二、算术运算指令

- 按加减乘除分为四类，每类包括数制调整指令：
  - 加法：**ADD、ADC、INC、AAA、DAA**
  - 减法：**SUB、SBB、DEC、NEG、CMP、AAS、DAS**
  - 乘法：**MUL、IMUL、AAM**
  - 除法：**DIV、IDIV、AAD、CDW、CWD**
- 算术运算指令可处理的数据类型：
  - 无符号二进制数
  - 带符号二进制数（补码）
  - 压缩**BCD**码（**1**字节表示**2**位十进制数）
  - 非压缩**BCD**码（**1**字节只含**1**位十进制数，高**4**位全**0**）

预备知识：关于溢出的讨论

一、回顾：**FLAGS**中的**6**个状态标志位（**ACOPSZ**）

- **CF**，进位标志。本次运算最高位有进位或借位发生，则**CF=1**。
- **AF… …**
- **ZF… …**
- **… …**
- **OF**，溢出标志。本次运算结果产生溢出（对于有符号数，**8**位表示范围为**-128～+127**，**16**位表示范围为**-32768～+32767**，超出此范围为溢出），或者运算过程中，最高位已无法再向前进位或借位，则**OF=1**。

二、以8位加法运算为例，观察CF和OF

1、无符号数，有符号数都不溢出

	无符号数	有符号数
04H	4	4
+ 0BH	+ 11	+ 11
0FH	15	15
	CF=0	OF=0

2、无符号数溢出，有符号数不溢出

	无符号数	有符号数
07H	7	7
+FBH	+251	+(-5)
1 02H	258 (>255)	+2
	CF=1	OF=0

3、有符号数溢出，无符号数不溢出

	无符号数	有符号数
09H	9	9
+7CH	+124	+124
85H	133	133
		(>127)
	CF=0	OF=1

4、无符号数，有符号数都溢出

	无符号数	有符号数
87H	135	-121
+F5H	+245	+(-11)
1 7CH	380	-132
	(>255)	(<-128)
	CF=1	OF=1

结论：OF——表示有符号数的溢出

CF——表示无符号数的溢出（进位）

（一）加法指令

1. 不带进位位的加法

格式：ADD DST，SRC ； DST←DST+SRC

例： ADD AL，50H ； AL ←AL+50H

ADD [BX+DI]，AX

注意：影响标志AF、OF、PF、SF、ZF，CF（参见P80）。

2. 带进位位的加法

格式：ADC DST，SRC ;DST←DST+SRC+**CF**

例： ADC AX，SI ;AX ←AX+SI+**CF**

ADC DX，[SI]

注意：影响标志AF、OF、PF、SF、ZF，CF（参见P80） 。

- 说明：

  1）ADD和ADC的SRC和DST不能都是内存单元

  2）SRC和DST的数据类型必须一致，即都是字节或字。

3. 加一指令

- 格式： INC DST ；DST ← DST+1
- 例： INC AX ； AX ← AX+1

  INC BL ； BL ← BL+1
- 该指令主要用于修改地址指针和循环计数器。
- 注意：影响AF、OF、PF、SF、ZF，但不影响CF。

关于BCD码加法运算以及十进制调整

CPU做加法运算时一律按二进制数规则加。因此两个BCD码相加时会出现两种情况：

① 和≦9

② 和>9　此时和的范围为0AH～12H（18）之间，其中，若和在0AH～0FH之间时，AF=0；若和在10H～12H之间时，AF=1或CF=1。

例：三个加法运算

0101	0101	1001
+ 0011	+ 0111	+ 1001
1000	1100	AF 0010

①结果正确　　②两个结果都是错误的

解决办法：十进制调整。

即遇到以上情况②时，对BCD结果进行加6调整。

原先	5+7	9+9
	0101	1001
	+ 0111	+ 1001
	1100	AF 0010
加6调整后	1100	AF 0010
	+ 0110	+ 0110
	10010	AF 1000

4.加法的ASCII调整指令

- 格式：AAA
- 功能：在用ADD或者ADC对两个非压缩BCD码或用ASCII码表示的十进制数作加法后，将AL中的和进行调整，结果放在AL或者AX中 。
- 策略：
  - 若AL低4位的值大于9或AF=1， 则AL＋6→AL，将AL高4位清零，同时置AF＝1、CF＝1、AH＋1→AH。
  - 否则仅将AL高4位清零。
- AAA指令影响标志AF、CF。AAA指令必须紧跟着ADD或ADC指令之后。

AAA指令示例1

计算十进制数9＋4

```
MOV AL, 9H        09H
MOV BL, 4H      + 04H
                -------
ADD AL, BL        0DH     ； >9
AAA             + 06 H    ； 调整
                -------
                 1 3H
```

送入AH

结果：

(AH)=01H (AL)=03H CF=AF=1

AAA指令示例2：

- 计算两个ASCII码“9”和“5”的加法
- ASCII码‘9’和‘5’的编码分别是39H和35H，一般应先使用0FH分别与其相与（∧0FH），将ASCII码转换成非压缩的BCD码后再相加。利用AAA指令则可以先将两个ASCII直接相加，然后再对结果进行调整。

```
MOV AL, 39H          39H
MOV BL, 35H        + 35H
                   ------
ADD AL, BL           6EH     ； 低4位>9
AAA                + 06H     ； 调整，低4位加6，高
                   ------
        （AX）＝ 01 04H      ； 4位清零，置CF＝1
                             ； CF＝1，AF＝1
```

送入AH（指向01）

- 结果调整：将AX的内容转换成ASCII码，只要使用“或”操作指令OR AX，3030H，AX的内容即为ASCII码‘1’‘4’。

5. 加法的十进制调整指令

- 格式：DAA
- 功能：在用ADD或者ADC对两个压缩BCD数作加法后，将AL中的和进行调整，得到一个正确的压缩BCD数并仍然放在AL或者AX中 。
- 策略：
  - 若AL低4位的值大于9或AF=1， 则AL＋6→AL；
  - 然后，若AL高4位的值大于9或CF=1， 则AL＋60H→AL，并置CF=1。否则CF＝0。
- DAA指令影响标志AF、CF、PF、SF、ZF 。DAA指令必须紧跟在ADD或者ADC加法指令之后。

