件标志的设置情况转移
- (2) 比较两个无符号数，并根据比较的结果转移
- (3) 比较两个带符号数，并根据比较的结果转移
- (4) 测试CX或ECX的值为0则转移

概述

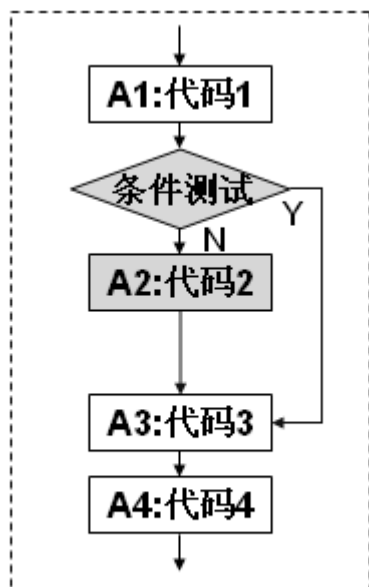
- FLAGS-标志寄存器
 - 条件码标志：记录程序中运行结果的状态信息，根据有关指令的运行结果由CPU自动设置，这些状态信息作为后续条件转移指令的控制条件
 - OF(overflow flag)-溢出标志
 - SF(sign flag)-符号标志
 - ZF(zero flag)-零标志
 - CF(carry flag)-进位标志

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF		SF	ZF		AF		PF		CF

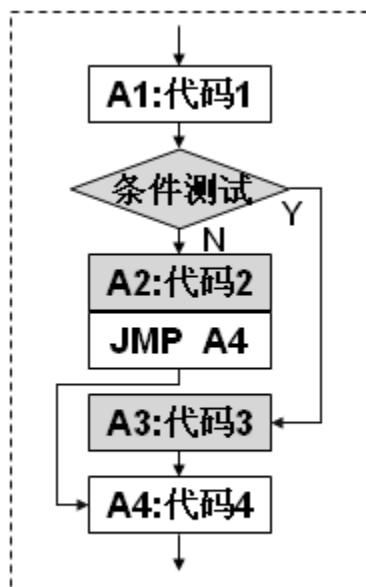
- 条件转移指令根据上一条指令所设置的条件码来判别测试条件
 - 满足测试条件则转移到由指令指定的转向地址
 - 如不满足条件则顺序执行下一条指令
- 格式： 条件转移指令 符号地址

概述

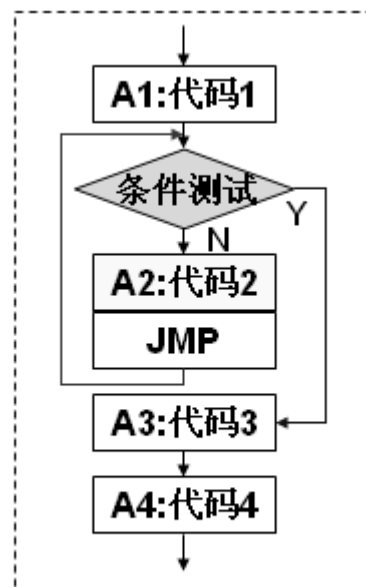
- 条件转移
 - 满足，则转移
 - 不满足，则顺序执行下一条指令



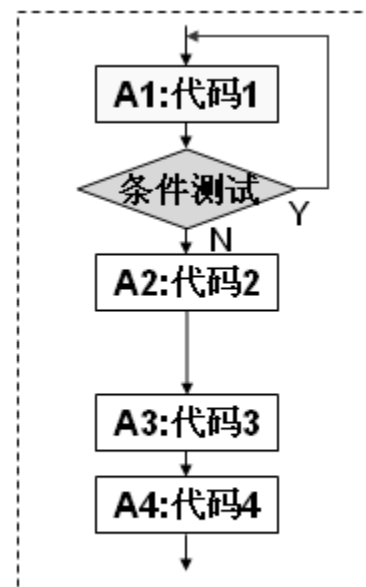
一个分支，相当于if()



