IBM-PC汇编语言程序设计

参考书目：沈美明，温冬婵.《IBM-PC汇编语言程序设计》，清华大学出版社。

第1章 基础知识

1. 二进制数、十进制数和十六进制数

2. ASCII码和**BCD**码

1. 二进制数、十进制数和十六进制数

- 二进制数和十进制数之间的转换
  - 二进制数转换为十进制数
    - 例：**1011100.10111B = $2^6 + 2^4 + 2^3 + 2^2 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5}$**

      = 92.71875D
  - 十进制数转换为二进制数
    - 降幂法 **(…, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, …)**

      例如：**117.8125D = 64 + 32 + 16 + 4 + 1 + 0.5 + 0.25 + 0.625**

      = $2^6 + 2^5 + 2^4 + 2^2 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$

      = 1110101.1101B
    - 除法

      例如：**N = 117.8125D**

      117D: 117/2 = 58/2 = 29/2 = 14/2 = 7/2 = 3/2 = 1/2 = 0

      1 0 1 0 1 1 1 ←

      0.8125D: 0.8125*2 = 0.625*2 = 0.25*2 = 0.5*2 = 1

      → **1 1 0 1**

      结果：**N=117.8125D=1110101.1101B**

- 十六进制数与二进制数的转换
  - 例：**011010110111111B = 011,0101,1011,1111 = 35BFH**

    A19CH = 1010,0001,1001,1100 = 1010000110011100B
- 十六进制数与十进制数的转换
  - 例：**BF3CH = 11×16^3+15×16^2+3×16^1+12×16^0 = 48956D**
  - 十进制转换为十六进制
    - 降幂法

      例：**48956D = 11×4096 + 15×256 + 3×16 + 12**

      = 11×16^3 + 15×16^2 + 3×16^1 + 12×16^0 = BF3CH
    - 除法

      例：**48956D　48956/16 = 3059/16 = 191/16 = 11/16 = 0**

      12(C)　3　15(F)　11(B)

      结果：**48956D = BF3CH**

2. ASCII码和BCD码

- **ASCII码**
  - 计算机处理的信息，除数字外，还有字符和字符串等，计算机中，字符一般采用美国信息交换标准代码**ASCII**码来表示。
  - 特点：用**1byte**表示一个字符，其中，低**7b**为字符的**ASCII**值，最高位**D7 = 0**。

    （ **’0’~ ’9’: 30~39H;**

    ‘A’~ ’Z’: 41~5AH;

    ‘a’ ~ ’z’: 61~7AH）

NUL	00	4	34	M	4D	f	66
BEL	07	5	35	N	4E	g	67
LF	0A	6	36	O	4F	h	68
FF	0C	7	37	P	50	i	69
CR	0D	8	38	Q	51	j	6A
SP	20	9	39	R	52	k	6B
!	21	:	3A	S	53	l	6C
”	22	;	3B	T	54	m	6D
#	23	<	3C	U	55	n	6E
$	24	=	3D	V	56	o	6F
%	25	>	3E	W	57	p	70
&	26	?	3F	X	58	q	71
’	27	@	40	Y	59	r	72
(	28	A	41	Z	5A	s	73
)	29	B	42	[	5B	t	74
*	2A	C	43	\	5C	u	75
+	2B	D	44	]	5D	v	76
,	2C	E	45	↑	5E	w	77
—	2D	F	46	←	5F	x	78
.	2E	G	47		60	y	79
/	2F	H	48	a	61	z	7A
0	30	I	49	b	62	{	7B
1	31	J	4A	c	63	|	7C
2	32	K	4B	d	64	}	7D
3	33	L	4C	e	65	~	7E