DAA指令示例

例：计算 89 ＋ 75 ＝ 164

MOV　AL，89H；89H看作是两个非压缩BCD码

MOV　BL，75H；89H看作是两个非压缩BCD码

ADD　AL，BL　；（AL）=0FEH，AF=0，CF=0

DAA　　　　　；（AL）=64H，CF=1

```
     1000 1001 (89)
 +   0111 0101 (75)
 ------------------
     1111 1110 (FEH)
 +   0110 0110 (低4位+6调整，高4位再+60H调整)
 ------------------
   1 0110 0100
```

(二)减法指令

1. 不带借位的减法

- 格式：**SUB DST，SRC** ；DST ← DST-SRC

（对比：ADD）

例1： SUB BX，CX ；BX ← BX-CX

例2： SUB WORD PTR [DI]，1000H

；说明：PTR是一条修改属性运算符的伪指令，表示将PTR左边的数据类型属性赋予右边的变量或标号。本例指令的作用是将起始地址为 DS×16＋(DI)的连续两个存储单元看成一个字，将该字减去1000H后结果仍然存在原来位置。

注意：SUB指令影响标志AF、OF、PF、SF、ZF，CF。

2. 带借位的减法

- **格式：SBB DST，SRC** ； DST ← DST–SRC–CF

（对比ADC）

例1： SBB AX，2030H；

功能：执行(AX) ← (AX)－2030H－CF

例2： SBB DX，[BX+20H]

功能：执行(DX)←(DX)－[DS×16＋(BX)＋20H]－CF

注意：SBB影响标志AF、OF、PF、SF、ZF，CF。

3. 减量指令

- 格式：**DEC DST** ； DST ←DST － 1（对比INC）

注意：影响标志AF、OF、PF、SF、ZF，CF。

4. 取负指令

- 格式：**NEG DST** ； 对DST求补码，DST←0－DST

5. 比较指令

- 格式：**CMP DST，SRC** ； DST－SRC

注意：CMP指令执行相减，但不回送结果，结果只影响标志位CF、OF、SF、ZF。

关于比较指令CMP的进一步讨论

1. 必须区分无符号数比较与有符号数比较

- 如：比较11111111B与00000000B
  - 无符号数比较：255 > 0
  - 有符号数比较：－1 < 0

2. 比较两数是否相等，应根据标志位ZF判断。若相等，则ZF=1；否则ZF=0

- 问题：如果两数不相等，执行比较指令以后如何判断两数之间的大小关系？

比较两数的大小　CMP DST, SRC

1、无符号数比较

DST≥SRC	DST<SRC
DST=80H　SRC=58H	DST=58H　SRC=80H
80H	58H
-58H	-80H
28H	D8H
CF=0　够减	CF=1　不够减

结论：用标志位CF判断无符号数的大小

CF=0，则 DST ≥ SRC

CF=1，则 DST < SRC

2、有符号数比较（分4种情况）

(1) DST>0, SRC>0 （必不溢出，OF=0）

DST=5AH, SRC=46H

```
  5AH
- 46H
-----
  14H
```

SF=0, DST>SRC

DST=46H, SRC=5AH

```
  46H
- 5AH
-----
  ECH
```

SF=1, DST<SRC

(2)DST>0, SRC<0 （必有DST>SRC）

DST=10H SRC=95H

```
  0001 0000B（10H）
- 1001 0101B（95H）
-------------------
  0111 1011B（7BH）
```

SF=0, OF=0

DST=62H SRC=95H

```
  0110 0010B（62H）
- 1001 0101B（95H）
-------------------
  1100 1101B（CDH）
```

SF=1, OF=1

(3)DST<0, SRC>0（必有DST<SRC）

DST=D3H SRC=38H

```
  D3H
- 38H
-----
  9BH
```

SF=1, OF=0

DST=BFH SRC=55H

```
  BFH
- 55H
-----
  6AH
```

SF=0, OF=1

(4)DST<0, SRC<0 （必不溢出，OF=0）

DST=B5H SRC=9CH

```
  B5H
- 9CH
-----
  19H
```

SF=0， DST>SRC

DST=9CH SRC=B5H

```
  9CH
- B5H
-----
  E7H
```

SF=1， DST<SRC

结论：用标志位SF和OF判断有符号数的大小

SF、OF值相同，则 DST≥SRC

SF、OF值不同，则 DST＜SRC

CMP指令示例1

若(AL)= −64　　(BL)=10

执行：**CMP AL, BL**

　　− 64
　　− 10
　　――――
　　− 74

OF=0　SF=1　　结论：**(DST) < (SRC)**

CMP指令示例2

若 (CL)= −100　　(DS:100)= −110

执行：**CMP CL, [100H]**

　　-100
　　− (−110)
　　――――
　　10

OF=0　SF=0　　结论：**(DST) > (SRC)**

6. 减法的ASCII码调整指令

- 格式：AAS ； （对比AAA）
- 功能：在用SUB或者SBB对两个非压缩十进制数或用ASCII码表示的十进制数作减法后，将AL中的和进行调整，正确的结果仍然放在AL或者AX中 。
- 策略：
  - 若AL低4位的值大于9或AF=1， 则AL－6→AL，将AL高4位清零，同时置AF＝1、CF＝1、AH－1→AH。
  - 否则不对AL的内容做任何调整。
- AAS应该紧跟在减法指令之后

注意与AAA的区别

AAS指令示例

计算十进制数3－8

```
MOV AL, 03H
MOV BL, 08H
SUB AL, BL ;  低4位大于9
AAS        ;  调整，低4位减6
           ;  高4位清零
```

```
   03H
-  08H
------
   FBH
-  06H
------
   05H
```

高4位清零

结果为AL＝05H，CF＝1，表示有借位

7.减法的十进制调整

- 格式：DAS ；（对比DAA）
- 功能：在用SUB或者SBB对两个压缩BCD数作减法后，将AL中的差进行调整，得到一个正确的压缩BCD数并仍然放在AL中 。
- 策略：
  - 若AL低4位大于9或AF=1，则AL－6→AL，AF置1；否则不需对低4位做调整。
  - 然后，若AL高4位大于9或CF＝1， 则AL－60H →AL，并置CF＝1。否则不需对高4位做调整。
- DAS指令必须紧跟在减法指令之后。

注意与DAA的区别

DAS指令示例1：

设AL＝BCD 56，CL＝BCD 28，求两数之差

```
SUB AL, CL   ；     0101 0110    BCD 56
             ；  －  0010 1001    BCD 28
             ；     0010 1110    低4位大于9，需要调整。
DAS          ；  －  0000 0110    减6调整
             ；     0010 1000    高4位小于9，不需调整。

             ；     结果为BCD 28，AF＝1，CF＝0，
             ；     AL没有向AH借位。
```

