

- **8086CPU的转移指令：**
 - 无条件转移指令（如：**jmp**）
 - 条件转移指令
 - 循环指令（如：**loop**）
 - 过程
 - 中断

无条件跳转 jmp指令

- **jmp**为无条件转移，可只修改**IP**，也可同时修改**CS**和**IP**；
- **jmp**指令要给出两种信息：
 - 转移的**目的地址**
 - 转移的**距离**
 - 段间转移
 - 段内转移
 - 段内短转移
 - 段内近转移

- **jmp short 标号**

转到标号处执行指令

指令实现段内短转移，对**IP**的修改范围为**-128~127**，也就是说，它向前转移时可以最多越过**128**个字节，向后转移可以最多越过**127**个字节

程序执行后，**ax**中的值为多少？

程序

```
assume cs:codesg
codesg segment
start:mov ax,0
      jmp short s
      add ax,1
      s:inc ax
codesg ends
end start
```

汇编指令与机器码的对应示例

| 汇编指令 | 机器指令 |
|------------------|-------------|
| mov ax,0123 | B8 23 01 |
| mov ax,ds:[0123] | A1 23 01 |
| push ds:[0123] | FF 36 23 01 |

汇编指令中的**idata**（立即数），数据或内存单元偏移地址，都会在对应的机器指令中出现，因为**CPU**执行的是机器指令，它必须要处理这些数据或地址

程序

```
assume cs:codesg
```

```
codesg segment
```

```
start:mov ax,0
```

```
    jmp short s
```

```
    add ax,1
```

```
    s:inc ax
```

```
codesg ends
```

```
end start
```

Debug 将 jmp short s 中的 s 表示为 inc ax 指令的偏移地址 8, 并将 jmp short s 表示为 jmp 0008, 表示转移到 cs:0008 处

-u

00BD:0000 B80000

MOV

AX,0000

00BD:0003 EB03

JMP

0008

00BD:0005 050100

ADD

AX,0001

00BD:0008 40

INC

AX

```

-u
0BB0:0000 B80000      MOV     AX,0000
0BB0:0003 EB03      JMP     0008
0BB0:0005 050100      ADD     AX,0001
0BB0:0008 40          INC     AX

```

- **jmp 0008 (Debug)**
jmp short s (汇编) 所对应的机器码为 **EB 03**，不包含转移目的地址
- CPU根据什么进行转移呢？
- 汇编指令**jmp short s**中，转移的目的地址（由标号 s 表示），机器指令后目的地址没了？
- CPU如何知道转移到哪里？

```

assume cs:codesg
codesg segment
start:mov ax,0
      mov bx,0
      jmp short s
      add ax,1
      s:inc ax
codesg ends
end start

```

```

assume cs:codesg
codesg segment
start:mov ax,0
      jmp short s
      add ax,1
      s:inc ax
codesg ends
end start

```

```

-u
0BBD:0000 B80000      MOV     AX,0000
0BBD:0003 BB0000      MOV     BX,0000
0BBD:0006 EB03        JMP     000B
0BBD:0008 050100      ADD     AX,0001
0BBD:000A 40         INC     AX

```

```

-u
0BBD:0000 B80000      MOV     AX,0000
0BBD:0003 EB03        JMP     0008
0BBD:0005 050100      ADD     AX,0001
0BBD:0008 40         INC     AX

```


依据位移进行转移的jmp指令

- CPU在执行jmp指令时并不需要转移的目的地址
- 在机器指令中不包含转移的目的地址
- CPU执行指令的过程：
 - (1) 从CS:IP指向内存单元读取指令，读取的指令进入指令缓冲区；
 - (2) $(IP) = (IP) + \text{所读取指令的长度}$ ，从而指向下一条指令；
 - (3) 执行指令。转到1，重复这个过程。
- jmp short s指令的读取和执行过程

-u

```
0BBD:0000 B80000      MOV     AX,0000
0BBD:0003 BB0000      MOV     BX,0000
0BBD:0006 EB03        JMP     000B
0BBD:0008 050100      ADD     AX,0001
0BBD:000B 40          INC     AX
```