BCD码

- BCD码指用二进制编码表示的十进制数（用4b/ 8b二进制数来表示一个十进制数）。又称为二一十进制数，或8421码。
- 表示：

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

- 类型：
  - 在IBM-PC机中，表示十进制数的BCD码可以用压缩的BCD码和非压缩的BCD码两种格式来表示。

    压缩BCD码：用4b二进制数表示1个十进制数。

    非压缩BCD码：用8b二进制数表示1个十进制数（高4b无定义，低4b是以8421的BCD码）。
  - 例如：9502D 1001 0101 0000 0010

    uuuu1001 uuuu0101 uuuu0000 uuuu0010
  - ‘0’～‘9’的ASCII码为非压缩的BCD码：高4b为0011，低4b为8421码。

第2章 寻址方式和指令系统

- 汇编语言指令由操作码和操作数构成，格式：

  【标号:】 操作码 【目的操作数】【,源操作数】 【；注释】

  - 说明：
    - 标号：字母开头的，字母和数字构成的字符串，具有唯一性。用于确定程序中特定位置。
    - 操作码：指示计算机所要执行的操作，必不可少。
    - 目的/源操作数：指令执行过程中所需要的操作数。不同的操作码所对应的操作数数目会不同（0个，1个，2个）
    - 注释：从“；”起到回车符之前均为注释，对程序起解释作用。
  - 例如：

```
        MOV CX , 100        ；循环次数
        MOV AX, 200
AGAIN:  DEC AX
        PUSH AX
        LOOP AGAIN
        …
```

2.1 寻址方式

- 汇编语言指令中用到的操作数，可能是具体的数据（立即数）、也可能是存放在寄存器或内存中的数据或地址。
- 寻址方式是为得到指令中操作数据而对操作数采用的表示方式。
- **IBM PC机的寻址方式：**
  - 立即寻址方式
  - 寄存器寻址方式
  - 直接寻址方式
  - 寄存器间接寻址方式
  - 寄存器相对寻址方式
  - 基址变址寻址方式
  - 相对基址变址寻址方式

2.1.1 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。
- 立即数可以是**8b**或**16b**的。若为**16b**数，则高字节数据放较高地址单元，低字节数据放在较低地址单元中。
- 例:

```
MOV AL , 25
  ;指令执行后(AL)=19H
MOV AX , 2076H
  ;指令执行后(AX)=2076H
MOV AL , 'A'
  ;指令执行后(AL)=41H
```

- 说明：该寻址方式常用于给寄存器赋初值，并且只能用于源操作数，不用于目的操作数。

2.1.2 寄存器寻址方式

- 操作数放在寄存器中，指令中指定寄存器号。
- 对**16b**操作数，寄存器可以是：**AX, BX, CX, DX, SI, DI, SP**和**BP**等；对**8b**操作数，寄存器可以是：**AL, AH, BL, BH, CL, CH, DL**和**DH**。
- 例：

```
MOV AX, BX
; (AX)←(BX), (BX)保持不变。
MOV DL, 25H
  ；指令执行后(DL)=25H
```

- 说明：操作数放在寄存器中，无需服务存储器来得到操作数，因而可以取得较高运算速度。

以下寻址方式中，操作数放在代码段以外的存储区中，通过不同方式求得操作数地址，从而得到操作数。

2.1.3 直接寻址方式

- **IBM PC机中将操作数的偏移地址称为有效地址EA，直接寻址方式中，EA就在指令中，此时物理地址＝16×(段寄存器)+EA，默认段为DS，若为其它段，则应在指令中指定段跨越前缀。**
- 例：

```
MOV AX, [1000H]          ;注意：1000H为EA
```

若(DS)=3000H，地址为31000H的字定义内容为6350H，则程序执行后(AX)=6350H。

```
MOV AX, VALUE            ; 等效于 MOV AX, [ VALUE ]
MOV AX, ES:[1234H]       ; (AX) ←(16×(ES)+1234H)
MOV AX, ES: VALUE        ; 等效于 MOV AX, ES: [ VALUE ]
```