DAS指令示例2：

设AL＝BCD 19，CL＝BCD 28，求两数之差

SUB AL, CL ；　　0001 1001　　BCD 19

；－　0010 1000　　BCD 28

；　　1111 0001　　低4位小于9，不需调整

DAS ；－　0110 0000　　高4位大于9，减60调整

；　　1001 0001　　CF置1，AL有借位发生

；结果为91，AF＝0，CF＝1，AL向AH有借位。

(三) 乘法指令

- 二进制乘法特点：
  - 两个8位数相乘，结果为16位数。
  - 两个16位数相乘，结果为32位数。
- 8086乘法指令特点：
  - 乘数总是放在AL（8位）或AX（16位）中；
  - 将DX看成是AX的扩展。

1、无符号数乘法指令

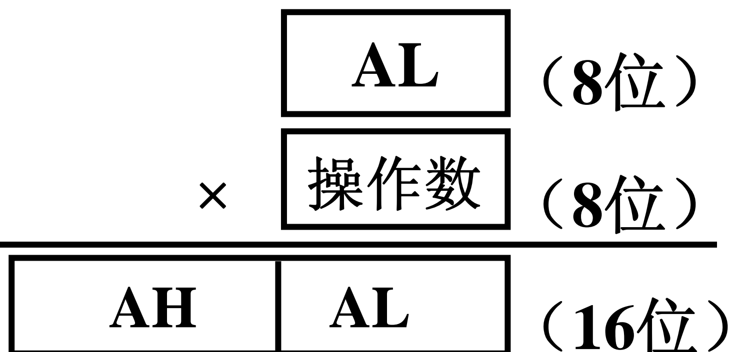
- 格式：MUL SRC ；AX ←AL×SRC，字节

  ；DX，AX←AX × SRC，字
- 说明：1）SRC不能是立即数；

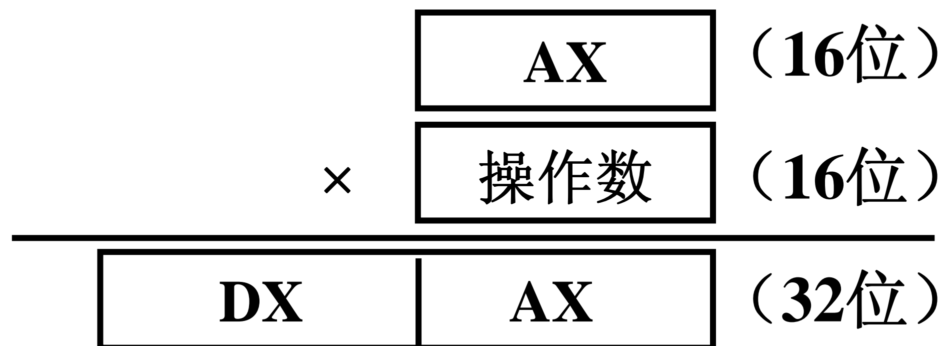
  2）若CF=OF=0，结果高半部分全为零，可去除。

  3）不能用于有符号数的相乘。压缩BCD码没有乘法指令。

8位乘法



16位乘法



2. 带符号数的乘法指令

- 格式：**IMUL SRC** ；AX ← AL×SRC，字节。

  ；DX、AX ← AX×SRC，字。

- 说明：**1**）在**8086**中，**SRC**不能是立即数；

  2）若**CF**＝**OF**＝**0**，结果的高半部分是低半部分符号位的扩展，可去除。

例**1**：**IMUL BL** ；**AL**的内容乘以**BL**的内容，

；结果送**AX**。

例**2**：**IMUL DI** ；**AX**的内容乘以**DI**的内容，

；结果送**DX**、**AX**。

80286以后，增加了以下两种类型IMUL指令：

格式：IMUL REG, SRC或IMM
操作：$REG_{16} \leftarrow (REG_{16}) \times$ (SRC或IMM)
$REG_{32} \leftarrow (REG_{32}) \times$ (SRC或IMM)

格式：IMUL REG, SRC, IMM
操作：$REG_{16} \leftarrow (SRC) \times IMM$
$REG_{32} \leftarrow (SRC) \times IMM$

说明：1）乘积字长和源、目的操作数的字长一致
2）可能溢出
3）SRC可用寄存器或存储器寻址方式；
REG只能是寄存器寻址方式；
IMM是立即数。

MUL, IMUL指令示例

MUL BL

(AL)=0B4H=180

(BL)=11H=17

```
        1011 0100
    ×   0001 0001
    -------------
        1011 0100
1011 0100
-----------------
101111110100
```

(AX)=0BF4H=3060

IMUL BL

(AL)=0B4H= -76,

(BL)=11H=17

$-76_{补}$=4CH

参见教材P87：IMUL实现方法

```
        0100 1100
    ×   0001 0001
    -------------
        0100 1100
0100 1100
-----------------
0101 0000 1100
```

$(AX)=(-050C)_{补}$ =FAF4H= -1292

3. 乘法的ASCII调整指令

- 格式：**AAM**
- 功能：

  1)将两个非压缩BCD码使用MUL指令后存放在AL中的积进行调整，使得在AX中得到正确的非压缩十进制数的乘积，高位放在AH，低位放在AL。

  2)两个ASCII码相乘之前，先屏蔽每个数字的高4位，使其先转换成非压缩BCD码，然后相乘，再用AAN对结果进行调整。
- 调整方法：（P87例3-60）

  1) AH ← AL/10 (0AH)；将AL除以10的商存放在AH中

  2) AL ← AL/10 (0AH)；将AL除以10的余数存放在AL中
- 注1：AAM必须紧跟在MUL指令之后。
- 注2：有符号乘法IMUL没有对应的十进制调整指令。

(四) 除法指令

1. 无符号数除法

- 格式：DIV SRC ；
  - 节节除：AX÷SRC，AL← 商，AH←余数
  - 字 除: (DX.AX)÷SRC，AX←商，DX←余数
- 说明：
  - SRC是除数，被除数在累加器中，且必须是除数的两倍字长。
  - 对于无符号除法，当被除数只有8位时，可将AH清零，把被除数扩展到16位；若除数是16位，被除数只有16位，可将DX清零，把把被除数扩展到32位。
  - SRC可以是除立即数以外所有寻址方式。

