C5常用指令系统

- 8086汇编语言指令的一般格式
  - [标号:] 指令助记符 [操作数] [;注释]
    [ ]中的内容为可选项。
  - START: MOV AX，DATA ；DATA 送AX
- 标号：
  - 符号地址，表示指令在内存中的位置。标号后应加冒号
- 指令助记符：
  - 指令名称，是指令功能的英文缩写
- 操作数：
  - 指令要操作的数据或数据所在的地址。寄存器，常量，变量，表达式
- 注释：
  - 以分号“；”开头，汇编程序不处理

8086指令系统分组

- 分为5组
  - 数据传送指令
  - 算术运算指令
  - 逻辑指令与移位指令
  - 串操作指令
  - 程序转移指令

5.1 数据传送指令

- 通用数据传送指令
- 累加器专用传送指令
- 地址传送指令
- 标志寄存器传送

5.1.1 通用数据传送指令

- **MOV** 传送
- **PUSH** 进栈
- **POP** 出栈
- **XCHG** 交换

（1）MOV传送指令

- 格式: MOV DST，SRC
- 功能：将源操作数传送到目的操作数。
- 操作: (DST)←(SRC)
  - DST 表示目的操作数
  - SRC 表示源操作数。

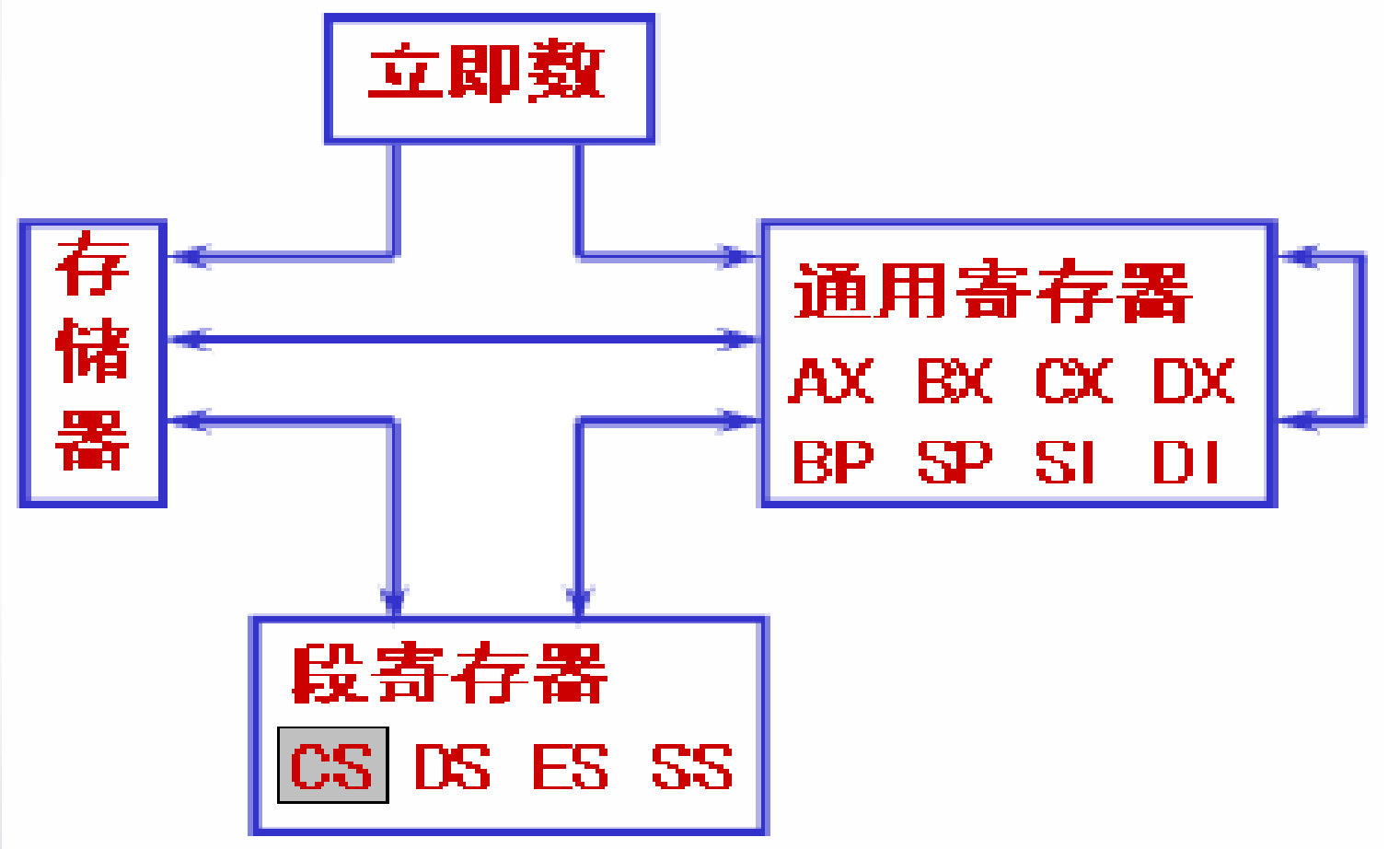
双操作数指令的规定

- 源与目的操作数的长度必须一致。
- 源与目的操作数不能同时为存储器。
- 目的操作数不能为CS和IP
  - 因为CS : IP是程序当前地址。
- 目的操作数不可以是立即数。

- 例 立即数与寄存器的传送
  - **MOV AH, 89　　；十进制数**
  - **MOV AX, 2016H　；十六进制数，后面加H**
  - **MOV AX, 0ABCDH　　；十六进制数，因非数字(0～9)开头，前面加0**
  - **MOV AL, 10001011B　　；二进制数，后面加B**
  - **MOV AL, ‘A’　　；字符‘A’的ASCII码是41H，相当于立即数**

- 以下指令是错误的
  - **MOV AH, 258**
  - **MOV AX, DH**

数据传送方向总图



- 例 在指令中说明内存单元的类型,以便操作数长度匹配。
  - **MOV [BX], AX**
- 以下指令是错误的:
  - **MOV [BX], 0**
  - 为什么?
- 指令改写为:

MOV BYTE PTR[BX], 0

MOV WORD PTR[BX], 0

- 例　段地址寄存器的传送
  - MOV AX, DATA_SEG
  - MOV DS, AX
- 段地址寄存器须通过寄存器得到段地址
  - 不能直接由符号地址、段寄存器、立即数得到。
- 以下指令是错误的:
  - MOV DS, DATA_SEG ;
  - MOV DS, ES ;
  - MOV DS, 1234 ;
  - MOV CS, AX ;

- 例　传送变量

  MOV BX, TABLE　　；假定TABLE是16位的变量

  把变量TABLE的值送给BX。

- 以下指令是错误的:

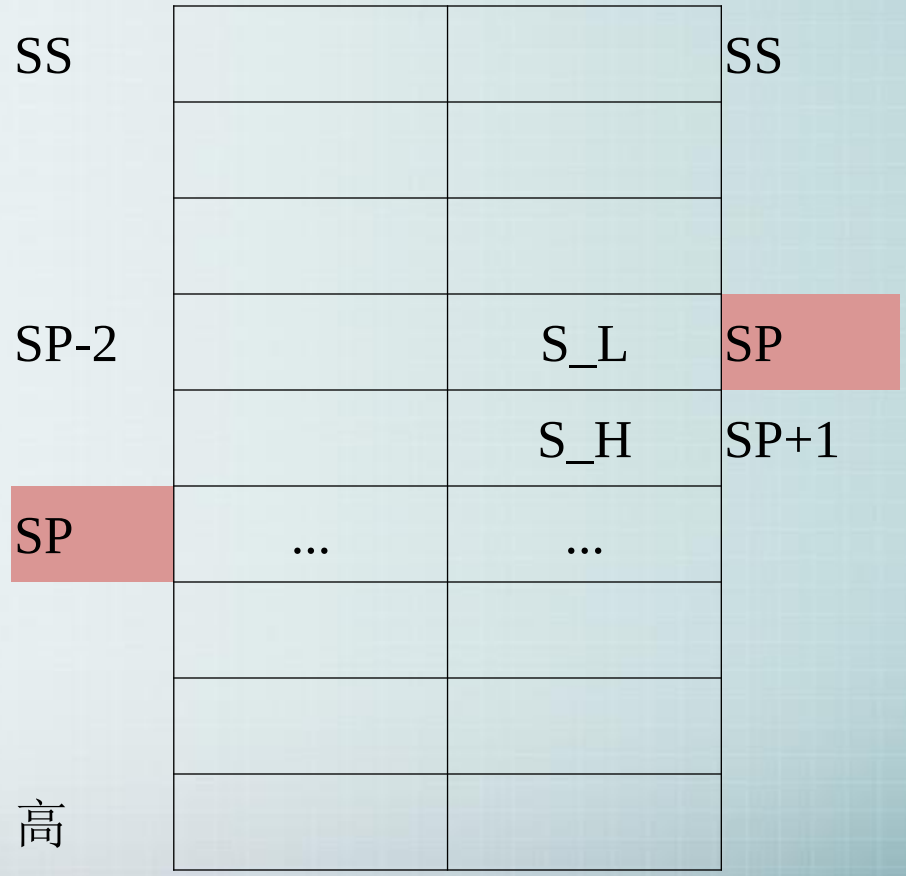
  MOV BL, TABLE　　；TABLE是16位的变量，操作数长度不一致

  MOV [BX], TABLE　　；两个操作数不能同为内存单元

- 例 传送地址
  - **MOV BX, OFFSET TABLE**
  - **OFFSET 为偏移地址属性操作符，通常是把变量TABLE的偏移地址送给BX。**
- 以下指令是错误的:
  - **MOV BL, OFFSET TABLE**
- 不管变量类型如何，其有效地址总是16位