- **jmp short s**指令的读取和执行过程：
 - (1) (CS)=0BBDH, (IP)=0006, CS:IP指向EB 03 (**jmp short s**的机器码)；
 - (2) 读取指令码EB 03进入指令缓冲器；
 - (3) (IP)=(IP)+所读取指令的长度=(IP)+2=0008, CS:IP指向**add ax,1**；
 - (4) CPU执行指令缓冲器中的指令EB 03；
 - (5) 指令EB 03执行后, (IP)=000BH, CS:IP指向**inc ax**。

依据位移进行转移的jmp指令

- CPU 将指令EB 03 读入后，IP 指向了下一条指令，即 CS:0008 处的add ax,1，接着执行EB 03
- 如果 EB 03 没有对 IP 进行修改的话，那么，接下来 CPU将执行 add ax,1
- 由于CPU 执行的 EB 03是一条修改IP的转移指令，执行后 (IP) = 000BH，CS:IP指向inc ax，CS:0008处的add ax,1没有被执行
- CPU在执行EB 03的时候是根据什么修改的IP，使其指向目标指令呢？根据指令码中的03

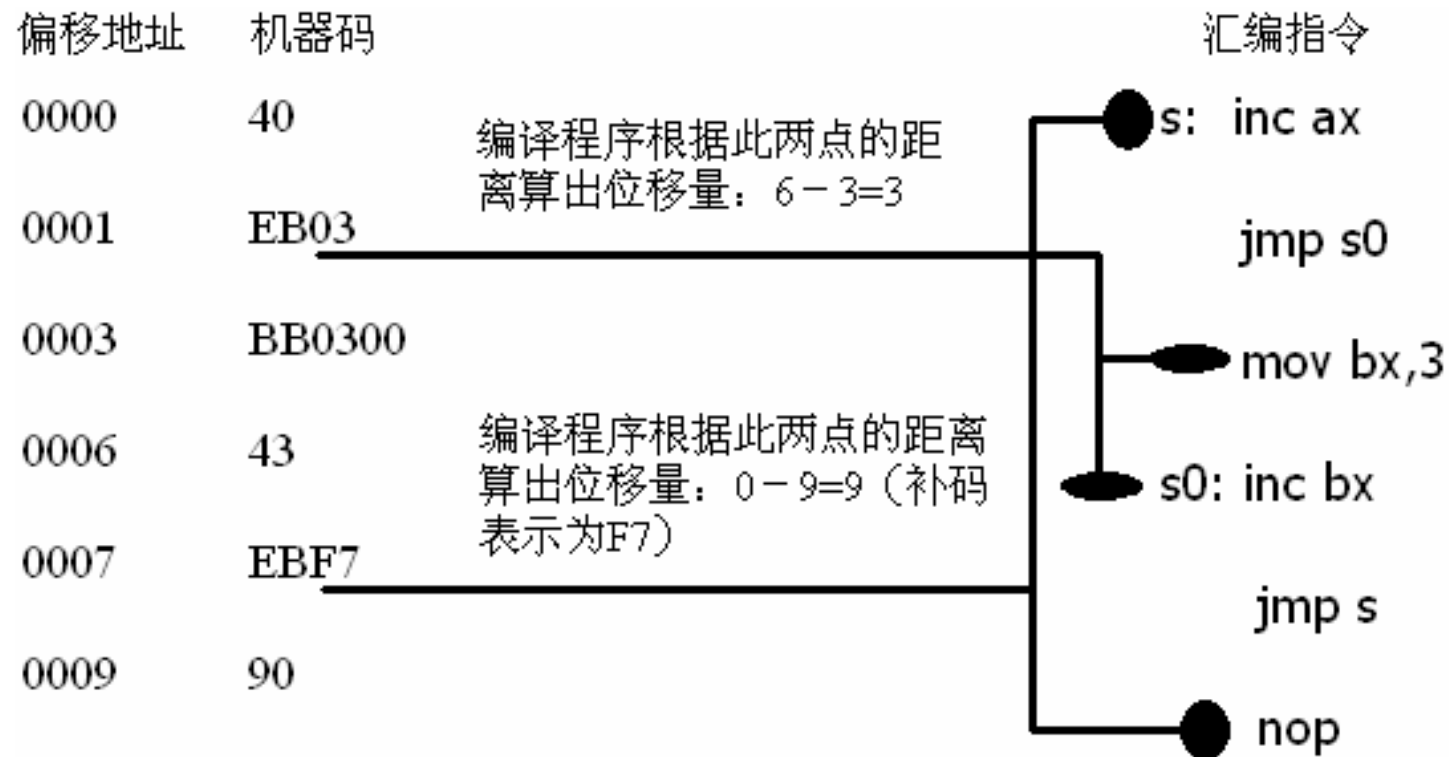
依据位移进行转移的jmp指令

- 要转移的目的地址是**CS:000B**，而CPU执行 **EB 03**时，当前的(IP)=**0008**，如果将当前的IP值加3，使(IP)=**000BH**，**CS:IP**就可以指向目标指令
- 在转移指令**EB 03**中并没有告诉CPU要转移的目的地址，但告诉了 CPU 要转移的位移，即将当前的IP向后移动3个字节

依据位移进行转移的jmp指令

- 因为程序1、2中的**jmp** 指令转移的位移相同，都是向后 3 个字节，所以它们的机器码都是**EB 03**。
- 原来如此，在“**jmp short 标号**”指令所对应的**机器码**中，并不包含转移的目的地址，而**包含的是转移的位移**。
- 这个位移，使编译器根据汇编指令中的“**标号**”计算出来的，