2. 有符号数除法

- 格式：IDIV SRC ；
  - 节节除：AX÷SRC，AL← 商，AH←余数
  - 字 除：(DX.AX)÷SRC，AX←商，DX←余数
- 说明：
  - 操作数、商和余数都是带符号数，并且余数的符号与被除数相同。
  - 无/有符号数除法都有除法溢出的问题。原因是如果被除数高位的绝对值大于除数的绝对值，而存放商的寄存器字长只有被除数字长的一半，商"装不下"。产生类型号为0的除法错中断，相当于执行了除0运算。
  - IDIV执行后，所有的标志位没有意义。
  - 被除数字长也必须是除数的两倍。对于有符号除法，不能简单地通过将高位置零的方式进行字长扩展。

3. 扩展字节为字

- 格式：**CBW** ； **AX ←AL**
- 功能：将**AL**的符号位扩展到**AH**中。将一个字节的有符号数扩展为一个字。

4. 扩展字为双字

- 格式：**CWD** ； **DX,AX ←AX**
- 功能：将**AX**的符号位扩展到**DX**寄存器中。将一个字的有符号数扩展为一个双字。

- 说明：**CBW**和**CWD**执行后不影响标志位。

6. 除法的ASCII调整

- 格式：AAD ；
- 功能：在除法运算之前，将BCD码转换成二进制数。
- 策略：(AL)←(AH)×10＋AL，(AH)←0
- 注意：AAD指令应该在除法指令之前。
- AAD指令示例：计算压缩BCD码28÷4

```
MOV  AX, 0208H      ；AX<--被除数
MOV   CL, 4         ；除数
AAD                 ；调整，(AX)=28
DIV  CL             ；结果 (AL)＝7，(AH)＝0
```

习题二：P122～

- 9
- 10

三、逻辑运算和移位指令

1. 逻辑运算指令
2. 非循环移位指令
3. 循环移位指令

（一）逻辑运算指令

① **NOT DST**

② **AND DST，SRC**

③ **OR DST，SRC**

④ **XOR DST，SRC**

⑤ **TEST DST，SRC**

⑥ 位扫描和位测试（**386**后新增）

①、逻辑非指令

- 格式： NOT DST ；DST按位取反
- 说明：本指令不影响标志位

NOT指令示例1

```
MOV AL, 52H        ；执行前AL＝01010010
NOT AL             ；执行后AL＝10101101
```

NOT指令示例2

```
NOT  BYTE PTR[BX]
```

对逻辑地址为DS:BX的存储单元内容取反后送回该单元。

②、逻辑与指令

- 格式： AND DST，SRC ；
- 功能： (DST) ← (DST) ∧ (SRC)
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

AND指令示例1

MOV AL, 32H ；执行前AL为ASCII码'2'
AND AL, 0FH ；屏蔽高4位，执行后AL＝02H

AND指令示例2

AND AX, AX ；执行后 AX内容不变， CF=0

③、逻辑或指令

- 格式： OR DST，SRC ；
- 功能： (DST) ← (DST)∨(SRC)
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

OR指令示例

```
MOV    AX,  0508H
OR     AX,  3030H
```

执行后AX＝3538H，将AX中原先存放的两位非压缩BCD码转换成两个ASCII码’5’和’8’。

④、异或指令

- **格式：XOR DST，SRC** ；
- 功能： (DST) ← (DST) ⊕ (SRC)，按位异或。
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

XOR指令示例1

```
MOV  AL,  0B6H  ;
XOR  AL,  0FH   ;
```

```
      1011 0110
    ⊕ 0000 1111
    -----------
AL＝1011 1001
    不变  变反
```

与0/1异或不变/求反

XOR指令示例2

XOR AL, AL ；清零操作，且CF=0

思考：用一条指令使AX清零，有几种方法？

⑤、测试指令

- 格式：TEST DST，SRC ；
- 功能：(DST)∧(SRC)，但是不回送结果
- 说明：影响标志位PF、SF、ZF，使CF=0、OF=0

TEST指令示例1

测试AL最低位是否为1，若是1则转移

```
TEST AL, 01H
JNZ  NEXT     ；是1非0 (ZF=0)，转NEXT
```

AND指令示例2

测试AX最高位是否为1，若是1则转移

```
TEST AX, 8000H
JNZ  THERE    ；是1非0 (ZF=0)，转THERE
```

⑥位测试和位扫描指令

- 80386以后CPU新增的命令
- 格式：OP DST，SRC ； OP为以下操作码
- OP：
  - BT (bit test) 位测试
  - BTS(bit test and set) 位测试并置1
  - BTR(bit test and reset) 位测试并清0
  - BTC(bit test and complement) 位测试并变反
  - BSF(bit scan forward) 正向位扫描
  - BSR(bit scan reverse) 反向位扫描

指令	操作
BT	BT测试DST中由SRC指定的位，将测试位送CF。
BTS	测试DST中由SRC指定的位，将测试位送CF，并将DST中该位置1。
BTR	测试DST中由SRC指定的位，将测试位送CF，并将DST中该位置0。
BTC	测试DST中由SRC指定的位，将测试位送CF，并将DST中该位变反。
BSF	从低位至高位扫描SRC第一个为“1”的位，并将位号送DST。
BSR	从高位至低位扫描SRC第一个为“1”的位，并且将位号送DST。