2.1.4 寄存器间接寻址方式

- 操作数有效地址**EA在BX, BP或SI, DI中**，操作数在存储器中。
  - 若指定的寄存器为**BX, SI或DI**, 则操作数默认在**DS**中，物理地址＝**16×(DS) + (SI / DI / BX)**
  - 若指定为**BP**，则操作数默认在**SS**中。物理地址＝**16×(SS) + (BP)**
  - 若操作数不在默认段中，则应在指令中指定段前缀。
- 例：

  MOV AX, [BX]

  如果**(AX)=1234H, (DS)=3000H, (BX)=1000H, (31000H)=5678H**, 则物理地址＝**31000H**，指令执行后**(AX)=5678H**。

  MOV AX, ES:[BX] **;(AX) ← (16×(ES) + (BX))**
- 此寻址方式一般用于数组或表格处理，执行完**1**条指令后修改寄存器内容就可取出表格中的下一项。

2.1.5 寄存器相对寻址方式

- 此方式下，操作数的有效地址**EA**和物理地址计算：
  - 有效地址**EA**＝ **(BX/BP/SI/DI) + 8b/16b**偏移量
  - 物理地址＝**16**×**(DS) + (BX/SI/DI)+8b/16b**偏移量

    或物理地址＝**16**×**(SS) + (BP) + 8b/16b**偏移量
- 例：

  MOV AX, 8[SI] ；也可写成 **MOV AX, [8+SI]**

  如果**(DS)=3000H, (SI)=2000H, (32008H)=1234H,** 则物理地址＝**32008H,**指令执行后**(AX)=1234H.**

  MOV DL, ES:STR[SI] **;(DL)←(16×(ES)+STR+(SI))**
- 该指令可用于表格处理，表格首地址设置为固定值，利用修改寄存器内容来取得表格中的值。

2.1.6 基址变址寻址方式

- 此方式下，操作数的有效地址**EA**和物理地址计算：
  - 有效地址**EA = (**基址寄存器**)**＋**(**变址寄存器**)**
  - 物理地址＝**16**×**(DS) + (BX) + (SI/DI)**

    或物理地址＝**16**×**(SS) + (BP) + (SI/DI)**
- 例:

  MOV AX, [BX][SI] ;或写成: **MOV AX, [BX+SI]**

  如果**(DS)=3000H, (BX)=1200H, (SI)=4100H,(35300H) =1234H,**
  则物理地址**=35300H,**指令执行后**(AX)=1234H**

  MOV AX, ES:[BX][SI] ; **(AX)←(16×(ES)+(BX)+(SI))**
- 同样适用于数组或表格处理，首地址放于基址寄存器，而用变址寄存器访问数组中各个元素，较直接变址方式灵活

2.1.7 相对基址变址寻址方式

- 此方式下，操作数的有效地址**EA**和物理地址计算：
  - 有效地址**EA = (**基址寄存器**) + (**变址寄存器**) + 8b/16b**偏移量
  - 物理地址　＝**16×(DS) + (BX) + (SI/DI) + 8b/16b**偏移量
    或物理地址＝**16×(SS) + (BP) + (SI/DI) + 8b/16b**偏移量
- 例：

  MOV ax, COUNT[BP][DI]　；或写成 **MOV AX, COUNT[BP+DI]**
  - 如果**(SS)=3000H,(BP)=1000H,(DI)=2000H,COUNT=2100H, (35100H) =1234H,**则物理地址＝**35100H**，指令执行后**(AX)= 1234H.**
- 方便了堆栈处理：**BP**指向栈顶，从栈顶到数组首地址用偏移量表示，变址寄存器用来访问数组中某个元素。

2.2 IBM PC机指令系统

- **8086/8088**指令系统按功能分为**6**类：
  - 数据传送指令
  - 算术运算指令
  - 逻辑运算指令
  - 串操作指令
  - 控制转移指令
  - 处理器控制指令