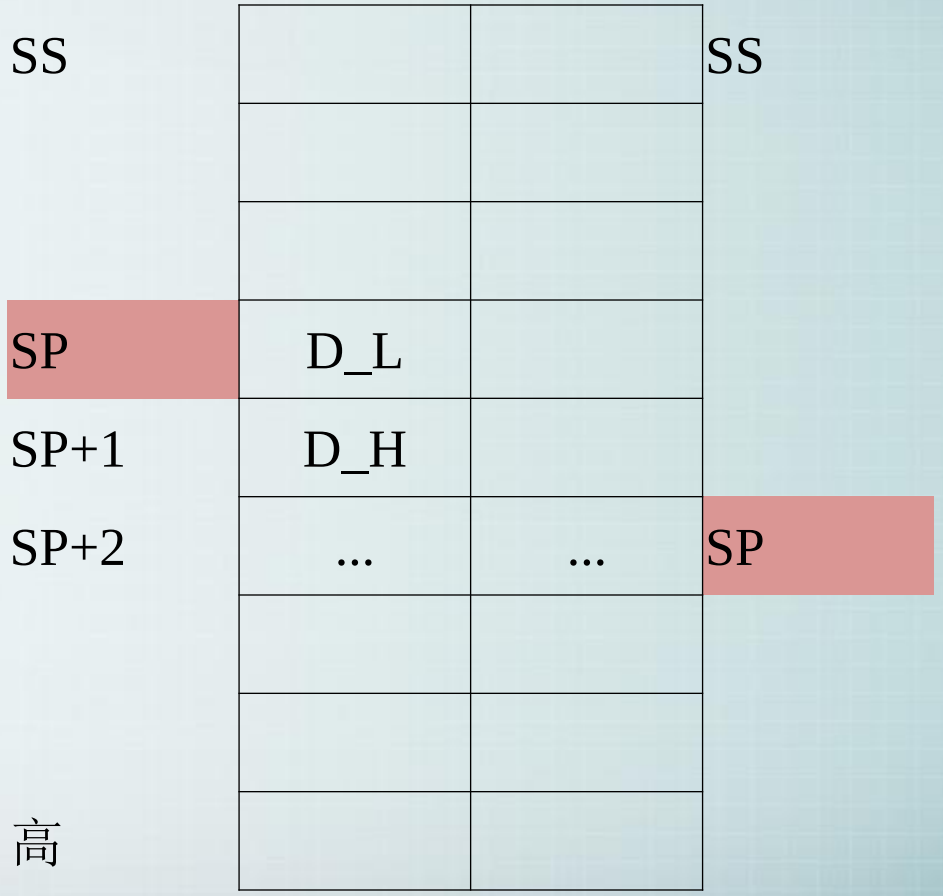
（2）PUSH 进栈指令

- 格式：
  - **PUSH　SRC**
- 操作：
  - **(SP)←(SP)-2**
  - **((SP)+1,(SP))←(SRC)**
- 堆栈：
  - 后进先出内存区
  - 以字为单位传送
  - **SS:SP总是指向栈顶**



（3）POP出栈指令

- 格式：
  - **POP　DST**
- 操作：
  - **(DST)←((SP)+1,(SP))**
  - **(SP)←(SP)+2**



❖例 进栈和出栈

MOV BX，1234H

PUSH BX

POP AX

- 例 在DEBUG下如下指令也是合法的：
  - PUSH [2016] ；把地址为DS:[2016]的字送往栈顶（SS:SP所指内存）
  - POP [2016] ；把栈顶（SS:SP所指内存）的字送往DS:[2016]的内存

（4）XCHG交换指令

- 格式：
  - XCHG　OPR1，OPR2
- 操作：
  - (OPR1) ↔ (OPR2)
- 功能：
  - 把两个操作数互换位置。
  - 遵循双操作数指令的规定，但操作数不能为立即数。

- 例
  - XCHG AX，BX ；两个寄存器长度相等
  - XCHG AX，[BX] ；AX要求[BX]也取字单元
  - XCHG AX，VAR ；VAR 必须是字变量
- 以下指令是错误的:
  - XCHG AX，5 ；显然操作数不能为立即数
  - XCHG [BX], VAR ；操作数不能同为内存单元
  - XCHG AX，BH ；操作数长度要一致

5.1.2 累加器专用传送指令

- IN ;从I/O端口输入
- OUT ;向I/O端口输出
- XLAT ;换码 （查表）

  - I/O端口是CPU 与外设传送数据的接口,单独编址,不属于内存
  - 端口地址范围: 0000~FFFFH.
  - 这组指令只限于AX, AL累加器。

（1）IN输入指令

- 长格式：
  - IN AL, PORT（字节） ;00~FFH
  - IN AX, PORT（字）

- 操作：AL←(PORT)
  AX←(PORT)

- 功能：把端口PORT的数据输入到累加器。

- 短格式：IN AL,DX（字节）；PORT放入DX

  IN AX,DX（字）

- 操作：AL←((DX))

  AX← ((DX))

- 功能：把DX指向的端口的数据输入到累加器。

❖例 读端口

IN AX， 61H

MOV BX， AX

把端口61H的16位数据输入到累加器AX，再转送BX。

- 例
  - MOV DX, 2F8H
  - IN   AL, DX
  - 把端口2F8H的8位数据输入到累加器AL。

- IN AX, 2F8H ;错，端口号超出8位，不能用长格式
- IN AX,[DX] ；错，端口地址不能用[ ]

（2）OUT输出指令

- 长格式：**OUT PORT,AL （字节） ;00-FFH**

  OUT PORT,AX （字）

- 操作：**PORT ← AL**

  PORT ← AX

- 功能：把累加器的数据输出到端口**PORT**。

- 短格式：**OUT DX,AL （字节） ;0000-FFFFH**

  OUT DX,AX （字）

- 操作： **(DX) ← AL**

  (DX) ← AX

- 功能：把累加器的数据输出到**DX**指向的端口。

- 例: 写端口

OUT 61H, AL

OUT DX, AL

（3）XLAT换码（查表）指令

- 格式：XLAT
- 操作：AL ←（BX+AL）
- 功能：把BX+AL的值作为有效地址，取出其中的一个字节送AL。

- 例: 换码
  - MOV AX,DATA
  - MOV DS,AX
  - MOV BX,100H
  - MOV AL,4
  - XLAT
  - INT 21H

- XLAT执行前

```
AX=1416  BX=0000  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=0005   NV UP EI PL NZ NA PO NC
1427:0005 BB0001        MOV     BX,0100
-d ds:0104
1416:0100                65 66 67 00-00 00 00 00 00 00 00 00       efg.........
1416:0110  B8 16 14 8E D8 BB 00 01-B0 04 D7 CD 21 B4 4C CD   ............!.L.
1416:0120  21 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   !...............
1416:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1416:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1416:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1416:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1416:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
1416:0180  00 00 00 00                                       ....
-t

AX=1416  BX=0100  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=0008   NV UP EI PL NZ NA PO NC
1427:0008 B004          MOV     AL,04
-t

AX=1404  BX=0100  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=000A   NV UP EI PL NZ NA PO NC
1427:000A D7            XLAT
```

- XLAT执行后

```
AX=1416  BX=0000  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=0005   NV UP EI PL NZ NA PO NC
1427:0005 BB0001        MOV     BX,0100
-t

AX=1416  BX=0100  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=0008   NV UP EI PL NZ NA PO NC
1427:0008 B004          MOV     AL,04
-t

AX=1404  BX=0100  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=000A   NV UP EI PL NZ NA PO NC
1427:000A D7            XLAT
-t

AX=1465  BX=0100  CX=0121  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=1416  ES=1406  SS=1416  CS=1427  IP=000B   NV UP EI PL NZ NA PO NC
1427:000B CD21          INT     21
```

5.1.3 地址传送指令

- LEA　　有效地址送寄存器
- LDS　　指针送寄存器和DS
- LES　　指针送寄存器和ES

（1）LEA有效地址送寄存器指令

- 格式：**LEA REG, SRC**
- 操作：**REG ← SRC**
- 功能：把源操作数的有效地址**EA**送到指定的寄存器。

❖例 取变量的有效地址

LEA BX，TABLE

MOV BX，OFFSET TABLE

上面2条指令等效。

TABLE无论是何类型的变量，其有效地址总是16位。

- 例

LEA　BX, [2016H]