位测试指令示例1

BT AX, 4

若执行前 (AX)=1234H，则执行后， CF=1

若执行前 (AX)=1224H，则执行后， CF=0

位测试指令示例2

若执行前 (AX)=0000H

BTS AX, 4

执行后，CF=0，(AX)=0010H

位扫描指令示例

BSF ECX, EAX ；从右至左扫描

BSR EDX, EAX ；从左至右扫描

若执行前 (EAX)=60000000H，

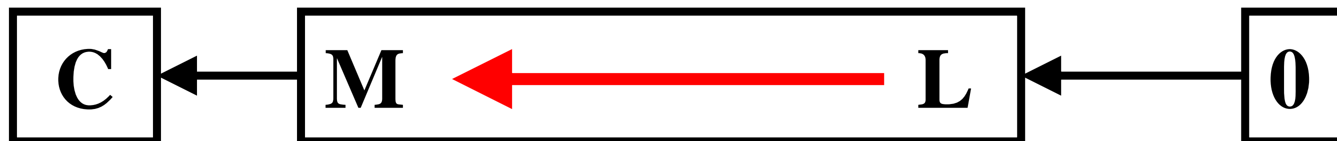
则执行后， (ECX)=29d， (EDX)=30d

(二) 非循环移位指令

① SHL DST，count；逻辑左移

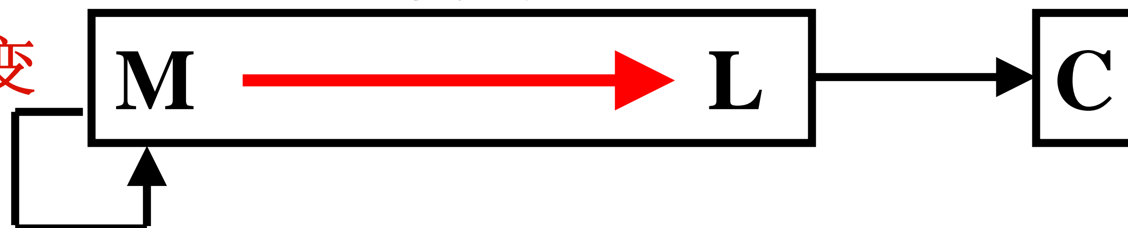
② SHR DST，count；逻辑右移

③ SAL DST，count；算数左移



④ SAR DST，count；算数右移

最高位不变



- 注意：
  - 移位指令影响标志位CF、OF、PF、SF和ZF。
  - 如果只移一位，指令中可用立即数'1'指出移位的位数，如果超过1位，移位的位数必须事先放在CL中。
  - CL可以表示的移位次数最大为255，80286以后的CPU规定最大的移位次数为31（00011111），超过此范围也仅截取低5位，
- 例：

```
MOV   CL，4
SAL   DX，1   ；DX中的数左移1位
SAL   AX，CL；AX中的数左移4位
SHL   AL，CL；AL中的数左移4位（同SAL AL，CL）
SHR   AL，CL；AL中的数右移4位
```

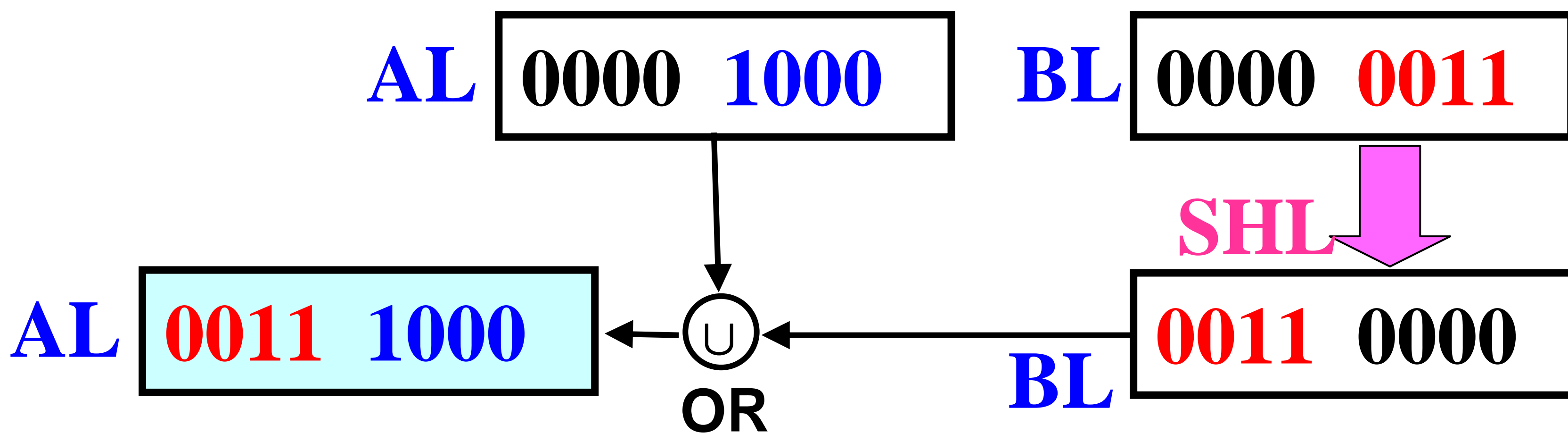
- 例：已知(AL)=0B4H, (CF)=1，分析下列指令执行后的结果

	CF	AL
执行前	1	10110100
(1) SAL AL, 1	1	01101000
(2) SAR AL, 1	0	11011010
(3) SHL AL, 1	1	01101000
(4) SHR AL, 1	0	01011010

- 算术左移与逻辑左移的结果相同
- 算术右移与逻辑右移的结果不同

例：分析下列程序片段，求执行后(AL)＝？

```
PRO1:   MOV   AL，04H
        MOV   BL，03H
        MOV   CL，4
        SHL   BL，CL
        OR    AL，BL
        HLT
```



- 通常:
  - 算术右移N位相当于有符号数除以2^N(有例外)
  - 逻辑右移N位相当于无符号数除以2^N
  - 算术/逻辑左移N位相当于无符号数乘以2^N

- 例：已知 AL＝ 01100100（64H＝100）

  MOV CL, 5

  SAR AL, CL ；AL＝00000011（03H＝3）

  当最右边的"1"移出后，除法运算结果不对

- 例题：已知变量Y中为一字节无符号数，计算(Y)×10，积放在AX中。

变换：(Y)×10＝(Y)×(8＋2)＝(Y)×2＋(Y)×8

提问： 为什么要这样变换？

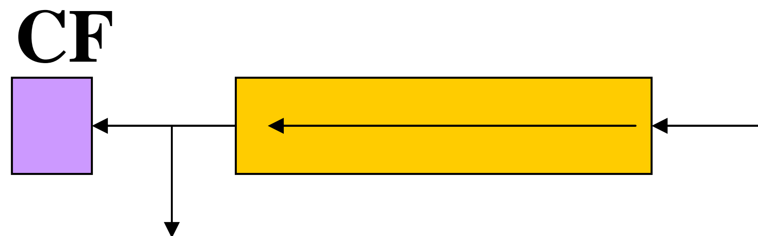
```
MOV   AL, Y     ; AL<--(Y)
MOV   AH, 0
SHL   AX, 1     ; (Y)×2
MOV   BX, AX
SHL   AX, 1     ; (Y)×4
SHL   AX, 1     ; (Y)×8
ADD   AX, BX    ; (Y)×10
```