2.2.1 数据传送指令

- 通用数据传送指令
  - **MOV指令；PUSH指令；POP指令；XCHG指令**
- 累加器专用传送指令
  - **IN指令；OUT指令；XLAT指令**
- 地址传送指令
  - **LEA指令；LDS指令；LES指令**
- 标志寄存器传送指令
  - **LAHF指令；SAHF指令；PUSHF指令；POPF指令**
- 类型转换指令
  - **CBW指令；CWD指令**

1.通用数据传送指令（MOV; PUSH,POP; XCHG）

① MOV指令

- 指令格式：**MOV DST, SRC ； DST和SRC为8b或16b**
- 执行操作：**(DST)←(SRC)**
- 注意：
  - **MOV指令不影响PSW；**
  - **DST和SRC的数据位数必须相同；MOV AX, BL ；×**
- 常用形式（**7**种）：
  - **MOV mem/reg1, mem/reg2**

    例：**MOV AX, BX　　;(AX) ← (BX)**

    MOV DAA, AX　　;(DAA) ← (AX)，传送1个字

    MOV DL, DAA　　;(DL) ← (DAA)，传送1个字节

    注意：**2个操作数不能同时为存储器操作数，不允许用段寄存器。**

- **MOV reg, data**

  例：　**MOV AL, 20H　　;(AL) ← 20H**

  　　MOV DX, 0FE20H　;(DX) ← 0FE20H, A~F作数头前加0

- **MOV mem, data**

  例:　**MOV DAA1,200H　; (DAA1) ← 200H**

- **MOV ac, mem**

  例:　**MOV AX, DATA_SEG**

  　　MOV AL, DAA1　　; (AL)←(DAA1)

  　　MOV AX, DAA2　　; (AX)←(DAA2)

  上述**DAA1**必须为单字节存储单元, **DAA2**必须为字存储单元

  　　MOV AX, Y[BP][SI]　; (AX)←(16×(SS)+(BP)+(SI)+Y)

- **MOV mem, ac**

- **MOV segreg, mem/reg**

例： **MOV SS, BX** ; (SS)←(BX)

MOV DS, AX ; (DS)←(AX)

注意:**(1) segreg不能是CS;**

(2)不允许将段地址直接传送给DS, 必须通过寄存器传送；

MOV DS，DS_SEG ×

MOV AX，DS_SEG ;正确方法

MOV DS，AX

(3)该指令执行完后不响应中断，要等到下一条指令执行完后才可能响应中断。

- **MOV mem/reg, segreg**

例: **MOV DAA1, DS** ; (DAA1)←(DS)

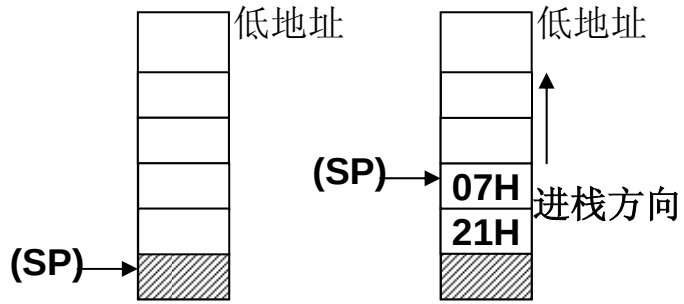
② 堆栈操作指令（PUSH，POP）

- **PUSH指令（进栈）**
  - 指令格式：**PUSH SRC ；**
  - 执行操作：**(SP) ← (SP)-2,**
    ((SP)+1,(SP)) ← (SRC)
- **POP指令（出栈）**
  - 指令格式：**POP DST ；**
  - 执行操作：**(DST)←((SP)+1, (SP)),**
    (SP) ← (SP)+2
- 说明：