MOV　BX,　OFFSET　[2016H]

指令执行后，**BX=？？？**

（2）LDS指针送寄存器和DS指令

- 格式： **LDS　REG, SRC**
- 操作：**REG←（SRC）**

  DS←（SRC+2）
- 功能：把源操作数**SRC**所指向的内存单元中的两个字送到指定的寄存器**REG**和**DS**。

- 例

LDS SI，[BX]

指令执行前，如 DS=2000H，BX=0400H，(2000:0400)=1234H, (2000:0402)=5678H,

指令执行后，SI=1234H，DS=5678H。

（3）LES指针送寄存器和ES指令

- 格式：
  - LES　REG, SRC
- 操作：
  - REG←（SRC）
  - ES←（SRC+2）
- 功能：
  - 把源操作数SRC所指向的内存单元中的两个字送到指定的寄存器REG和ES。
- 例
  - LES　DI,［10H］
  - DS=C000H,（C0010H)=0180H, (C0012H)=2000H
  - 结果 DI=0180H,　ES=2000H
- 与LDS的功能差别是什么？

5.1.4 标志寄存器传送指令

- **LAHF** 标志寄存器FLAGS的低字节送AH
- **SAHF** AH送FLAGS的低字节
- **PUSHF** 标志进栈
- **POPF** 标志出栈

- 除SAHF，POPF 外，以上传送类其他指令均不影响标志位

例

- **LAHF ； 标志寄存器低字节送AH寄存器**
- **SAHF ； AH送标志寄存器**
- **PUSHF ； 标志入栈**
- **POPF ； 标志出栈**

- 画出数据在数据段中的存放情况，程序执行后，BX、DI、CX、DX寄存器中的内容是什么

```
DATA  SEGMENT
ARRAY  DW  20，30，40，20H，30H，-6
BUFF   DB  ‘ABCD$’
DATA  ENDS
CODE  SEGMENT
    ASSUME  CS:CODE,DS:DATA
START:
    MOV    AX, DATA
    MOV    DS, AX
        MOV    BX, ARRAY+1
        MOV    DI, OFFSET ARRAY
        MOV    CX, [DI+5]
        MOV    DL, BUFF+3
        MOV    AH, 4CH
        INT    21H
CODE  ENDS
        END    START
```

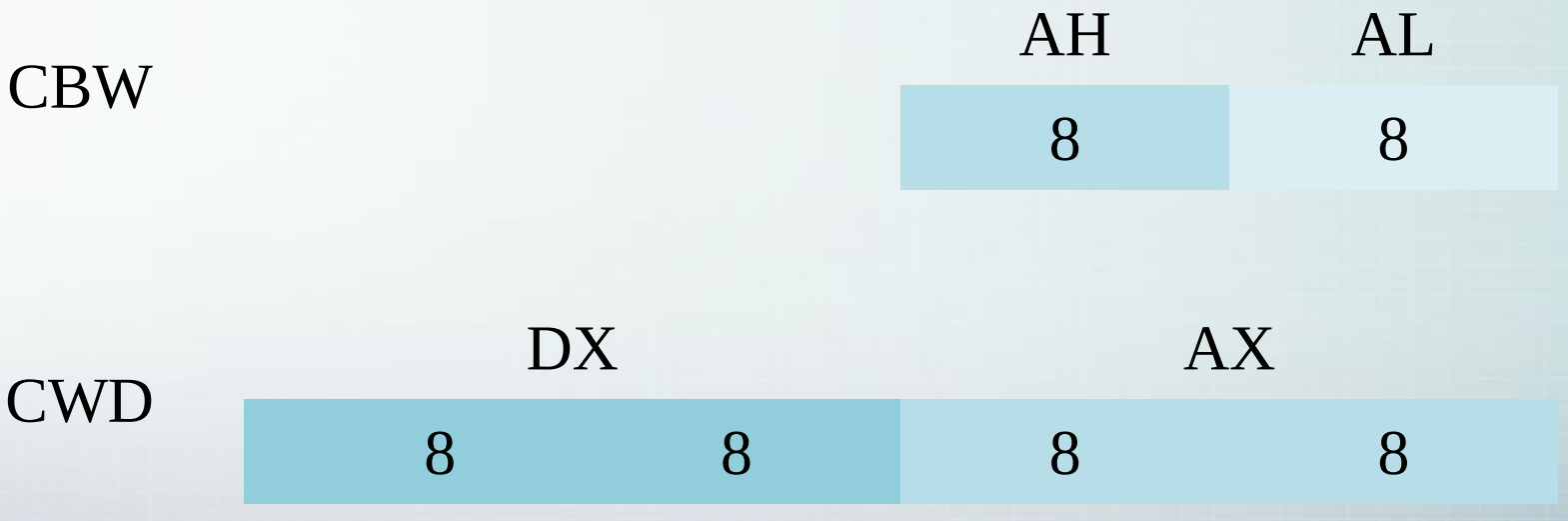
- 14 00 1E 00 28 00 20 00 30 00 FA FF 61 62 63 64 $
- (BX)=1E00H，(DI)=0000H，(CX)=2000H，(DX)=0064H

5.2 算术运算指令

- 加减乘除四则运算是计算机经常进行的基本操作。
- 算术运算指令主要实现二进制（和十进制）数据的四则运算。

5.2.1 类型扩展指令

- **CBW**：**AL 扩展为 AX**
- **CWD**：**AX 扩展为 DX，AX**
- 扩展规则：符号扩展。



例 正数的扩展

- **MOV AL, 52H ； AL中的52H是正数**
- **CBW ； 指令执行后，AX=0052H**
- **CWD ； 指令执行后，DX=0000H**
- **AX=0052H**

例 负数的扩展

- **MOV AL，88H ；AL中的88H是负数**
- **CBW ；指令执行后，AX=FF88H**
- **CWD ；指令执行后，DX=FFFFH**
- **AX=FF88H**

5.2.2 加法指令

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

（1）ADD加法指令

- 格式：
  - ADD　DST，SRC
- 操作：
  - （DST）←（DST）+（SRC）
- 例 无符号数的溢出标志位CF
  - MOV　AL，72H
  - ADD　AL，93H

- 例 有符号数的溢出标志位OF

- **MOV AL，92H**
- **ADD AL，93H**

溢出判断,以8位二进制数为例

	无符号数	有符号数	
(1).	4	4	0000 0100
	+11	11	+ 0000 1011
	15	15	0000 1111
	CF=0	OF=0	
(2)	7	7	0000 0111
	+251	− 5	+ 1111 1011
	258	2	1 0000 0010
	CF=1 错	OF=0	CF

(3)

9	+9	0000 1001
+124	+ (+124)	+ 0111 1100
133	133	1000 0101
CF=0	OF=1	负数-123 错

(4)

135	-121	1000 0111
+ 245	- 11	+ 1111 0101
380	-132	1 0111 1100
CF=1	OF=1	
+124 错	+124 错	

- 综上所述
  - CF=1 为无符号数的溢出
  - OF=1 为有符号数的溢出
- **OF位的规律：**
  - 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
- **CF位的规律：**
  - 有进位/借位时CF=1
  - 否则CF=0

（2）ADC带进位加法指令

- 格式：**ADC　DST，SRC**
- 操作：**（DST）←（DST）+（SRC）+CF**

- 例　用16位指令实现32位的双精度数的加法运算。设数A存放在目的操作数寄存器DX和AX，其中DX存放高位字。数B存放在寄存器BX和CX，其中BX存放高位字。如：
  - DX=2000H，AX=8000H
  - BX=4000H，CX=9000H
- 指令序列怎么写
  - ADD　AX, CX　　；低位字加
  - ADC　DX, BX　　；高位字加
- 执行效果是什么
  - 第一条指令执行后，AX=1000H，CF=1，OF=1，此处OF=1不必在意。
  - 第二条指令执行后，DX=6001H，CF=0，OF=0，表示结果正常，无溢出。

32位加法运算解析

例 两个双精度数相加

X: DX=0002H , AX=0F365H

Y: BX=0005H, CX=0E024H

X+Y: ADD AX, CX

ADC DX, BX 进位 CF

	第二条指令					第一条指令				
DX	0000	0000	0000	0010		1111	0011	0110	0101	AX
BX	0000	0000	0000	0101		1110	0000	0010	0100	CX
DX	0000	0000	0000	1000	1	1101	0011	1000	1001	AX
	（DX）=0008H					（AX）=D389H				

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

（3）INC 加1指令

- 格式：INC　OPR
- 操作：（OPR）←（OPR）+1

- 除INC不影响CF外，它们都影响条件标志位.
- 运算处理过程涉及的常用的条件标志位:
  - 进位 CF
  - 零 ZF
  - 符号 SF
  - 溢出 OF