思考题：利用移位指令计算 DX＝3×AX＋7×BX

(三) 循环移位指令

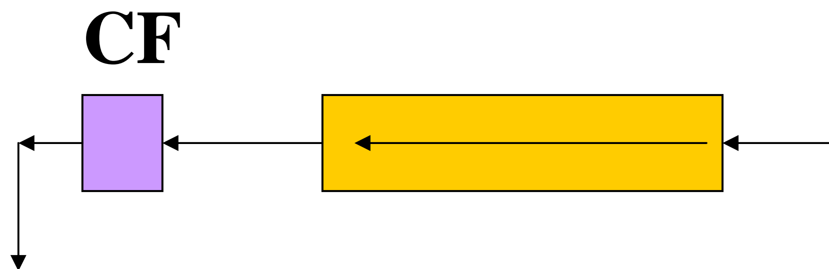
1) 循环左移

ROL DST，count



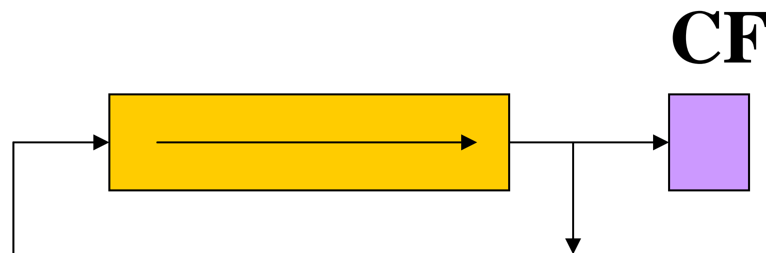
2) 带进位循环左移

RCL DST，count



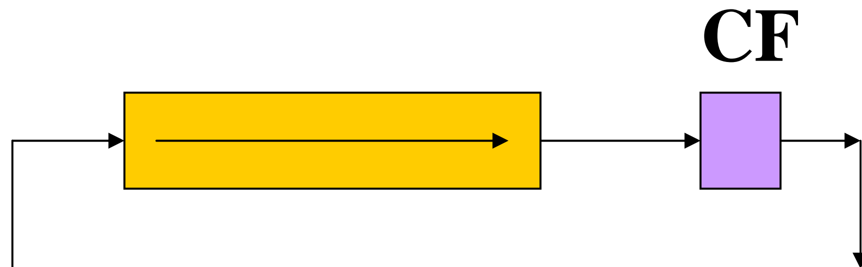
3) 循环右移

ROR DST，count



4) 带进位循环右移

RCR DST，count



- 说明
  - 不带CF位的循环移位有时也称为小循环；带CF位的称为大循环。
  - 如果只移一位，指令中可用立即数'1'指出移位的位数，如果超过1位，移位的位数必须事先放在CL中。
  - 循环移位指令操作结果只对CF和OF有影响
    - CF由最后一次移入CF的内容决定；
    - OF只在移位次数为1时有效，若移位后的符号位发生改变，则OF=1，否则为OF＝0。多位循环移位时，OF的值不确定。

四、字符串操作指令

- 字符串－存放在存储器中的一系列字或字节数据。
- 8086提供5条串操作指令
  1、MOVS——串传送指令
  2、CMPS——串比较指令
  3、SCAS——串扫描指令
  4、LODS——装入串指令
  5、STOS——存储串指令
- 80386以后增加了
  6、INS——串输入
  7、OUTS——串输出

串操作的隐含规定

1. 源串地址——DS:SI，允许使用段超越
2. 目的串地址——ES:DI，不许使用段超越。若源串和目的串在同一段，应将ES和DS相同。
3. 每执行一次串指令，SI和DI自动修改指针。
   - DF＝0/1，SI和DI自动＋n/－n
   - 当操作数类型为字节或字或双字时，n为1或2或4。
4. 串长度（字数或字节数或者双字数）放在CX中。每传送一次CX←(CX－1)；当CX＝0时，串操作结束。

5. 串操作的重复前缀

- **REP** 重复，直到**CX=0**
- **REPE/REPZ** 相等/为零则重复，直到**CX=0**
- **REPNE/REPNZ** 不相等/不为零则重复，直到**CX=0**

6. 串操作指令的不同形式

- 目的：表明操作对象是字节/字/双字
- 方法1：用指令中的操作数类型表明
- 方法2：指令助记符后面加**B/W/D**表明

① 串传送指令

- 格式：**MOVS 目的串，源串**

  或者：**MOVSB、MOVSW、MOVSD**

- 操作：目的串**ES:DI←**源串**(DS:SI)**

  SI←(SI+n/-n) ；修改源指针

  DI←(DI+n/-n) ；修改目的指针

  CX←(CX-1) ；重复次数－1

- 重复条件：一般配合前缀**REP**，直到**CX＝0**

- 例：将100个字节从开始地址为AR1单元区传送到开始地址为AR2的单元区。

```
LEA   SI，AR1  ；SI指向源串地址
LEA   DI，AR2  ；DI指向目的串地址
MOV   CX，100  ；置计数器
CLD            ；DF＝0（地址自动加1）
REP  MOVSB     ；开始传送，无需再指明
HLT            ；源串和目的串。
```

②串比较指令

- 格式：CMPS 目的串，源串
  或者：CMPSB、CMPSW、CMPSD
- 操作： 源串(DS:SI) －目的串ES:DI
  SI←(SI+n/-n)， DI←(DI+n/-n)
  CX←CX-1
- 重复条件：
  - 根据前缀REPE/REPZ，或者REPNE/REPNZ；
  - 或者遇到CX＝0
- 影响标志位： A、C、O、P、S， Z！

- 例：串String1和String2分别定义在DS段和ES段中，比较两串，如相等则转移到标号NEXT处。

```
        String1  DB 'HELP'          ；定义String1
        String2  DB 'HEPP'          ；定义String2
        ……
        CLD                         ；DF=0
        LEA   SI, String1           ；源串地址→SI
        LEA   DI, String2           ；目的串地址→DI
        MOV   CX, 4                 ；重复次数→CX
        REPZ  CMPSB                 ；重复比较，直到发现
                                    ；不同或者CX＝0
        JZ    NEXT                  ；串相同则转移
        ……
NEXT:
```

③ 字符串扫描指令

- 格式：**SCAS 目的串**
  或者：**SCASB、SCASW、SCASD**
- 操作："关键字"－目的串**ES:DI**、**DI←(DI+n/-n)**。
- 功能：从一个字串中查找一个与**AL/AX/EAX**中的内容（即所谓的"关键字"）相同或不同的字符。
- 重复条件：
  - 根据前缀**REPE/REPZ**或者**REPNE/REPNZ**
  - 或者遇到**CX＝0**
- 影响标志位：**A、C、O、P、S，Z！**