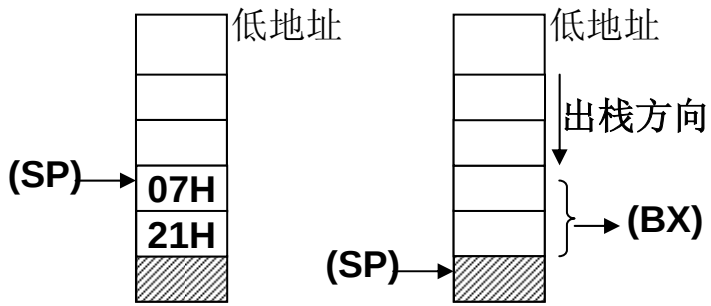
(1)两指令中的操作数必须以字为单位，且不能是立即数；

(2)指令可以指定段寄存器作为操作数，但**POP**指令不允许用**CS**寄存器；

(3)两指令不影响标志寄存器**PSW**。



执行PUSH AX；（AX）=2107H

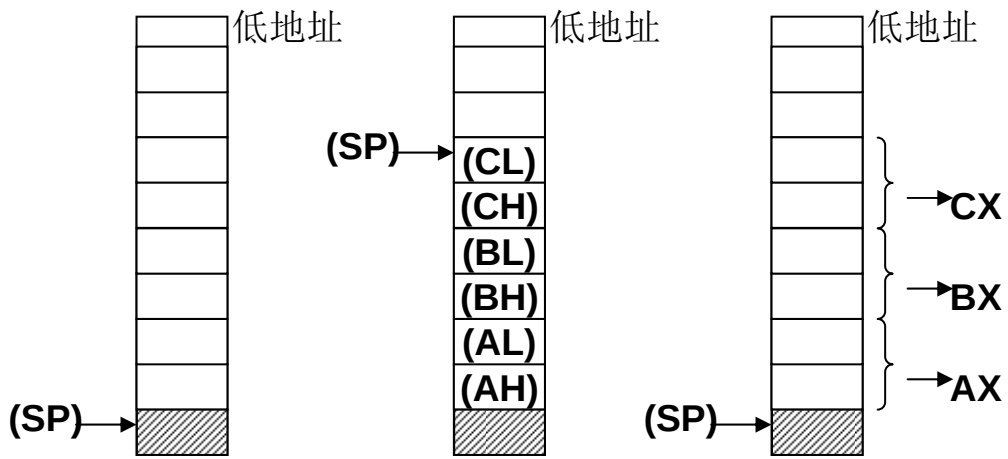


执行POP BX；（BX）=2107H

– 用途：

- 如果在程序中用到某些寄存器，但它的内容却在将来还有用，这时就可以用堆栈将它们保存，然后在必要时再恢复其内容。
- 例：

```
PUSH AX
PUSH BX
PUSH CX
…
POP CX
POP BX
POP AX
```



注意：进栈多个数据时，出栈和进栈时顺序相反。

③ XCHG 交换指令

- 指令格式：**XCHG OPR1，OPR2**
- 执行操作：**(OPR1) ↔ (OPR2)**
- 例：**XCHG BX，[BP+SI]**
  - 如指令执行前：**(BX) = 6F30H, (BP) = 0200H, (SI) = 0046H,**
    (SS)= 2F00H, (2F246H) = 4154H

    则指令执行后，**(BX) = 4154H，(2F246H)= 6F30H**
  - **MOV AL, DA_BYTE1** ;两个存储单元之间的数据交换

    XCHG AL, DA_BYTE2

    XCHG AL, DA_BYTE1
- 指令说明：
  - 该指令的两个操作数中必须有一个在寄存器中，但不允许使用段寄存器
  - 指令允许字/字节操作，且不影响**PSW**中标志位

2.累加器专用传送指令（IN,OUT; XLAT）

I/O操作指令（IN，OUT）不影响PSW.