转移位移具体的计算方法

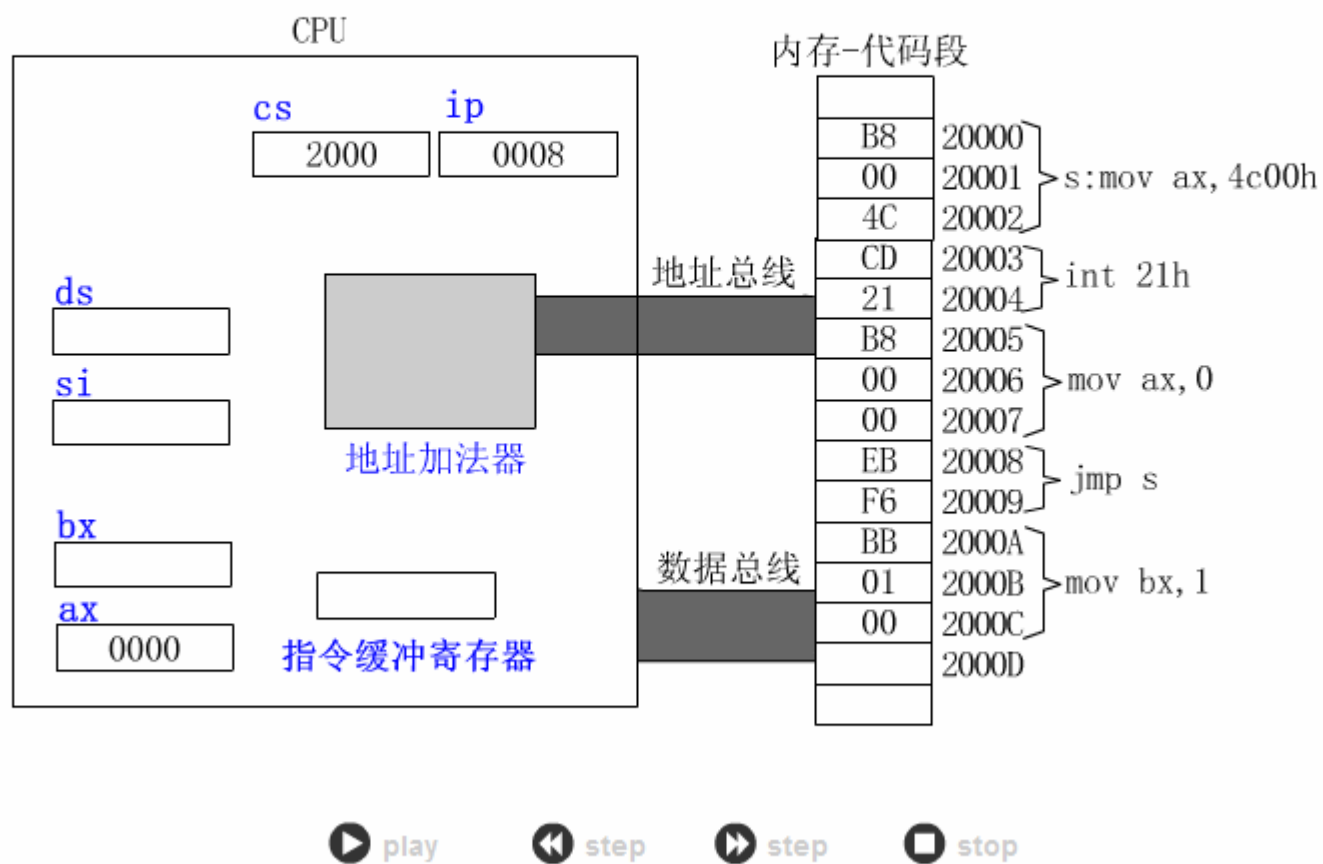


转移位移的计算方法

依据位移进行转移的jmp指令

- 结论：
CPU执行 **jmp short 标号** 指令时并不需要转移的目的地址，只需要知道转移的位移就行了。
- 具体过程[演示](#)

依据位移进行转移的 jmp 指令



依据位移进行转移的jmp指令

- **jmp short 标号**
- 功能: $(IP) = (IP) + 8\text{位位移}$
 - (1) 8位位移=“标号”处的地址-jmp指令后的第一个字节的地址;
 - (2) short指明此处的位移为8位位移;
 - (3) 8位位移的范围为-128~127, 用补码表示
 - (4) 8位位移由编译程序在编译时算出。

段内直接近转移

- **jmp near ptr 标号**
- 功能: $(IP) = (IP) + 16\text{位位移}$, 实现段内近转移
 - (1) 16位位移 = “标号”处的地址 - jmp指令后的第一个字节的地址
 - (2) near ptr指明此处的位移为16位位移, 进行的是段内近转移
 - (3) 16位位移的范围为-32769~32767, 用补码表示
 - (4) 16位位移由编译程序在编译时算出

段间直接远转移

- **jmp far ptr 标号**
- 实现段间转移，又称为远转移
- 功能如下：
 - (CS)=标号所在段的段地址；
 - (IP)=标号所在段中的偏移地址。
 - **far ptr**指明了指令用标号的段地址和偏移地址修改CS和IP。
- 程序实例

Dup(P129)

- **Dup:** 用于和**db**、**dw**、**dd** 等数据定义伪指令配合使用进行数据的重复
- **dup**的使用格式如下:
 - **db** 重复的次数 **dup** （重复的字节型数据）
 - **dw** 重复的次数 **dup** （重复的字型数据）
 - **dd** 重复的次数 **dup** （重复的双字数据）

Dup(P129)

- 用例

- **db 3 dup (0)**

- 定义了3个字节，它们的值都是0，相当于 **db 0,0,0**

- **db 3 dup (0,1,2)**

- 定义了9个字节，它们是 0、1、2、0、1、2、0、1、2，
相当于 **db 0,1,2,0,1,2,0,1,2**

- **db 3 dup ('abc','ABC')**

- 定义了18个字节，它们是 'abcABcAbcABcAbcABC'，
相当于 **db 'abcABcAbcABcAbcABC'**

- **stack segment**

- db 200 dup (0)**

- stack ends**

```

assume cs:codesg
codesg segment
    start:mov ax,0
           mov bx,0
           jmp far ptr s
           db 256 dup (0)
    s: add ax,1
       inc ax
codesg ends
end start