- 例:在串“That is CAI”中查找字符‘a’，找到，则转到标号FOUND处。

```
      String  DB  ‘That is CAI’    ；定义串
       ......
      CLD                          ；DF＝0
      LEA    DI, String            ；串地址→DI
      MOV    AL, ‘a’               ；关键字符’a’→ AL
      MOV    CX, 11                ；重复次数→ CX
      REPNZ SCASB                  ；重复扫描
      JZ     FOUND                 ；找到目的串元素转移
      ......
FOUND:
```

④ 字符串装入指令

- 格式：**LODS 源串**
  或者：**LODSB、LODSW、LODSD**
- 操作：**AL/AX/EAX←源串DS:SI**
  SI←(SI+n/-n)
  CX←(CX-1)
- 功能：将**DS:SI**的内容取到**AL/AX/EAX**中。
- 注意：由于**AL/AX/EAX**仅能保留一次装入的数据，所以该指令一般不使用重复前缀。

⑤ 字符串存储指令

- 格式：**STOS** 目的串
  或者：**STOSB、STOSW、STOSD**
- 操作：**AL/AX/EAX → 目的串ES:DI**
  DI←(DI+n/-n)
  CX←(CX-1)
- 功能：将**AL/AX/EAX**的内容存储到**ES:DI**指示的目的串单元中。
- **STOS**可以使用重复前缀吗？
  √ **STOS**可以与**REP**前缀连用，用一个放在**AL/AX/EAX**中的常数对一个数据块进行初始化。

⑥ 串输入指令INS

- 格式：INSB、INSW、INSD
- 操作：目的串ES:DI←[DX]

  DI←(DI+n/-n)

  CX←(CX-1)
- 功能：从[DX]寻址的端口输入一串数据到ES:DI为起址的一连串存储单元中。
- 可以使用重复前缀。

（386新增指令，参见教材P480）

⑦ 串输出指令OUTS

- 格式：OUTSB、OUTSW、OUTSD
- 操作：源串DS:SI→［DX］

  SI←(SI+n/-n)

  CX←(CX-1)
- 功能：将以DS:SI为起址的一连串存储单元中的内容输出到[DX]指明（寻址）的IO端口。
- 可以使用重复前缀。

（386新增指令，参见教材P480）

五 控制转移指令

- 作用：控制程序流程。可分为以下几类：

(一) 无条件转移和过程调用指令

(二) 条件转移指令

(三) 条件循环控制指令

(四) 中断指令以及中断返回指令

- 学习难点
  - 转移或者调用目的地址的寻址方式

(一) 无条件转移和过程调用指令

- **JMP** ； 无条件转移指令
- **CALL** ； 过程调用指令
- **RET** ； 过程返回指令

1. JMP无条件转移指令

- 格式：**JMP** 目的
- 功能：使程序无条件的转移到目的地址去执行。

- 两种类型
  - 段内转移或近转移：**JMP**指令与转移的目的地址在同一代码段，转移仅需改变**IP**。
  - 段间转移或远转移： **JMP**指令与转移的目的地址不在同一段中，转移时**CS**和**IP**都要改变。
- 两种方式
  - 直接方式：指令代码中直接给出目的地址。
  - 间接方式：目的地址包含在某个寄存器或存储单元中。

- 两种类型＋两种方式 ＝ 四种组合

（1）段内直接转移

（2）段内间接转移

（3）段间直接转移

（4）段间间接转移

(1) 段内直接转移

① 段内直接短转移：

JMP SHORT 标号	；	2字节指令

② 段内直接近转移：

	JMP NEAR PTR 标号	；	3字节指令
或	JMP 标号	；	

- 操作：CS不变；IP＝IP＋2（或＋3）＋disp
- 说明：
  - 编程只需确定短转还是近转，地址由汇编自动计算。
  - 短转是近转的特例，位移量disp为8位，转移范围-128字节至+127字节(以下条指令为基准)，偏移量用补码表示。
  - 近转位移量disp为16位，转移范围-32768字节至+32767字节(以下条指令为基准)，NEAR PTR可省略不写。

(2) 段内间接转移

- 格式：JMP WORD PTR OPR
- 操作：CS不变，IP←(EA)，EA由OPR除了立即数以外的任何寻址方式确定，如：JMP REG16
- 例一：JMP BX ；
  假设BX＝4500H，执行后程序转到CS:4500H处。
- 例二：

```
ADDRESS  DW  2000H        ；定义转移地址
   ...
LEA   SI, ADDRESS         ；偏移量→SI
   ...
JMP   WORD  PTR［SI］      ；转移到CS:2000H
```

(3) 段间直接(远)转移

- 格式：JMP FAR PTR OPR(标号) ；
- 操作：IP← 标号的段内偏移量

  CS← 标号所在的段基址

  其中：FAR PTR为段间转移操作符

- 例：假设标号PROG-F所在的段基址为3500H，偏移量为1234H。则执行

  JMP FAR PTR PROG-F

  之后，程序转移到3500H:1234H处继续执行。

(4) 段间间接转移

- 格式：JMP DWORD PTR［SI＋DISP］
- 操作：由(DS)×16＋SI＋DISP寻址连续4个字节的内存单元，其中前两个单元的内容→IP，后两个单元内容→CS。即:

  IP←[(DS)×16＋SI＋DISP]

  CS←[(DS)×16＋SI＋DISP＋2]

- 例：假设DS＝2500H，SI＝1300H，内存单元

  (26425H)＝4500H，(26427H)＝32F0H。则执行在指令

  JMP DWORD PTR［SI＋125H］

  以后，程序转移到32F0H:4500H处继续执行。

* 为什么不是从偶地址开始? 有许多指令的字长是奇数字节。

2. 过程调用与过程返回指令

- 过程（**Procedure**）——为完成某些特定功能而编写的相对独立的程序模块。也常被称为子程序（**Subroutine**）。
- 在主程序中使用过程调用指令**CALL**对过程进行调用。
- 过程以语句**PROC**开头，以语句**ENDP**结尾。在**ENDP**前要安排一条过程返回语句**RET**，使过程结束后能够正确返回主程序。
- 过程嵌套——在过程中又调用了另外一个过程。