PORT为≤0FFH的端口地址

- IN指令 （从I/O端口读）
  - 指令格式：IN AL，PORT（字节） ；操作：(AL) ← (PORT)
    IN AX，PORT（字） ；操作：(AX) ←((PORT+1,PORT)
    IN AL, DX（字节） ；操作： (AL) ← ((DX));
    IN AX, DX（字） ；操作：(AX) ← ((DX)+1,(DX))
  - 例：(1) IN AX, 28H
    MOV DATA_WORD, AX
    (2) MOV DX, 3FCH
    IN AX, DX
- OUT指令 （向I/O端口写）
  - 指令格式：OUT PORT , AL（字节） ；操作：(PORT) ← (AL) ;
    OUT PORT, AX（字） ；操作：(PORT+1,PORT) ← (AX)
    OUT DX , AL（字节） ；操作：((DX)) ← (AL);
    OUT DX , AX （字） ；操作：((DX)+1,(DX)) ← (AX)
  - 例：(1) MOV AX , DATA_WORD
    OUT 28H , AX
    (2) MOV DX, 3FCH
    OUT DX, AX

XLAT 换码指令

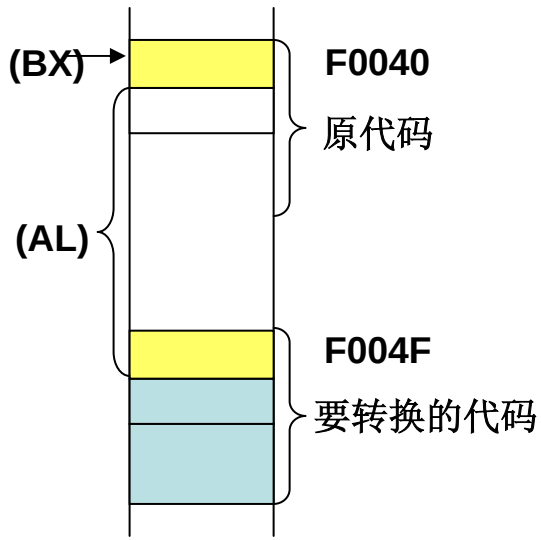
- 指令格式：**XLAT OPR**

  XLAT

- 执行操作：**(AL) ← ((BX)+(AL))**
- 用途：常用于将一种代码转换为另一种代码。
- 例:如（**BX)=0040H, (AL)=0FH, (DS)=F000H,**则执行

  XLAT

  后　**(AL)=(F0000+0040+0F)**



3.地址传送指令（LEA, LDS, LES）

- **LEA** 有效地址送寄存器
  - 指令格式：**LEA REG, SRC**
  - 执行操作：**(REG) ← SRC**　　;将原操作数地址送到指定**REG**。
  - 例：**LEA BX,[BX+SI+0F62H]**

    如执行前 **(BX)=0400H, (SI)=003CH**

    则执行后 **(BX)=0400+003C+0F62H=139EH; (BX)**为偏移地址
- **LDS** 指针送寄存器和**DS**指令
  - 指令格式：**LDS REG, SRC**
  - 执行操作: **(REG) ← (SRC);**　　**(DS) ← (SRC+2)**
  - 例：　**LDS SI, [10H]**

    如执行前 **(DS)=C000H, (C0010H)=0180H, (C0012H)= 2000H**

    则执行后 **(SI)=0180H, (DS) = 2000H**

– **LES 指针送寄存器和ES指令**

- 指令格式：**LES REG, SRC**
- 执行操作: **(REG) ← (SRC);　　(ES) ← (SRC+2)**
- 例：　**LES DI, [BX]**

  如执行前 **(DS)=B000H, (BX)=080AH, (0B080AH)=05AEH,**

  (0B080CH)= 4000H

  则执行后 **(DI)=05AEH, (ES) = 4000H**

– 说明：

- 以上**3**条指令中的**REG**不能用段寄存器，且**SRC**必须使用除立即寻址方式和寄存器方式以外的其它寻址方式。
- 这些指令不影响**PSW**。