5.2.3 减法指令

- **SUB** 减法
- **SBB** 带借位减法
- **DEC** 减1
- **NEG** 求补
- **CMP** 比较

（1）SUB 减法指令

- 格式：SUB DST，SRC
- 操作：（DST）←（DST）-（SRC）

（2）SBB 带借位减法指令

- 格式：**SBB DST，SRC**
- 操作：**（DST）←（DST）-（SRC）-CF**

- CF=1 为无符号数溢出, OF=1 为有符号数溢出.
- 例 考察减法中的标志位CF、OF
  - MOV AL， 72H
  - SUB AL， 93H

- 例：用16位指令实现32位的双精度数的减法运算。设数A存放在目的操作数寄存器DX和AX，其中DX存放高位字。数B存放在寄存器BX和CX，其中BX存放高位字。如：
  - DX=2001H， AX=8000H
  - BX=2000H， CX=9000H
- 指令序列：
  - SUB AX， CX ；低位字减法
  - SBB DX， BX ；高位字减法
- 执行效果
  - 第一条指令执行后，AX=F000H，CF=1，而对OF=0，ZF=0, SF=1，不必在意
  - 第二条指令执行后，DX=0000H，CF=0，OF=0，表示结果正确。ZF=1, SF=0

（3） DEC　减1指令

- 格式：**DEC　OPR**
- 操作：（OPR）←（OPR）-1

（4） NEG　求补指令

- 格式：NEG　OPR
- 操作：（OPR）←-（OPR）
- 功能：对OPR求补,求- OPR, 即反码+1.
- 只有OPR为0时，CF=0。

- 例 考察NEG指令
  - MOV AX，3
  - NEG AX
  - MOV DX，0
  - NEG DX
- 指令序列执行后，AX=FFFDH = -3 (补码)，DX=0H。
- 可以看出
  - NEG指令实际上就是求数X的相反数，即 求0-X
  - 只有当X=0时，CF=0
  - 其它情况下，CF=1

- 习题：求出7450H与以下各十六进制数的和及差，并根据结果标出SF、ZF、CF、OF标志位的值。
- (1) 1234H　　(2)5678H　　(3)9804H　　(4)E0A0H
  - SF、ZF、CF、OF
  - (1)1234H+7450H，　1　0　0　1
  - (2)5678H+7450H，　1　0　0　1
  - (3)9804H+7450H，　0　0　1　0
  - (4)E0A0H+7450H，　0　0　1　0

（5） CMP 比较指令

- 格式：**CMP OPR1, OPR2**
- 操作：（**OPR1**）-（**OPR2**）
- 不回送结果，只产生标志位。

- 例 考察CMP指令

  MOV AX， 5

  DEC AX

  CMP AX， 5

  指令序列执行后，**AX=4，ZF=0，SF=1，CF=1，OF=0**。

- **CMP**指令虽作减法，但不回送结果，只是产生标志位，为程序员比较两个数的大小提供判断依据。

5.2.4 乘法指令

- **MUL** 无符号数乘法
- **IMUL** 有符号数乘法

（1）MUL 无符号数乘法指令

- 格式：**MUL　SRC**
- 操作：

  操作数为字节时：**(AX)← (AL) X (SRC)**

  操作数为字时：**(DX,AX)← (AX) X (SRC)**

（2）IMUL 带符号数乘法指令

- 格式：**IMUL　SRC**
- 操作：

  操作数为字节时：**(AX)← (AL) X (SRC)**

  操作数为字时：**(DX,AX)← (AX) X (SRC)**

- 两个相乘的数必须长度相同。
- **SRC**不能是立即数。

- 例 无符号数和带符号数的乘法

```
MOV   AL，  0F1H
MOV   BL，  AL
MUL   BL
```

指令序列执行后，AX=E2E1H。

- 如果看成是两个带符号相乘，则应选择如下指令：

```
MOV   AL，  0F1H
MOV   BL，  AL
IMUL  BL
```

指令序列执行后，AX=00E1H。说明了两个负数相乘，结果为正数。

5.2.5 除法指令

- **DIV** 无符号数除法
- **IDIV** 有符号数除法

（1）DIV 无符号数除法指令

格式：　DIV　　SRC

(1)字节操作：　(AL) ← (AX) / (SRC) 的商
　　(AH) ← (AX) / (SRC) 的余数

(2)字操作：　(AX) ← (DX,AX) / (SRC) 的商
　　(DX) ← (DX,AX) / (SRC) 的余数

（2）IDIV 带符号数除法指令

- 格式：**IDIV　SRC**

  操作与**DIV** 相同

- 余数和被除数同符号。
- 被除数长度应为除数长度的两倍。
- **SRC**不能是立即数。

- 例　作(字节)除法300H/2H，商产生溢出
  - MOV AX，300H
  - MOV BL，2
  - DIV BL
- 此时被除数的高8位(AH=3)绝对值＞除数的绝对值2，则商会产生溢出
- 实际上换成十进制计算也可说明商会产生溢出：300H/2H=768/2=384
- 显然，8位的AL寄存器容不下商384

- 例 作(字)除法300H/2H，商不会产生溢出
  - MOV AX， 300H
  - CWD
  - MOV BX， 2
  - DIV BX
- 此时被除数的高16位(DX=0)，则商不会产生溢出。显然AX寄存器完全能容下商384。

	字	双字
CBW	AL扩展到AX	
CWD		AX扩展到DX
MUL	(AX)← (AL) X (SRC)	(DX,AX)← (AX) X (SRC)
IMUL	(AX)← (AL) X (SRC)	(DX,AX)← (AX) X (SRC)
DIV	商 AL=AX/src 余数AH=AX/src	商 AX=DX:AX/src 余数DX=DX:AX/src
IDIV	商 AL=AX/src 余数AH=AX/src	商 AX=DX:AX/src 余数DX=DX:AX/src

算术运算综合举例

❖例　算术运算综合，计算:(V－(X×Y＋Z－16))／X，其中X、Y、Z、V均为16位带符号数，在数据段定义，要求上式计算结果的商存入AX，余数存入DX寄存器。

```
data   segment
    x  dw  4
    y  dw  2
    z  dw  14H
    v  dw  18H
data   ends

code   segment
    assume  cs:code, ds:data
start:
    mov    ax, data
    mov    ds, ax
    mov    ax，x
    imul   y              ；x×y
    mov    cx，ax         ；暂存(x×y)的结果
    mov    bx，dx
    mov    ax，z
    cwd                   ；z符号扩展
    add    cx，ax         ；加z
    adc    bx，dx
    sub    cx，16         ；减16
    sbb    bx，0
    mov    ax，v
    cwd                   ；v符号扩展
    sub    ax，cx         ；v减(x×y)的结果
    sbb    dx，bx
    idiv   x
    mov    ah，4ch
    int    21h
  code   ends
end  start
```

```
D:\masm6>DEBUG 329.EXE
-U0 2E
0B78:0000 B8770B          MOV     AX,0B77
0B78:0003 8ED8            MOV     DS,AX
0B78:0005 A10000          MOV     AX,[0000]
0B78:0008 F72E0200        IMUL    WORD PTR [0002]
0B78:000C 8BC8            MOV     CX,AX
0B78:000E 8BDA            MOV     BX,DX
0B78:0010 A10400          MOV     AX,[0004]
0B78:0013 99              CWD
0B78:0014 03C8            ADD     CX,AX
0B78:0016 13DA            ADC     BX,DX
0B78:0018 83E910          SUB     CX,+10
0B78:001B 83DB00          SBB     BX,+00
0B78:001E A10600          MOV     AX,[0006]
0B78:0021 99              CWD
0B78:0022 2BC1            SUB     AX,CX
0B78:0024 1BD3            SBB     DX,BX
0B78:0026 F73E0000        IDIV    WORD PTR [0000]
0B78:002A B44C            MOV     AH,4C
0B78:002C CD21            INT     21
0B78:002E 050100          ADD     AX,0001
-
```

```
0B78:002A B44C            MOV      AH,4C
0B78:002C CD21            INT      21
0B78:002E 050100          ADD      AX,0001
-R
AX=0000  BX=0000  CX=003E  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0B67  ES=0B67  SS=0B77  CS=0B78  IP=0000   NV UP EI PL NZ NA PO NC
0B78:0000 B8770B          MOV      AX,0B77
-T

AX=0B77  BX=0000  CX=003E  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0B67  ES=0B67  SS=0B77  CS=0B78  IP=0003   NV UP EI PL NZ NA PO NC
0B78:0003 8ED8            MOV      DS,AX
-T

AX=0B77  BX=0000  CX=003E  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0B77  ES=0B67  SS=0B77  CS=0B78  IP=0005   NV UP EI PL NZ NA PO NC
0B78:0005 A10000          MOV      AX,[0000]                    DS:0000=0004
-D0 L 8
0B77:0000  04 00 02 00 14 00 18 00                              ........
-G=0 2A

AX=0003  BX=0000  CX=000C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=0B77  ES=0B67  SS=0B77  CS=0B78  IP=002A   NV UP EI PL ZR NA PE NC
0B78:002A B44C            MOV      AH,4C
-
```

BCD码的十进制调整指令

- 前面提到的所有算术运算指令都是二进制数的运算。
- 为便于十进制计算，提供了十进制调整指令
- 在二进制数计算的基础上，给予十进制调整，直接得到十进制结果。

- **BCD码 (Binary Coded Decimal)：用二进制编码表示十进制数.**
- **四位二进制数表示一位十进制数，由于四位二进制数的权分别为8，4，2，1，所以又称为8421码.**

十进制	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

- 压缩的BCD码调整指令主要有两条：

  (1) DAA　；加法十进制调整指令

  (2) DAS　；减法十进制调整指令

(1) DAA

- 格式：DAA
- 操作的语义：

IF CF=1 or AL高4位是[A～F] THEN

AL+60H.