```

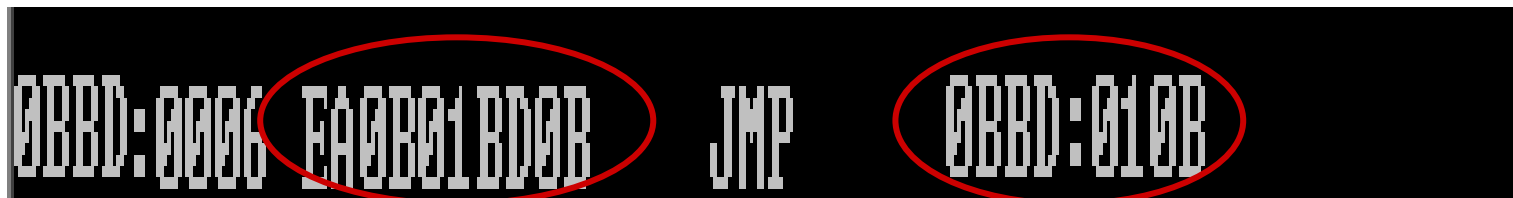
源程序

Debug翻译成机器码

```

-u
0BB0:0000 B8000000 MOV     AX,0000
0BB0:0003 BB000000 MOV     BX,0000
0BB0:0006 EAB01B00 JMP     0B0B
0BB0:000B 00000000 ADD     [BX+SI],AL
0BB0:000D 00000000 ADD     [BX+SI],AL
0BB0:000F 00000000 ADD     [BX+SI],AL
0BB0:0011 00000000 ADD     [BX+SI],AL

```



0BBD:0006 EA0B01BD0B JMP 0BBD:010B

- **db 256 dup (0)**，被Debug解释为相应的若干条汇编指令
- **jmp far ptr s**所对应的机器码：**EA 0B 01 BD 0B**，其中包含转移的目的地址“**0B 01 BD 0B**”，是目的地址在指令中的存储顺序，高地址“**BD 0B**”是转移的段地址：**0BBDH**，低地址“**0B 01**”是偏移地址：**010BH**

- 指令格式: **jmp 16位寄存器**
 - 功能: **IP = (16位寄存器)**
 - 用寄存器中的指修改**IP**
 - 等价于: **mov IP, ax**
- 例如:
(DS)=2000H,(BX)=1256H,(SI)=528FH
 - **jmp bx** 指令执行后, **(IP)=1256H**
- (参考P45)

转移地址在内存中的jmp指令

- 两种格式:
 - **jmp word ptr 内存单元地址**（段内转移）
功能：从内存单元地址处开始存放着一个字，是转移的目的偏移地址
 - **jmp dword ptr 内存单元地址**（段间转移）
功能：从内存单元地址处开始存放着两个字，高地址处的字是转移的目的段地址，低地址处是转移的目的偏移地址。
 $(CS)=(内存单元地址+2)$
 $(IP)=(内存单元地址)$

- 示例:

```
mov ax,0123H  
mov ds:[0],ax  
jmp word ptr ds:[0]
```

执行后, (IP)=0123H

| | |
|---|----|
| 0 | 23 |
| 1 | 01 |
| 2 | |
| 3 | |

```
mov ax,0123H  
mov [bx],ax  
jmp word ptr [bx]
```

执行后, (IP)=0123H

内存单元地址可用寻址方式的任一格式给出

- 示例:

```
mov ax,0123H
mov ds:[0],ax
mov word ptr ds:[2],0
jmp dword ptr ds:[0]
```

(CS)=0
(IP)=0123H
CS:IP指向0000:0123

```
mov ax,0123H
mov [bx],ax
mov word ptr [bx+2],0
jmp dword ptr [bx]
```

| | |
|-------------|-----------|
| DS:0 | 23 |
| 1 | 01 |
| 2 | 00 |
| 3 | 00 |

(CS)=0
(IP)=0123H
CS:IP指向0000:0123

内存单元地址可用寻址方式的任一格式给出

jcxz指令 (P91)

- 有条件转移指令，所有的有条件转移指令都是短转移，在机器码中包含转移的位移，而不是目的地址。对IP的修改范围都为-128~127。
- 指令格式：
 jcxz 标号
 （如果(c**x**)=0，则转移到标号处执行）
- 功能相当于：
 if ((cx**)==0) jmp short** 标号
- 操作

jcxz 标号 指令操作

- 当 $(\text{cx})=0$ 时， $(\text{IP})=(\text{IP})+8\text{位位移}$
 - 8位位移=“标号”处的地址-jcxz指令后的第一个字节的地址；
 - 8位位移的范围为-128~127，用补码表示；
 - 8位位移由编译程序在编译时算出。
 - 当 $(\text{cx})=0$ 时，什么也不做（程序向下执行）

loop指令

- 循环指令，所有的循环指令都是短转移，在对应的机器码中包含转移的位移，而不是目的地址。对IP的修改范围都为-128~127。
- 指令格式：
loop 标号
 $((\text{cx})) = (\text{cx}) - 1$
如果 $(\text{cx}) \neq 0$ ，转移到标号处执行
- 功能相当于：
 $(\text{cx})--;$
 if $((\text{cx}) \neq 0)$ jmp short 标号