- 近过程调用——过程调用指令和被调用的过程在同一代码段。
- 远过程调用——过程调用指令和被调用的过程不在同一代码段。
- CALL指令执行的两个步骤

  ① 将返回地址（紧跟CALL指令的下一条指令）IP压栈。近调用只需压栈IP；远调用需先将CS压栈，再将IP压栈。（教材P105～P106）

  ② 转到过程入口地址去执行被调用的过程。
- 与无条件转移指令相比，CALL指令增加了返回地址的压栈和弹出操作。调用过程入口地址的寻址方法与无条件转移指令基本相同

- 过程的调用方式
  - 根据被调用过程与**CALL**指令的相对位置关系以及过程入口地址的寻址方法，过程调用有以下**4**种方式

(1) 段内直接调用

(2) 段内间接调用

(3) 段间直接调用

(4) 段间间接调用

- 注意：没有段内短调用指令，因为考虑到被调过程不太可能在**CALL**指令的－**128**字节～**127**字节范围内。

- 过程的返回
  - 近调用返回：堆栈弹出一个字→**IP**，**SP+2→SP**
  - 远调用返回：堆栈先弹出一个字→**IP**，**SP+2→SP**；再弹出一个字→**CS**，**SP+2→SP**。

(1) 段内直接调用与返回

- 调用格式：CALL PROG_N
- 说明：PROG_N是近标号，也是子程序名。
- 操作：

  SP←SP－2

  返回地址IP压栈

  IP←IP＋DISP（汇编程序自动计算）

- 返回格式：RET
- 操作：IP出栈，IP ←（SP和SP＋1）单元内容，

  SP ← SP＋2

- 段内直接调用示例：（P106例3-83）
  - 调用前：CS＝2000H，IP＝1050H，SS＝5000H，SP＝0100H，PROG_F与CALL指令的之间的字节距离DISP为1234H。
  - 调用指令：　CALL PROG_F　；　（3字节）
  - 执行后：

  1）返回地址为1050H＋3＝1053H，因此应将1053H压栈

  2）过程入口地址应为1053H＋1234H＝2287H，所以程序转到CS:2287H处执行。

(2) 段内间接调用与返回

- 调用格式：CALL WORD PTR OPR
- 操作：SP←SP－2
  返回地址IP压栈
  IP←（EA）
- 说明：EA是由OPR的寻址方式确定的16位有效地址。例如以下两种形式：

```
CALL  BX                ；地址在寄存器中
CALL WORD PTR (BX＋SI)  ；地址被寻址的内存中
```

- 返回格式：RET
- 操作：与段内直接调用相同。

(3) 段间直接调用与返回—I

- 调用格式：CALL FAR PTR PROG_F
- 说明：
  - 被调用的过程与CALL指令不在同一个代码段。
  - PROG_F是远标号，也是子程序名。
  - 在程序中给出子程序名，就可以得到子程序的入口偏移地址(EA)及段基址。
- 操作：1）返回地址压栈

  SP←SP－2，CS压栈

  SP←SP－2，IP压栈

  2）转到过程入口地址

  IP ←子程序对应的偏移量（EA）

  CS←子程序所在的段基址

(3) 段间直接调用与返回—II

- 段间返回格式：**RET**
- 操作：

  1) 返回地址IP出栈：IP ←[(SP)+1，(SP)]

  SP ←(SP)+2

  2) 返回地址CS出栈：CS ←[(SP)+1，(SP)]

  SP ←(SP)+2

- 段间直接调用示例：（P107例3-85）
  - 调用指令：CALL FAR PTR PROG_F
  - 假设调用前： CS＝1000H，IP＝205AH；SS＝2500H ，SP＝0050H；标号PROG_F所在单元的地址指针： CS＝3000H，IP＝0050H
  - 该指令长度为5个字节，下条需要返回的指令地址为：CS＝1000H，IP＝205AH＋5＝205FH。因此堆栈段2500H:004EH←1000H，2500H:004DH←205FH。
  - 调用后程序转移到3000H:0050H处执行。
  - 过程执行完毕，RET指令顺序将IP和CS从堆栈中弹出，程序返回1000H:205FH继续执行，堆栈指针最后为2500H:0050H。

(4) 段间间接调用与返回—I

- **调用格式：CALL DWORD PTR OPR**
- 说明：
  - 被调用的过程与CALL指令不在同一个代码段。
  - 被调用的过程的入口地址（CS和IP）放在内存某处连续4个存储单元中。内存单元由OPR表明的寻址方式进行寻址。
  - 段间调用必须保存返回地址的IP和CS。
- 操作：1）返回地址压栈

  SP←SP－2，CS压栈

  SP←SP－2，IP压栈

(4) 段间间接调用与返回－II

2）转到过程入口地址

IP ← （EA）

CS ← （EA＋2）

EA为OPR寻址方式确定的内存单元有效地址。

- 段间返回格式：RET
- 操作：

1) 返回地址IP出栈: IP ← [(SP)＋1，(SP)]

SP ← (SP)＋2

2) 返回地址CS出栈: CS ← [(SP)＋1，(SP)]

SP ← (SP)＋2

- 段间间接调用示例：（P109例3-86）
  - 调用前：DS＝1000H，BX＝200H，(10200H)＝31F4H，(10202H)＝5200H
  - 调用指令：　　**CALL DWORD PTR [BX]**
  - 执行后：1）顺序将返回地址CS和IP压栈；
    2）转到过程入口地址
  - 根据调用指令的寻址方式可计算出存放过程入口地址的内存单元首地址为：DS×16＋BX＝10200H
  - 从10200H单元取得2个字节31F4H→IP
  - 从10200H＋2＝10202H取得2个字节5200H→CS
  - 执行完毕，RET指令顺序将IP和CS从堆栈中弹出，程序返回主程序继续执行。

返回的特别情况——带参数的返回指令

- 段内带参数返回：**RET n**
- 段间带参数返回：**RET n**
- 说明：
  - **n**称为弹出值。它使得**CPU**在弹出返回地址之后再从堆栈中弹出**n**个字节。**n**用表达式表示，其值可以是**0000H～FFFFH**范围内的任意一个偶数。
- 作用：
  - 主程序可以通过堆栈向被调用的过程传递传输，这些传输必须在调用前压入堆栈，过程在运行中通过堆栈指针取出这些参数。
  - 过程运行完毕，这些参数已失去作用，利用**RET n**可以顺便将这些传输从堆栈中弹出。