IF AF=1 or AL低4位是[A～F] THEN

AL+6

- 例 AL=28H=28(BCD),BL=65H=65(BCD)
  - ADD AL，BL ；AL=28H+65H=8DH
  - DAA ；AL=AL+6H=8DH+6H=93H=93(BCD)
- AL和BL中都是用BCD码表示的十进制数，含义分别是28和65，ADD指令作二进制加法后得到8DH，不是BCD码
- DAA指令作用后，把和调整为93H，但它表示的是十进制数93的BCD码。

- 例　如AX=88H=88(BCD), BX=89H=89(BCD)
  - ADD AL，BL
    - AL=88H+89H=11H，AF=1，CF=1
  - DAA
    - AL=AL+66H=11H+66H=77H=77(BCD)，CF=1
  - ADC　AH, 0
    - X=177H=177(BCD)

- 第一条加法指令中的低四位加产生了向高四位的进位，这使得辅助进位AF置1，高四位加产生的进位使得进位CF置1
- DAA指令作用后，把和调整为77H，CF=1
- 最后ADC指令使AX中得到177H，即十进制数177的BCD码

例　　ADD AL, BL　　0000 1001　　9

　　DAA　　+　0000 0100　　4

　　0000 1101　　13

　　+　　0110

　　0001 0011

❖BCD码9+4的结果是（10011）BCD即13

(2) DAS

- 格式：DAS
- 操作：

IF AF=1 OR AL低4位是[A～F] THEN

AL-6

IF CF=1 OR AL高4位是[A～F] THEN

AL-60H.

❖例

如AL=93H=93(BCD),BL=65H=65(BCD)

```
SUB   AL，BL      ；AL=93H-65H=2EH
DAS               ；AL=AL-6H=2EH-
                     6H=28H=28(BCD)
```

例　　SUB AL, AH

　　DAS

AL=86H=86(BCD), AH=07H=07(BCD)

SUB 即 86H-07H=7FH

DAS 即 7FH-6H=79H=79(BCD)

- BCD调整指令一般跟在算术运算指令后
- 用于将计算结果转换成BCD码
- 使用它，意味着把算术运算指令涉及的输入数据也当作BCD码
- 用到了算术运算指令产生的AF值。（半字节对应着一个BCD码）

5.3 逻辑，移位

- AND ;与
- OR ;或
- NOT ;非
- XOR ;异或
- TEST ;测试

- SHL
- SHR
- SAL
- SAR
- ROL
- ROR
- RCL
- RCR

5.3.1 逻辑指令

- **AND** ;与
- **OR** ;或
- **NOT** ;非
- **XOR** ;异或
- **TEST** ;测试
- 按位操作，至少一个操作数是寄存器。

(1) AND 与指令

- 格式: AND DST，SRC
- 操作: (DST)←(DST)∧(SRC)

(2) OR 或指令

- 格式: OR　DST，SRC
- 操作: (DST)←(DST)∨(SRC)

(3) NOT 非指令

- 格式: NOT　OPR
- 操作: (OPR)←(OPR)

(4) XOR 异或指令

- 格式: XOR DST，SRC
- 操作: (DST)←(DST)∨(SRC)

(5) TEST 测试指令

- 格式: TEST　OPR1，OPR2
- 操作: (OPR1)∧(OPR2)
- TEST执行AND操作, 但不保存结果,只根据其特征置标志位(例如符号标志位SF)。