loop 标号 指令操作

- (1) $(\text{cx}) = (\text{cx}) - 1$;
- (2) 如果 $(\text{cx}) \neq 0$, $(\text{IP}) = (\text{IP}) + 8$ 位移。
 - 8位移 = “标号”处的地址 - loop指令后的第一个字节的地址;
 - 8位移的范围为 -128~127, 用补码表示;
 - 8位移由编译程序在编译时算出。
- 当 $(\text{cx}) = 0$, 什么也不做 (程序向下执行)

根据位移进行转移的意义

- **jmp short 标号**
jmp near ptr 标号
jcxz 标号
loop 标号

对 **IP** 的修改是根据转移目的地址和转移起始地址之间的位移来进行的。在它们对应的机器码中不包含转移的目的地址，而包含的是到目的地址的位移。

- 这样设计，方便了程序段在内存中的浮动装配。

根据位移进行转移的意义

- 例如:

| 汇编指令 | 机器代码 |
|--------------|----------|
| mov cx,6 | B9 06 00 |
| mov ax,10 | B8 10 00 |
| s: add ax,ax | 01 C0 |
| loop s | E2 FC |

- 这段程序装在内存中的不同位置都可正确执行，因为 **loop s** 在执行时只涉及到 **s** 的位移（-4，前移4个字节，补码表示为**FCH**），而不是**s**的地址。

根据位移进行转移的意义

- 如果loop s的机器码中包含的是s的地址，则就对程序段在内存中的偏移地址有了严格的限制；
- 因为机器码中包含的是s的地址，如果s处的指令不在目的地址处，程序的执行就会出错。

根据位移进行转移的意义

- **loop s**的机器码中包含的是转移的位移，无论 s处的指令的实际地址是多少，**loop**指令的转移位移是不变的
- 编译器将报错
 - 在源程序中出现了转移范围超界
 - 源程序中使用“**jmp 2000:0100**” 转移指令
 - **Debug** 中使用的汇编指令，汇编编译器并不认识

编译器对转移位移超界的检测

```
assume cs:code  
code segment  
start: jmp short s  
       db 128 dup(0)  
       s: mov ax,0ffffh  
code ends  
end start
```

引起编译错误

jmp short s的转移范围是
-128~127，**IP**最多向后移动**127**
个字节

编译器对转移位移超界的检测

用Debug查看内存,结果如下:

2000:1000 BE 00 06 00 00 00

则此时, CPU执行指令:

```
mov ax, 2000H
```

```
mov es, ax
```

```
jmp dword ptr es:[1000H]
```

后, (CS)=? **0006H** (IP)=? **00BEH**

条件转移指令

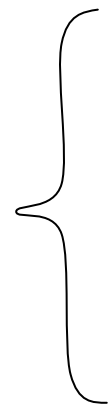
注意： 只能使用段内直接寻址的8位位移量（-128~127）

根据单个条件标志的设置情况转移：

| 格式 | 测试条件 |
|----------------|----------------|
| { JZ(JE) OPR | ZF=1 |
| { JNZ(JNE) OPR | ZF=0 |
| { JS OPR | SF=1 |
| { JNS OPR | SF=0 |
| { JO OPR | OF=1 |
| { JNO OPR | OF=0 |
| { JP OPR | PF=1 |
| { JNP OPR | PF=0 |
| JC OPR | CF=1 (JB、JNAE) |
| JNC OPR | CF=0 (JNB、JAE) |

根据多个标志状态条件判定转移

适用与数值比较，即进行关系运算

 无符号数比较: **A,B**

有符号数比较: **L,G**

比较两个无符号数，并根据比较结果转移：*

| | 格式 | 测试条件 |
|---|-------------------------|----------------|
| < | JB(JNAE,JC) OPR | CF=1 |
| ≥ | JNB(JAE,JNC) OPR | CF=0 |
| ≤ | JBE(JNA) OPR | CF∨ZF=1 |
| > | JNBE(JA) OPR | CF∨ZF=0 |