两个分支，相当于if() else



循环结构



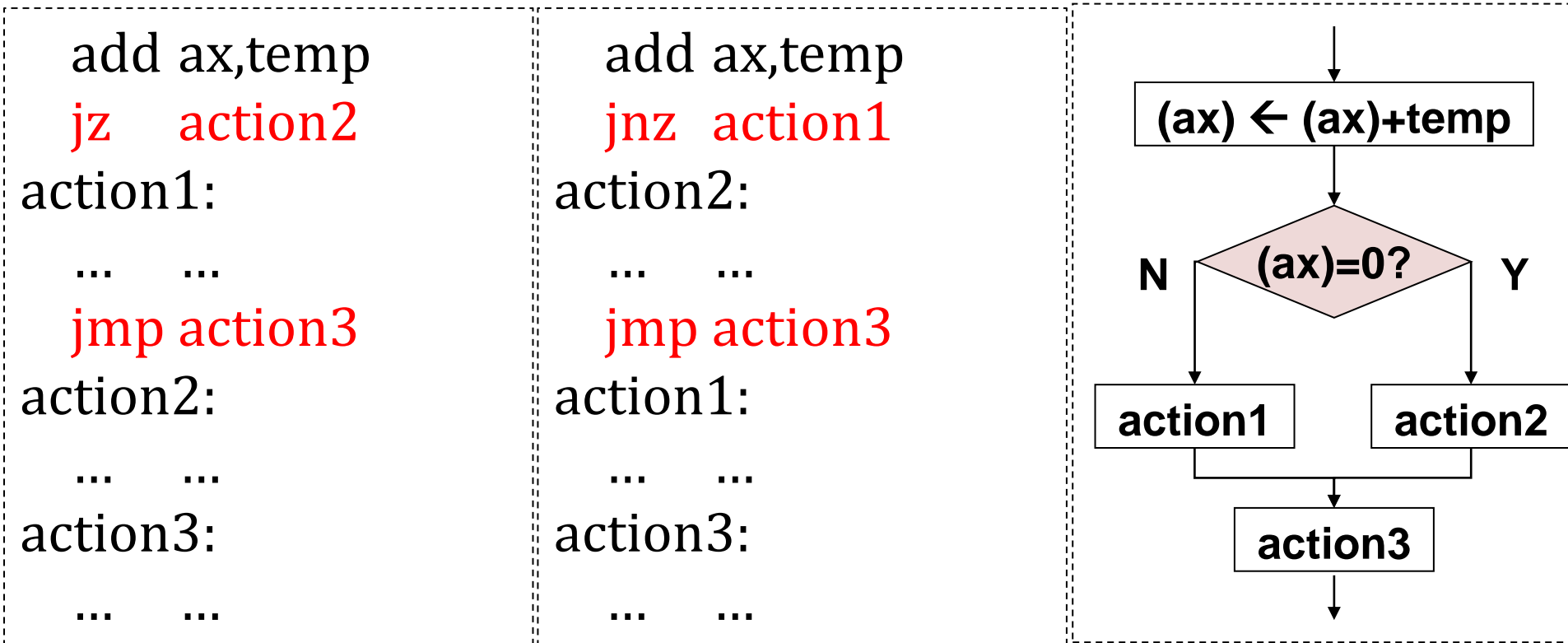
循环结构

(1)根据单个条件标志的设置情况转移

指令	功能	条件
❶ JZ / JE	结果为零（或相等）	ZF = 1
JNZ / JNE	结果不为零（或不相等）	ZF = 0
❷ JS	结果为负	SF = 1
JNS	结果为正	SF = 0
❸ JO	溢出	OF = 1
JNO	不溢出	OF = 0
❹ JC / JB / JNAE	进位，或低于	CF = 1
JNC / JNB / JAE	无进位，或不低于	CF = 0

例3.74

根据一次加法运算的结果实行不同的处理，如结果为0做动作2，否则做动作1。



(2)根据两个无符号数比较结果转移

指令	功能	条件
❶ JZ / JE	等于	ZF = 1
❷ JNZ / JNE	不等于	ZF = 0
❸ JB / JNAE / JC	低于	CF = 1
❹ JNB / JAЕ / JNC	不低于，高于或等于	CF = 0
❺ JBE / JNA	低于或等于，不高于	CF ∨ ZF = 1
❻ JNBE / JA	高于	CF ∨ ZF = 0