❖例　屏蔽AL寄存器的高四位，如AL=36H

AND　AL，0FH

指令执行的结果使AL=06H

❖例　对AL寄存器的最低两位置1，如AL=36H。

OR　AL，03H

指令执行的结果使AL=37H

- 例 对AL寄存器的最低两位取反，如AL=36H。

  XOR　AL，03H

  指令执行的结果使AL=35H
- 例　测试AL寄存器中的数，如果是负数则转到标号NEXT去执行。如AL=86H。

  TEST　AL，80H

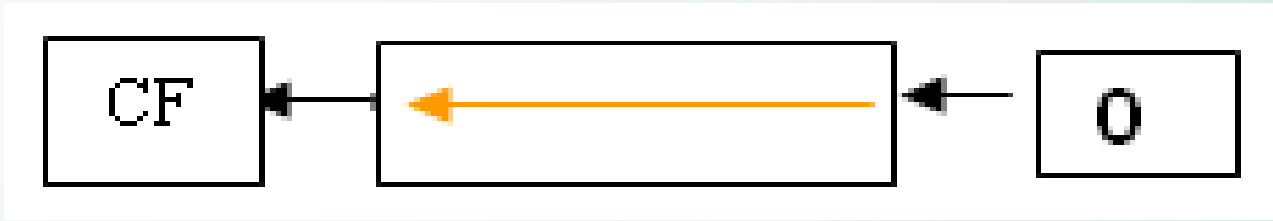
  JS　　NEXT

  指令执行的结果AL=86H(不变)，我们只要注意到FLAGS标志寄存器的SF=1，所以程序转到标号NEXT去执行。

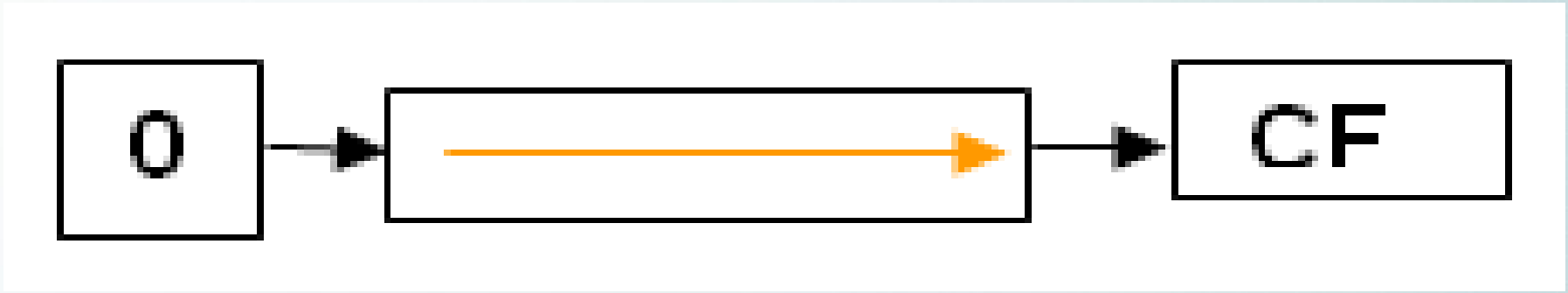
5.3.2 移位指令

- SHL
- SHR
- SAL
- SAR
- ROL
- ROR
- RCL
- RCR

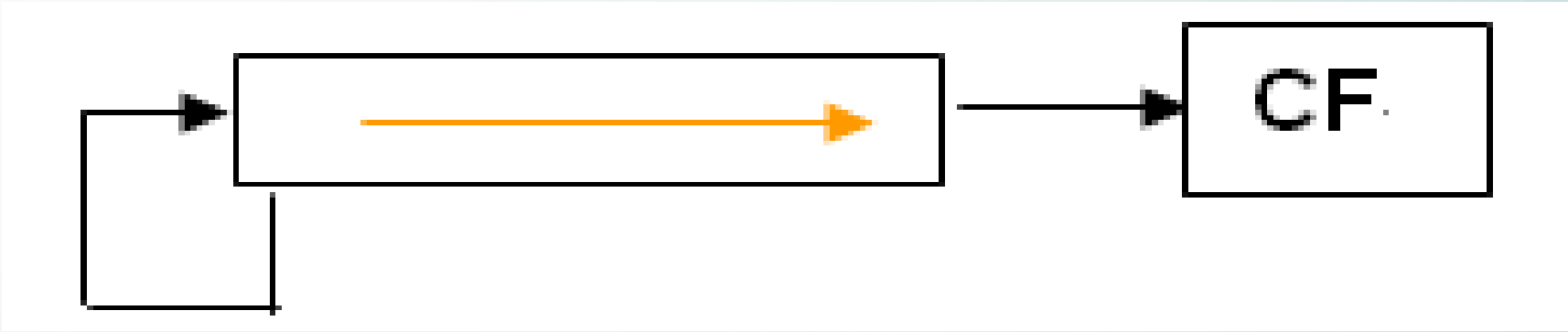
- **SHL** ;逻辑左移，
- **SAL** ;算术左移



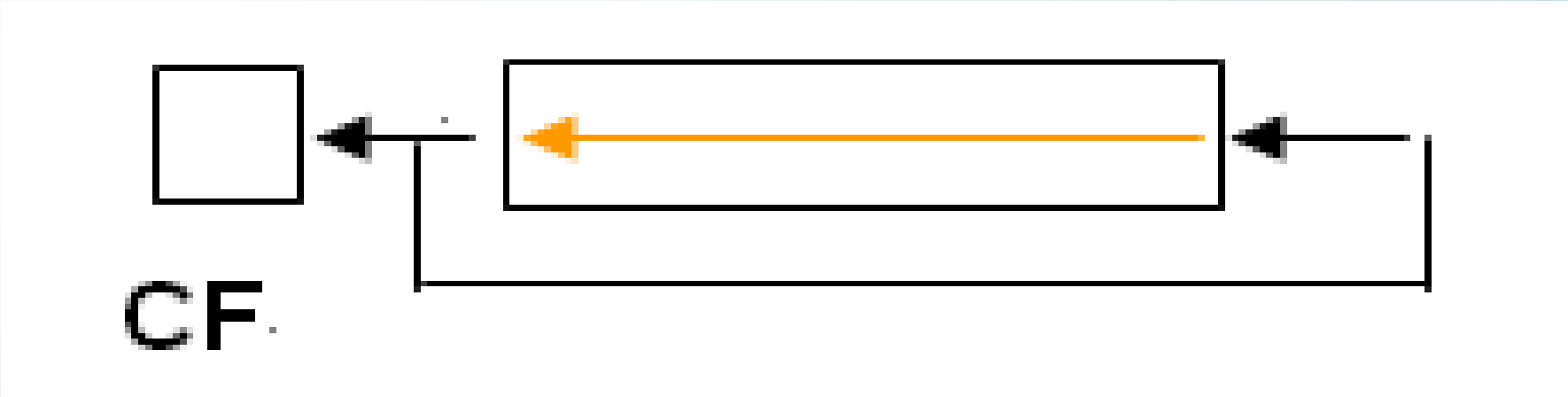
❖**SHR　　;逻辑右移**



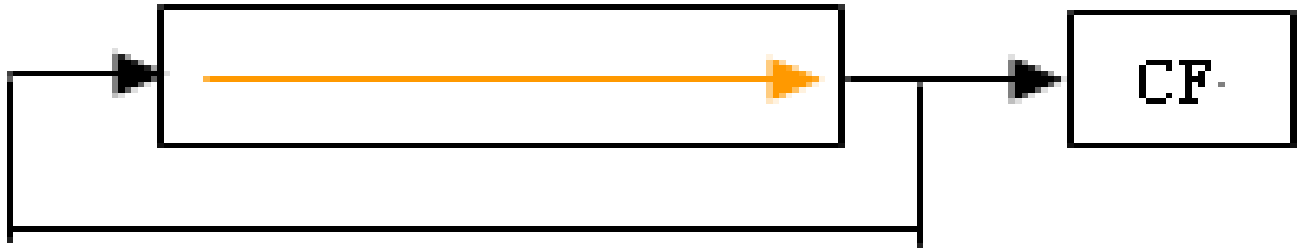
❖**SAR　　;算术右移**



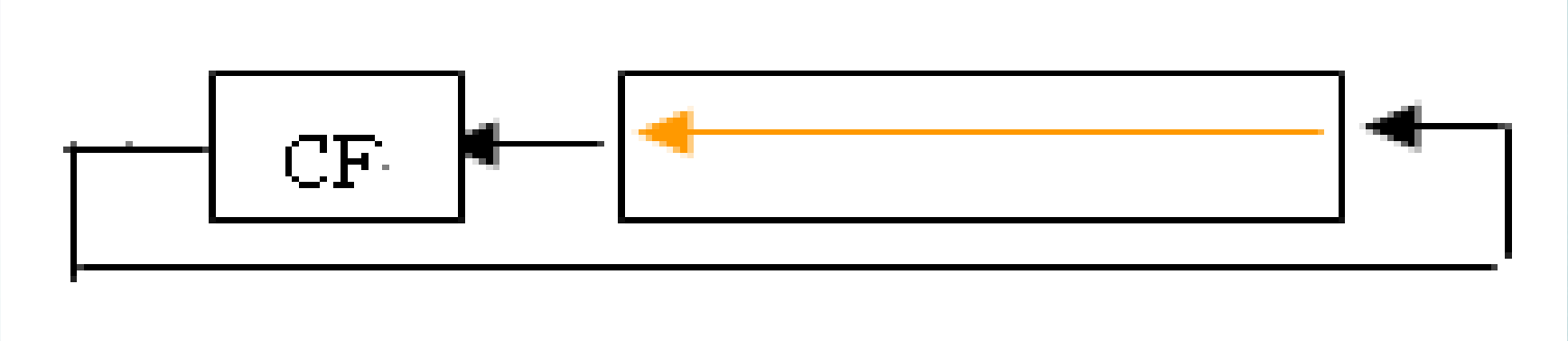
❖**ROL　　;循环左移，**



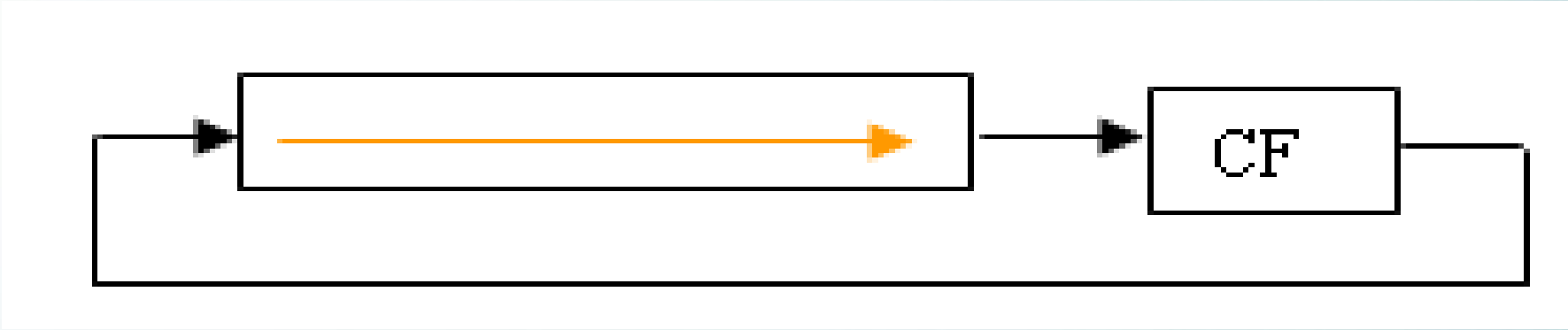
❖**ROR　　;循环右移，**



❖**RCL　　;带进位循环左移,**



❖**RCR　　;带进位循环右移**



- 格式：SHL　OPR, CNT
- CNT可以是1或CL寄存器，如需移位的次数大于1，则可以在该移位指令前把移位次数先送到CL寄存器。
  - SHL ax, 1
  - SHL ax, cl

- 算术移位指令适用于带符号数运算，SAL用来乘以2，SAR用来除以2；逻辑移位指令适用于无符号数运算，SHL用来乘以2，SHR用来除以2。

- 例
  - （1）SHL　AX，1
  - （2）SHR　AX，1
  - （3）SAR　AX，1
  - （4）ROL　AX，1
  - （5）ROR　AX，1
  - （6）RCL　AX，1
  - （7）RCR　AX，1

- 例　对AX中内容实现半字交换，即交换AH和AL中的内容。
  - MOV　CL，8
  - ROL　AX，CL
  - 若指令执行前，AX=1234H，则在指令执行后，AX=3412H。

5.4 串操作指令

- MOVS ;串传送
- CMPS ;串比较
- SCAS ;串扫描
- STOS ;存入串
- LODS ;从串取

- 串操作指令每次处理的是字节或字
  - 需要重复执行串操作指令才能处理完一个数据串。
- 串操作指令通常需要和以下前缀配合使用：
  - **REP** 重复
  - **REPE / REPZ** 相等或为零则重复
  - **REPNE / REPNZ** 不相等或不为零则重复

- **REP的作用**
  - 重复执行串操作指令，直到CX=0为止.
  - 串操作指令每执行一次，使CX自动减1.
- **REPE/REPZ的作用**
  - 当CX≠0并且ZF=1时，重复执行串操作指令，直到CX=0 或者 ZF=0为止.
  - 串操作指令每执行一次，使CX自动减1.
- **REPNE/REPNZ的作用**
  - 当CX≠0并且ZF=0时，重复执行串操作指令，直到CX=0 或者 ZF=1为止.
  - 串操作指令每执行一次，使CX自动减1.

5.4.1 MOVS串传送指令

格式有3种:

- MOVS DST, SRC ;操作数寻址方式固定
- MOVSB ;字节
  - 字节操作:

    (ES:DI)←(DS:SI), SI=SI±1, DI=DI±1
- MOVSW ;字
  - 字操作:

    (ES:DI)←(DS:SI), SI=SI±2, DI=DI±2

当方向标志DF=0,用+,DF=1,用-

实现整个串传送前的准备工作

- SI=源串首地址（如反向传送则是末地址）.
- DI=目的串首地址（如反向传送则是末地址）.
- CX=串长度.
- 设置方向标志DF.