* 适用于地址或双精度数低位字的比较

比较两个带符号数，并根据比较结果转移： *

| | 格式 | 测试条件 |
|--------|---------------------|---|
| < | JL(JNGE) OPR | SF\veeOF=1 |
| \geq | JNL(JGE) OPR | SF\veeOF=0 |
| \leq | JLE(JNG) OPR | (SF\veeOF)\veeZF=1 |
| > | JNLE(JG) OPR | (SF\veeOF)\veeZF=0 |

* 适用于带符号数的比较

测试CX的值为0则转移：


| 格式 | 测试条件 |
|-----------------|---------------|
| JCXZ OPR | (CX)=0 |

例3.76 X,Y为字操作数，完成如下任务

若： $X > 50$ ，转到TOO_HIGH；

否则：计算 $X-Y$ ，如果 溢出转到OVERFLOW，
否则 $|X-Y| \rightarrow \text{RESULT}$

由题目可知，含有求绝对值的运算，故操作数应该是有符号数的概念。所以应该选用**L,G**判定系列



```
MOV AX, X  
CMP AX, 50  
JG TOO_HIGH  
SUB AX, Y  
JO OVERFLOW  
JNS NONNEG  
NEG AX  
  
NONNEG: MOV RESULT, AX  
TOO_HIGH: ...  
  
OVERFLOW: ...  
  
...
```

例3.77 α 、 β 是双精度数，分别存于DX,AX及BX,CX中， $\alpha > \beta$ 时转X，否则转Y。

CMP DX, BX

JG X

JL Y

CMP AX, CX

JA X

Y: ...

...

X: ...

...

注意高一半的比较和低一半比较的不同。
双精度有符号数的符号位在高一半，低一半只有数值。

逻辑指令（位运算）

- 逻辑运算指令
基本的逻辑运算与、或、非
异或
- 移位指令
逻辑移位、算术移位指令
循环移位指令（大、小循环）

逻辑运算指令

逻辑非指令: **NOT OPR**

执行操作: $(\text{OPR}) \leftarrow \neg (\text{OPR})$

* **OPR**不能为立即数

* 不影响标志位

逻辑与指令: **AND DST, SRC**

执行操作: $(\text{DST}) \leftarrow (\text{DST}) \wedge (\text{SRC})$

逻辑或指令: **OR DST, SRC**

执行操作: $(\text{DST}) \leftarrow (\text{DST}) \vee (\text{SRC})$

CF OF SF ZF PF AF

异或指令: **XOR DST, SRC**

执行操作: $(\text{DST}) \leftarrow (\text{DST}) \vee (\text{SRC})$

0 0 * * * 无定义

测试指令: **TEST OPR1, OPR2**

执行操作: $(\text{OPR1}) \wedge (\text{OPR2})$

例：屏蔽AL的0、1两位

AND AL, 0FCH

例：置AL的第5位为1

OR AL, 20H

例：使AL的0、1位变反

XOR AL, 3

例：测试某些位是0是1

TEST AL, 1
JZ EVEN

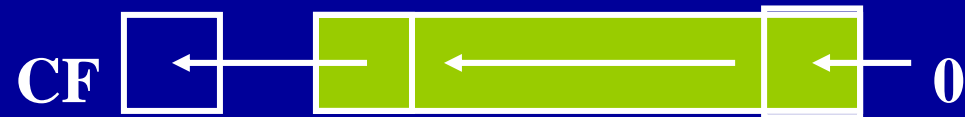
| | | | | | | | | |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| | * | * | * | * | * | * | * | * |
| AND | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| | * | * | * | * | * | * | 0 | 0 |

| | | | | | | | | |
|-----------|----------|----------|----------|----------|----------|----------|----------|----------|
| | * | * | * | * | * | * | * | * |
| OR | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | * | * | 1 | * | * | * | * | * |

| | | | | | | | | |
|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| | * | * | * | * | * | * | * | * |
| XOR | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | * | * | * | * | * | * | * | * |

移位指令:

逻辑左移 **SHL OPR, CNT** (最高位移入CF, 最低位补0)