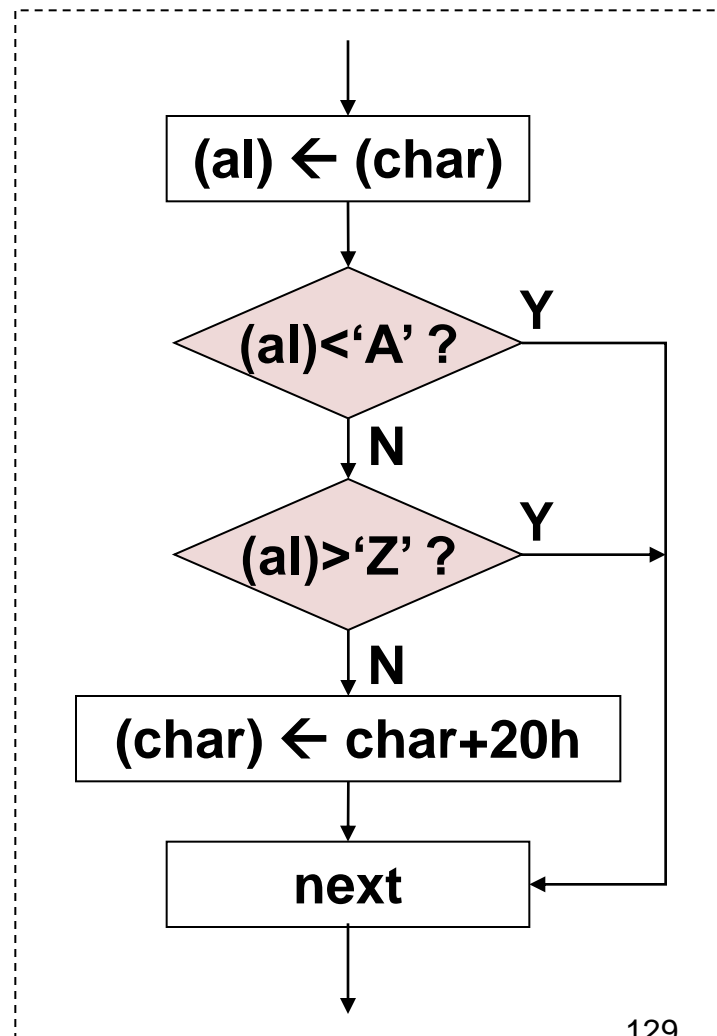
例如

已知一个字节变量char，试编写一程序段，把其所存的大写字母变成小写字母。

```
...  
char    db    'F'    ;变量说明
```

```
...  
mov     al, char  
cmp     al, 'A'  
jb      next  
cmp     al, 'Z'  
ja      next  
add     char, 20H
```

```
next:    ...
```



(3)根据两个带符号数比较结果转移

指令	功能	条件
❶JZ / JE	等于	ZF = 1
❷JNZ / JNE	不等于	ZF = 0
❸JL / JNGE	小于	(SF∨OF) = 1
❹JNL / JGE	大于等于	(SF∨OF) = 0
❺JLE / JNG	小于等于	(SF∨OF) ∨ ZF=1
❻JNLE / JG	大于	(SF∨OF) ∨ ZF=0

(4)测试计数器的值为0则转移

- JCXZ SHORT OPR;CX=0
- JECXZ SHORT OPR;ECX=0

例3.77

α 、 β 为两个双精度数，分别存储于DX、AX和BX、CX中。若 $\alpha > \beta$ 转向X执行，否则转向Y执行

CMP DX, BX

JG X ;带符号数

JL Y ;带符号数

CMP AX, CX

JA X ;无符号数

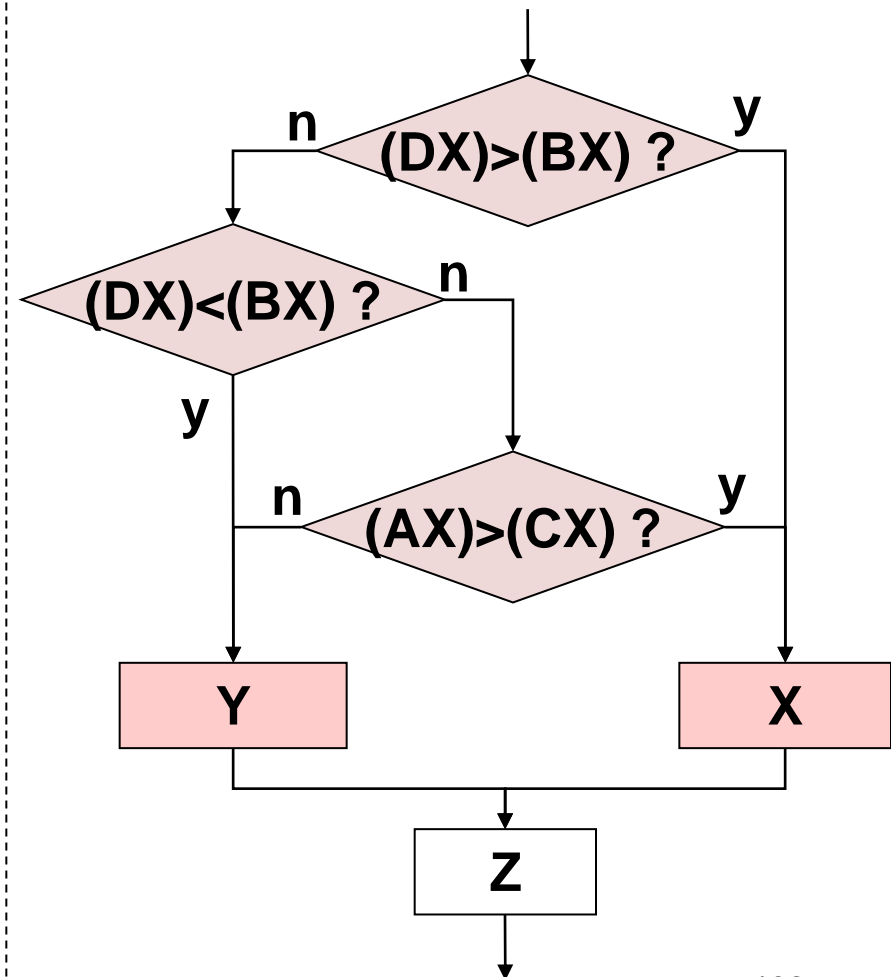
Y:

JMP Z

X:

Z:

... ..



例3.78

- 在存储器中有一个首地址为array的N数组，要求测试其中正数、0及负数的个数。正数的个数放在DI中，0的个数放在SI中，并根据 $N-(DI)-(SI)$ 求得负数的个数放在AX中。

```
data segment
    array    dw    10, 0, -3, 40, 50, 0, 70, -80, 90, 100
    n        dw    10
data ends
```

```

mov    cx,N
mov    bx,0
mov    di,bx
mov    si,bx

```

again:

```

cmp    array[bx],0
jle    less_or_eq
inc    di
jmp    next

```

less_or_eq:

```

jl     next
inc    si

```

next:

```

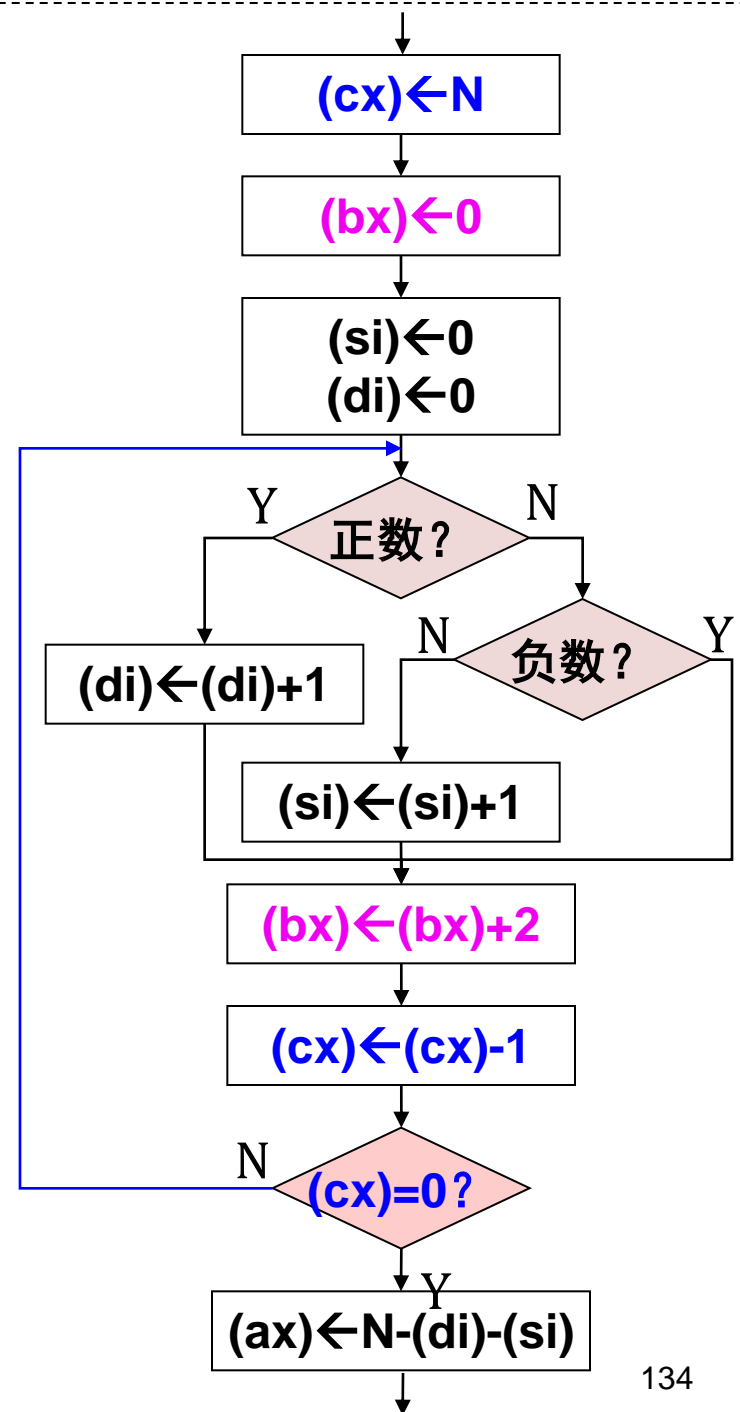
add    bx,2
dec    cx
jnz    again