- 设置方向标志DF：

  CLD　设置正向 (DF=0,向前,地址自动增量)

  STD　设置反向 (DF=1,向后,地址自动减量)

❖ 例　在数据段中有一个字符串MESS，其长度为19，要求把它们转送到附加段中名为BUFF的一个缓冲区中，并显示出BUFF字符串，编制程序如下所示。

```
data  segment
      mess   db   'COMPUTER SOFTWARE $'
data  ends
ext  segment
      buff   db    19 dup(?)
ext  ends
```

(ES:DI) - (DS:SI)

```
code  segment
    assume  cs:code，ds:data，es：ext
start:
    mov    ax, data          ；赋段地址
    mov    ds, ax
    mov    ax, ext
    mov    es, ax
    lea    si, mess          ；赋偏移地址
    lea    di, buff
    mov    cx, 19
    cld
    rep    movsb；完成串传送
    mov    bx, es    ；准备显示buff字符串
    mov    ds, bx    ；DS:DX指向待显示串的地址
    lea    dx, buff
    mov    ah, 9
    int    21h
    mov    ah, 4ch
    int    21h
code  ends
```

5.4.2 CMPS串比较指令

格式有3种:

- CMPS SRC，DST ;操作数寻址方式固定
- CMPSB ;字节
  - 字节操作:

    (ES:DI)-(DS:SI), SI=SI±1, DI=DI±1
- CMPSW ;字
  - 字操作:

    (ES:DI)-(DS:SI), SI=SI±2, DI=DI±2

当方向标志DF=0,用+,DF=1,用-

指令不保存结果，只是根据结果设置标志位。

- 例：在数据段中有一个长度为19的字符串MESS1，还有一个长度为19的字符串MESS2，比较它们是否相等。若相等显示‘Y’，否则显示‘N’。程序如下所示。

```
data  segment
      mess1   db   ‘computer software $ ’
      mess2   db   ‘comkuter software $ ’
data  ends
code  segment
      assume  cs:code，ds:data
```

(ES:DI) - (DS:SI)

```
start:
    mov   ax, data
    mov   ds, ax
    mov   es, ax    ;  DS=ES
    lea   si, mess1
    lea   di, mess2
    mov   cx, 19
    cld
    repe  cmpsb     ;  比较结束
    jcxz  yes       ;  如果cx=0，说明相等，
    跳转到标号yes
          mov   dl, 'N'    ;  两串不相等
          jmp   disp       ;  跳转disp
yes:
          mov   dl, 'Y'
disp:
          mov   ah，2
          int   21h
          mov   ah, 4ch
          int   21h
code  ends
end  start
```

5.4.3 SCAS串扫描指令

格式有3种：

- SCAS DST ;操作数寻址方式固定
- SCASB ;字节
  - 字节操作：

    AL-(ES:DI), DI=DI±1
- SCASW ;字
  - 字操作：

    AX-(ES:DI), DI=DI±2

当方向标志DF=0,用+, DF=1, 用-

指令不保存结果，只是根据结果设置标志位。

- 字节操作:

  AL-(ES:DI), DI=DI±1

- 字操作:

  AX-(ES:DI), DI=DI±2

- 当方向标志**DF=0**,用**+**,**DF=1**,用**-**
- 指令不保存结果，只是根据结果设置标志位。

❖ 例　在附加段中有一个字符串MESS，其长度为19，要求查找其中有无空格符，若有空格符，把首次发现的空格符改为‘#’，存回该单元，并显示‘Y’，否则显示‘N’。编制程序如下所示。

```
ext   segment
        mess   db   ‘COMPUTER SOFTWARE $ ’
ext   ends
code  segment
        assume  cs:code，  es:ext
start:
        mov    ax, ext
        mov    es, ax
```

```
lea    di, mess
mov    cx, 19
mov    al，20h        ；空格符
cld
repne  scasb
jz     yes            ；如果zf=1跳转到标号yes
mov    dl, ‘n’
jmp    disp           ；跳转到标号disp
```

AL - (ES:DI)

```
yes:   dec    di
       mov    byte ptr es:[di],23h    ;    ‘#’送原空格位置
       mov    dl,    ‘y’
disp:  mov    ah，2
       int    21h
       mov    ah, 4ch
       int    21h
code  ends
       end  start
```

5.4.4 STOS存入串指令

格式有3种：

- STOS DST ;操作数寻址方式固定
- STOSB ;字节
- STOSW ;字

- 字节操作:

  (ES:DI)←AL, DI=DI±1

- 字操作:

  (ES:DI)←AX, DI=DI±2

- 当方向标志**DF=0**，用**+**，**DF=1**，用**-**

❖例　写出把附加段EXT中的首地址为MESS，长度为9个字的缓冲区置为0值的程序片段。

(ES:DI)←AX

```
MOV    AX, EXT
MOV    ES, AX
LEA    DI, MESS
MOV    CX, 9
MOV    AX，0
CLD
REP    STOSW
```

❖注意：REP STOSW是字操作，每次执行时DI自动+2。

5.4.5 LODS从串取指令

格式有3种：

- LODS SRC ;操作数寻址方式固定
- LODSB ;字节
- LODSW ;字

- 字节操作:

  AL←(DS:SI), SI=SI±1

- 字操作:

  AX←(DS:SI), SI=SI±2

- 当方向标志**DF=0**,用**+**,**DF=1**,用**-**
- 指令一般不和**REP**连用。

5.5 程序转移指令

- 无条件转移指令
- 条件转移指令
- 循环指令
- 子程序调用指令
- 中断调用指令

5.5.1 无条件转移指令

- **JMP 跳转指令**
  - 无条件转移到指令指定的地址去执行程序。
- 转移的目标地址和本跳转指令在同一个代码段，则为段内转移；否则是段间转移。
- 转移的目标地址在跳转指令中直接给出，则为直接转移；否则是间接转移。

（1）段内直接转移

- 格式: JMP NEAR PTR OPR
- 操作: IP←IP+16位位移量
- NEAR PTR为目标地址OPR的属性说明，表明是一个近（段内）跳转，通常可以省略。
- 位移量是带符号数，IP的值可能减小（程序向后跳），也可能增加（程序向前跳）。
- 程序的重新定位并不影响程序的正确执行。

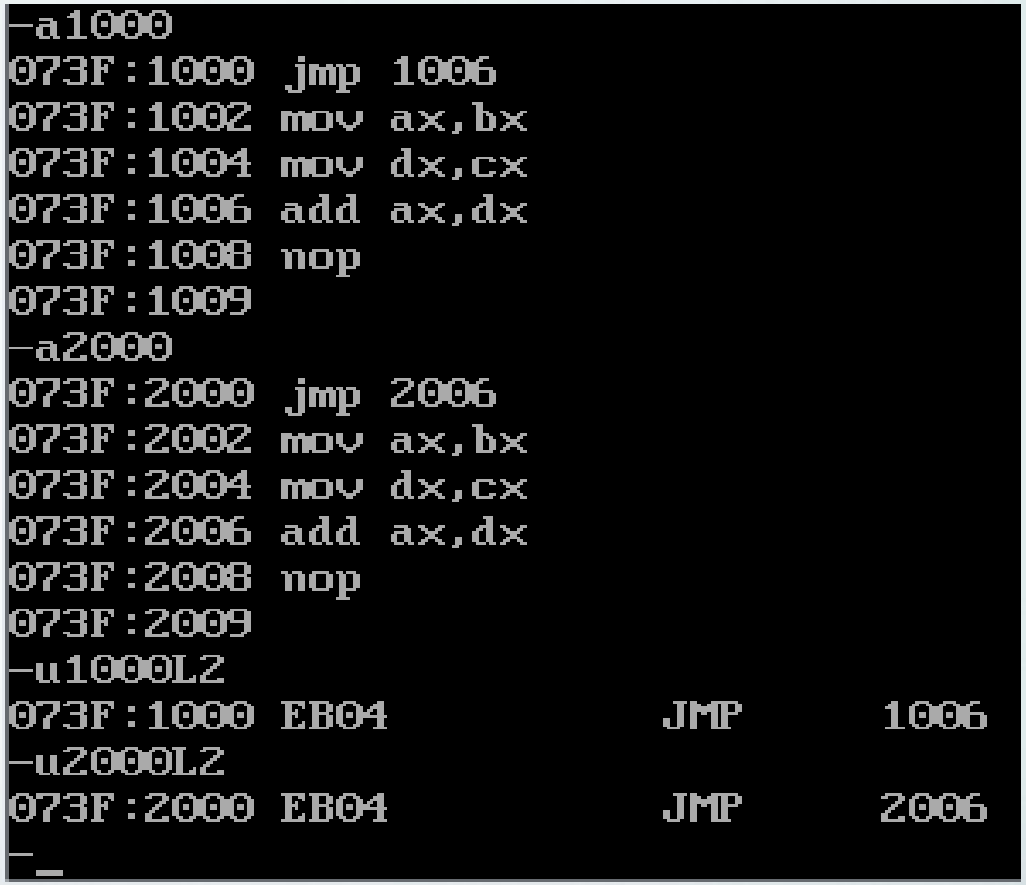
- 例：关于程序的可重新定位的讨论。

```
1000：JMP  P1        ；1000H是本条指令的所在偏移地址
1002：MOV  AX，  BX
1004：MOV  DX，  CX
P1：ADD  AX，  DX   ；P1是标号，其值为1006H
```