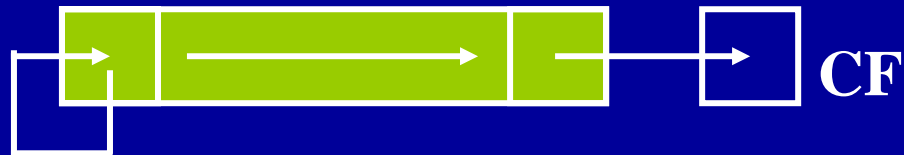


逻辑右移 **SHR OPR, CNT** (最低位移入CF, 最高位补0)



算术左移 **SAL OPR, CNT** (同逻辑左移)

算术右移 **SAR OPR, CNT** (最低位移入CF, 最高位保持不变)



示例

SHL BL,1

SHL CX,1

SHL ALFA[DI],1

或

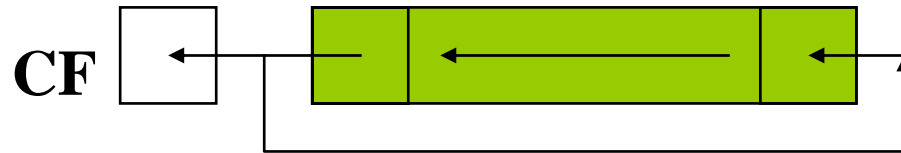
MOV CL,3

SHR DX,CL

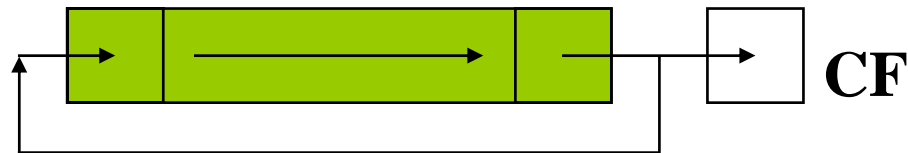
SHR DAT[DI],CL

循环移位指令：

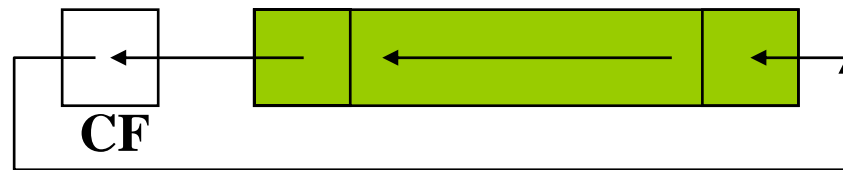
循环左移 **ROL** OPR, CNT （将目的操作数最高位和最低位连成一个环，将环中的所有位一起向左移动CNT规定的次数）



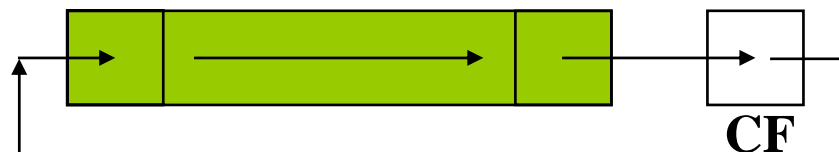
循环右移 **ROR** OPR, CNT



带进位循环左移 **RCL** OPR, CNT (将目的操作数与CF位一起)



带进位循环右移 **RCR** OPR, CNT



逻辑指令—逻辑运算指令

注意:

- * **OPR**可用立即数以外的任何寻址方式

- * **CNT=1, SHL OPR, 1**

- CNT>1, MOV CL, CNT**

- SHL OPR, CL** ; 以SHL为例

- * 条件标志位:

- CF** = 移入的数值

- OF = 1** **CNT=1**时, 最高有效位的值发生变化

- 0** **CNT=1**时, 最高有效位的值不变

- 移位指令:

- SF、ZF、PF** 根据移位结果设置

- AF**无定义

- 循环移位指令:

- 不影响 **SF、ZF、PF、AF**

逻辑指令—逻辑运算指令

例: (AX)= 0012H, (BX)= 0034H, 把它们装配成(AX)= 1234H

```
MOV CL, 8
ROL AX, CL
ADD AX, BX
```

例: (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)/2

```
SAR BX, 1 ; (BX) = 0C278H
```

(3) 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
```

| | | |
|---|------|---|
| 0 | 10B | 2 |
| 1 | 100B | 4 |

| | |
|------|--------|
| | |
| 0000 | ← (SP) |
| 000F | |
| 0004 | |
| 0008 | |