```

```

mov    ax,N
sub    ax,di
sub    ax,si

```

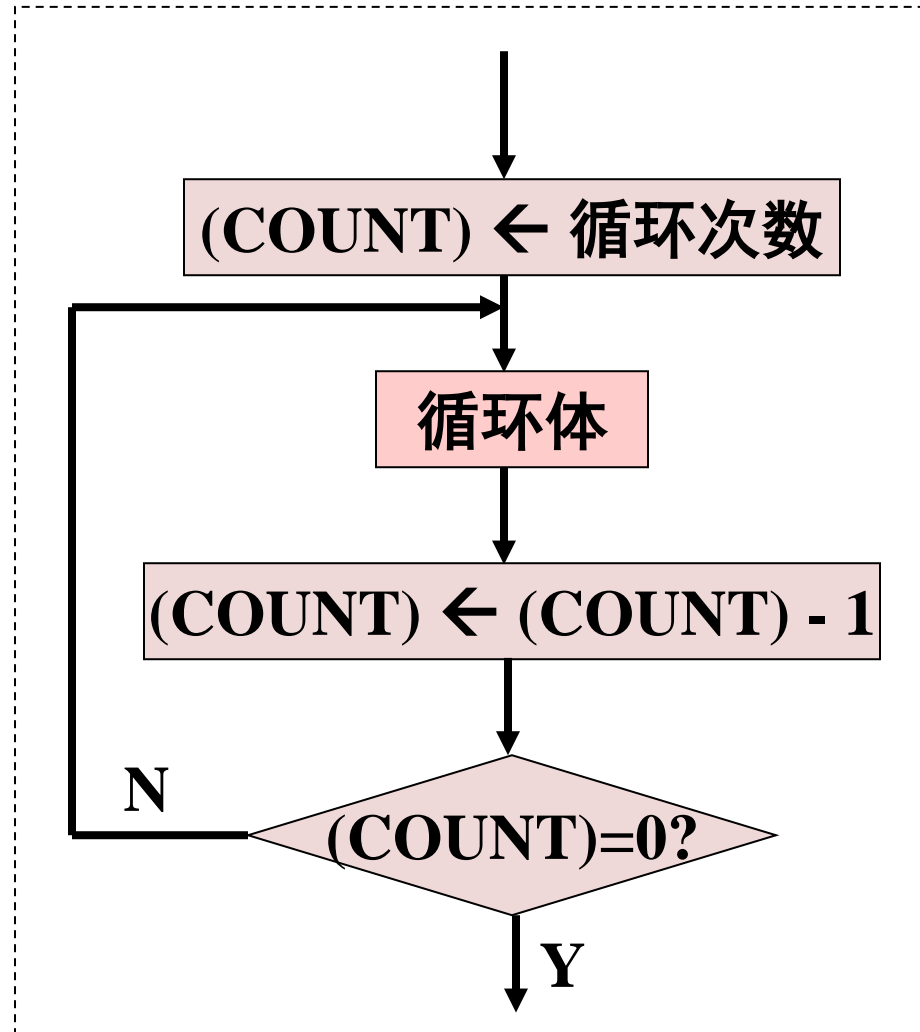


4. 循环指令

- (1) LOOP 循环
- (2) LOOPZ / LOOPE 为零或相等时循环
- (3) LOOPNZ / LOOPZE 不为零或不相等时循环

例3.79 循环结构

```
MOV CX, N
again:
...      ;循环体
...
DEC CX
JNZ again
```



循环指令

(1) **LOOP** 标号

操作: $(CX) \leftarrow (CX) - 1$
 $(CX) \neq 0$, 转到标号循环执行

(2) **LOOPZ / LOOPE** 标号

操作: $(CX) \leftarrow (CX) - 1$
若 $(CX) \neq 0$ 并且 $ZF=1$ 转到标号
若 $(CX)=0$ 或 $ZF=0$, 停止循环, 往下执行