- 如果把这个程序放在内存中的另一个位置，如下所示：

```
2000：JMP  P1        ；2000H是本条指令的所在偏移地址
2002：MOV  AX，  BX
2004：MOV  DX，  CX
P1：ADD  AX，  DX   ；P1是标号，其值为2006H
```

- 显然这两段程序是一样的，无论在内存什么位置，不应影响运行结果。


-a1000
073F:1000 jmp 1006
073F:1002 mov ax,bx
073F:1004 mov dx,cx
073F:1006 add ax,dx
073F:1008 nop
073F:1009
-a2000
073F:2000 jmp 2006
073F:2002 mov ax,bx
073F:2004 mov dx,cx
073F:2006 add ax,dx
073F:2008 nop
073F:2009
-u1000L2
073F:1000 EB04 JMP 1006
-u2000L2
073F:2000 EB04 JMP 2006
-_

（2）段内间接转移

- 格式: **JMP WORD PTR OPR**
- 操作: **IP←（EA）**
- 可以使用除立即数以外的任何一种寻址方式。

❖ 例　如果BX=2000H，DS=4000H，(42000H)=6050H，(44000H)=8090H，TABLE的偏移地址为2000H，分析下面四条指令单独执行后IP的值。

```
JMP   BX                    ；寄存器寻址，IP=BX
JMP   WORD PTR [BX]         ；寄存器间接寻址，IP=[DS:BX]
JMP   WORD PTR TABLE        ；直接寻址，IP=[DS:TABLE]
JMP   TABLE[BX]             ；寄存器相对寻址，IP=[DS:(TABLE+BX)]
```

第一条指令执行后，IP=BX=2000H。

第二条指令执行后，IP=(DS:2000H)=(40000H+2000H)=(42000H)=6050H。

第三条指令执行后，IP=(DS:2000H)=(40000H+2000H)=(42000H)=6050H。

第四条指令执行后，IP=(DS:4000H)=(40000H+4000H)=(44000H)=8090H。

（3）段间直接转移

- 格式: JMP FAR PTR OPR
- 操作: IP←OPR的偏移地址

  CS←OPR所在段的段地址

（4）段间间接转移

- 格式: **JMP DWORD PTR OPR**
- 操作: **IP←（EA）**

  CS←（EA+2）
- 可以使用除立即数和寄存器方式以外的任何一种寻址方式。

❖例 如果BX=2000H，DS=4000H，(42000H)=6050H，(42002H)=1234H，指出下面指令执行后IP和CS的值。

JMP DWORD PTR [BX]

指令执行后，

IP=(DS:2000H)=(40000H+2000H)=(42000H)=6050H；CS=(42002H)=1234H。

5.5.2 条件转移指令

- 条件转移指令根据上一条指令所设置的标志位来判别测试条件，从而决定程序转向。
- 通常在使用条件转移指令之前，应有一条能产生标志位的前导指令，如**CMP**指令。
- 汇编指令格式中，转向地址由标号表示。
- 所有的条件转移指令都不影响标志位。

- 第一组: 根据单个条件标志的设置情况转移
- 第二组: 测试CX寄存器的值为0则转移
- 第三组: 比较两个无符号数, 根据结果转移
- 第四组: 比较两个有符号数, 根据结果转移

(1) 根据单个条件标志的设置情况转移

- JZ (JE) 结果为零转移
  - 格式: JZ OPR
  - 测试条件:ZF=1
- JNZ (JNE) 结果不为零转移
  - 格式: JNZ OPR
  - 测试条件:ZF=0
- JS 结果为负转移
  - 格式: JS OPR
  - 测试条件:SF=1

- **JNS OPR　结果不为负（为正）转移**
  - 测试条件:SF=0
- **JO　OPR　结果溢出转移**
  - 测试条件:OF=1
- **JNO　OPR　结果不溢出转移**
  - 测试条件:OF=0
- **JP (JPE)　奇偶位为1转移**
  - 格式: JP　OPR
  - 测试条件:PF=1

- **JNP (JPO)　奇偶位为0转移**
  - 格式: JNP　OPR
  - 测试条件:PF=0
- **JB　(JNAE,JC)　低于,(不高于等于,进位位为1),则转移.**
  - 格式: JB　OPR
  - 测试条件:CF=1
- **JNB　(JAE,JNC)　不低于,(高于等于,进位位为0),则转移.**
  - 格式: JNB　OPR
  - 测试条件:CF=0

(2)测试CX寄存器的值为0则转移

- 格式: JCXZ　OPR
- 测试条件:CX=0

(3)比较无符号数,根据结果转移

- **JB (JNAE,JC) 低于,(不高于或等于,进位位为1),则转移.**
  - 格式: JB OPR
  - 测试条件:CF=1
- **JNB (JAE,JNC) 不低于,(高于等于,进位位为0),则转移.**
  - 格式: JNB OPR
  - 测试条件:CF=0
- **JBE (JNA) 低于或等于,(不高于),则转移.**
  - 格式: JBE OPR
  - 测试条件:CF OR ZF=1
- **JNBE (JA) 不低于或等于,(高于),则转移.**
  - 格式: JNBE OPR
  - 测试条件:CF OR ZF=0

(4)比较带符号数，根据结果转移

- JL (JNGE) 小于,(不大于等于),则转移. <
  - 格式: JL OPR
  - 测试条件:SF XOR OF=1
- JNL (JGE) 不小于,(大于等于),则转移. >=
  - 格式: JNL OPR
  - 测试条件:SF XOR OF=0
- JLE (JNG) 小于等于,(不大于),则转移. <=
  - 格式: JLE OPR
  - 测试条件:(SF XOR OF) OR ZF=1
- JNLE (JG) 不小于等于,(大于),则转移. >
  - 格式: JNLE OPR
  - 测试条件:(SF XOR OF) OR ZF=0
- 表5-3 有符号数的比较判断条件

****

为何针对有符号数和无符号数须用不同指令?

****

8位二进制数FFH 和 00H ,哪个大?

若为无符号数, FFH大,若为有符号数,00H大.

❖ 例 有一个长为19字节的字符串，首地址为MESS。查找其中的‘空格’(20H) 字符,如找到则继续执行，否则转标号NO。

```
      MOV   AL，20H
      MOV   CX，19
      MOV   DI，-1
LK:   INC   DI
      DEC   CX
      CMP   AL, MESS[DI]
      JCXZ  NO
      JNE   LK
      …
      …
```

5.5.3 循环指令

- **LOOP** 循环
- **LOOPZ / LOOPE** 为零或相等时循环
- **LOOPNZ / LOOPNE** 不为零或不相等时循环

- 指令：**LOOP　OPR**

  测试条件:**CX≠0**，则循环
- 指令：**LOOPZ / LOOPE　OPR**

  测试条件:**ZF=1 AND CX≠0**，则循环
- 指令：**LOOPNZ / LOOPNE　OPR**

  测试条件:**ZF=0 AND CX≠0**，则循环
- 操作：首先CX寄存器减1，然后根据测试条件决定是否转移。

❖ 例　在首地址为MESS长为19字节的字符串中查找 ‘空格’(20H) 字符,如找到则继续执行，否则转标号NO。用循环指令实现程序的循环。

```
    MOV    AL，20H
    MOV    CX，19
    MOV    DI，-1
LK: INC    DI
    CMP    AL, MESS[DI]
    LOOPNE  LK
    JNZ    NO
    …
```

本章关键词

- 数据传送指令
  - **MOV, PUSH, POP, XCHG**
  - **IN, OUT, XLAT**
  - **LEA, LDS, LES**
  - **LAHF, SAHF, PUSHF, POPF**
- 算术运算指令
  - **CBW, CWD**
  - **ADD, ADC, INC**
  - **SUB, SBB, DEC, NEG, CMP**
  - **MUL, IMUL**
  - **DIV, IDIV**
  - **DAA, DAS**

- 逻辑指令与移位指令
  - **AND, OR, NOT, XOR, TEST**
  - **SHL, SHR, SAL, SAR, ROL, ROR, RCL, RCR**
- 串操作指令
  - **MOVS, CMPS, SCAS, STOS, LODS　　(B, W)**
  - **REP, REPE / REPZ, REPNE / REPNZ**

程序转移指令

- **JMP,**
- **JZ (JE), JS, JO, JNO, JP(JPE), JNP, JCXZ**
- **JB(JNAE, JC), JNB(JAE, JNC), JBE(JNA), JNBE(JA),**
- **JL(JNGE), JNL(JGE), JLE(JNG), JNLE(JG)**
- **LOOP, LOOPZ / LOOPE, LOOPNZ / LOOPNE**