(3) **LOOPNZ / LOOPNE** 标号

操作: $(CX) \leftarrow (CX) - 1$
若 $(CX) \neq 0$ 并且 $ZF=0$ 转到标号
若 $(CX)=0$ 或 $ZF=1$, 停止循环, 往下执行

循环结构

```
MOV CX, N
again:
... ;循环体
...
LOOP again
```

```
MOV CX, N
again:
... ;循环体
...
DEC CX
JNZ again
```

例3.80

有一个首地址为array的m字数组，求数组的内容之和（不考虑溢出），并把结果存入total中

```
data segment
```

```
    array    dw    10,20h,30,40,50h,60,70,80,90h,100
```

```
    m        dw    10
```

```
    total    dw    ?
```

```
data ends
```

例3.80

有一个首地址为array的m字数组，求数组的内容之和（不考虑溢出），并把结果存入total中

```
    mov    cx, m
    mov    ax, 0
    mov    si, ax
start_loop:
    add    ax, array[si]
    add    si, 2
    loop   start_loop
    mov    total, ax
```

例3.81

- 有一串len个字符的字符串存储于首地址为ascii_str的存储区中。要求在字符串中查找“空格”（20H），找到继续执行，未找到则转到not_found执行。

```
data segment
    ascii_str    db    'hello world!'
    len          dw    $-ascii_str
data ends
```

例3.81

```
mov  cx , len
```

```
mov  si , -1
```

```
mov  al , 20h
```

```
next:
```

```
inc  si
```

```
cmp  al , ascii_str[si]
```

```
loopne      next
```

```
jnz  not_found
```

```
... ..
```

```
not_found:
```

```
... ..
```

两种可能性:

找到: **ZF=1**

未找到: **(CX)=0 ZF=0**

DOS功能调用

- (1) 输入字符
- (2) 输出字符
- (3) 输入字符串
- (4) 输出字符串

DOS功能调用

- DOS功能调用是DOS操作系统为用户提供的许多功能子程序，可以实现输入输出。
- 基本的调用方式是：
 设置调用所需的参数；
 功能号送AH寄存器；
 用INT 21H来调用。

DOS功能调用(p319)

- 功能调用01

功能：从键盘输入一个字符

入口参数：无

出口参数：键入字符的ASCII码存AL寄存器

调用：

```
MOV AH, 01
```

```
INT 21H
```

DOS功能调用(p335)

- 功能调用02

功能：将一个字符送显示器显示

入口参数：显示字符的ASCII码存DL寄存器

出口参数：无

调用：

```
MOV DL, 'A'
```

```
MOV AH, 02
```

```
INT 21H
```

-
- 输入大写字母，将其转换为小写字母并输出。

DOS功能调用(p319)

- 功能调用0A

功能：从键盘输入一个字符串

入口参数：存放输入字符串的数据区地址存DX寄存器

出口参数：字符串的长度和每个字符的ASCII码存入数据区的相应位置

— 说明

- 第1个字节：最大字符数，由用户程序给出
- 第2个字节：实际输入字符数，自动填入
- 第3个字节开始：字符串按字节存入缓冲区，最后结束字符串的回车0DH也要占用1个字节

— 例如：

- 数据段定义字符串缓冲区如下：

```
maxlen db 16
```

```
actlen db ?
```

```
string db 16 dup(?)
```

- 输入字符串的指令如下：

```
lea dx, maxlen
```

```
mov ah, 0ah
```

```
int 21h
```

- 键入如下字符:123456789abcdef✓

- 缓冲区各字节存储情况：

maxlen

回车

10	0f	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46	0d
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

actlen

string

DOS功能调用(p335)

- 功能调用09

功能：将一个字符串送显示器显示

入口参数：显示字符串的首地址存入DX寄存器

出口参数：无

– 例如：

– 数据段定义字符串如下：

```
string    db    'hello,world!$'
```

– 输出字符串的指令如下：

```
lea       dx, string
```

```
mov       ah, 09h
```

```
int       21h
```