4. 标志寄存器FLAG传送指令

- **LAHF** 标志送**AH**指令（不影响标志位）
  - 指令格式：**LAHF** ；操作：**(AH) ← (PSW**低字节**)**
- **SAHF AH**送标志寄存器指令
  - 指令格式：**SAHF** ；操作：**(PSW**低字节**) ←(AH)**
- **PUSHF** 标志进堆栈指令（不影响标志位）
  - 指令格式：**PUSHF** ；操作：**(SP) ← (SP) - 2**
    ((SP)+1, (SP)) ← (PSW)
- **POPF** 标志出堆栈指令
  - 指令格式：**POPF** ；操作：**(PSW) ← ((SP)+1, SP)**
    (SP) ← (SP)+2
- 其它与**FLAG**有关的指令:
  - **CLD** ;复位方向标志**DF**＝**0**
    STD ;设置方向标志**DF**＝**1**
  - **CLI** ;复位可屏蔽中断允许标志**IF**＝**0**
    STI ;设置可屏蔽中断允许标志**IF**＝**1**
  - **CLC** ;复位进位标志**CF**＝**0**
    STC ;设置进位标志**CF**＝**1**
    CMC ;进位标志**CF**取反。**CF** ← $\overline{\text{CF}}$

5.类型转换指令

- **CBW** 字节转化为字
  - 指令格式：**CBW　　；将AL内容转换为AX**
  - 执行操作：**AL的内容符号扩展到AH。如(AL)的最高有效位为0，则(AH)=00；如(AL)最高有效位为1，则(AH)=0FFH**

- **CWD** 字转化为双字
  - 指令格式：**CWD　　；将AX内容转换为DX,AX**
  - 执行操作：**AX的内容符号扩展到DX。如(AX)的最高有效位为0，则(DX)=0000；如(AX)最高有效位为1，则(DX)= 0FFFFH**

2.2.2 算术运算指令

- 加法指令
  - **ADD，ADC，INC**
- 减法指令
  - **SUB，SBB，DEC，NEG，CMP**
- 乘法指令
  - **MUL，IMUL**
- 除法指令
  - **DIV，IDIV**
- 十进制调整指令
  - 压缩**BCD**码调整指令（**DAA，DAS**）
  - 非压缩**BCD**码调整指令（**AAA，AAS，AAM，AAD**）

1.加法指令

– ADD 加法

- 指令格式：**ADD DST, SRC (8/16b)**
- 执行操作: **(DST) ← (SRC)+(DST)**
- 说明：该指令的执行结果将影响**PSW**中**ZF,CF,OF,SF**

  无符号数溢出: 用**CF**判断**(CF=1)**;

  有符号数溢出: 用**OF**判断**(OF=1)**（两操作数符号相同，相加后符号相反，则**OF=1**,否则**OF=0**）
- 例：**ADD DX,0F0F0H**

  如指令执行前**(DX)=4652H**，则执行后

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

```
    0100 0110 0101 0010
+   1111 0000 1111 0000
-------------------------
1↙  0011 0111 0100 0010
```

– ADC 带进位的加法

- 指令格式：**ADC DST, SRC (8/16b)**
- 执行操作：**(DST) ← (SRC)+(DST)+CF**
- 说明： 同**ADD**指令（影响**PSW**中**ZF,CF,OF,SF** ）
- 例：执行**2**个双精度数加法。设目的操作数存放在**DX**和**AX**，其中**DX**存放高位字；源操作数存于**BX**和**CX**，**BX**存放高位字。

ADD AX, CX

ADC DX, BX

设执行前**(DX)=0002H, (AX)=0F365H,**

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

```
   F365          0002
+  E024          0005
-------       +     1(CF)
1  D389       -------
                 0008
```

则执行后：**(DX)=0008H,(AX)=D389H,SF=0,ZF=0,CF=1,OF=0**

– INC 加1指令

- 指令格式**: INC OPR(8/16b)**
- 执行操作：**(OPR) ← (OPR)+1**
- 说明： 同**ADD**指令，但不影响**CF**标志。

2.减法指令

– SUB 减法

- 指令格式：**SUB DST, SRC**
- 执行操作：**(DST) ← (DST) – (SRC)**
- 说明：该指令的执行结果影响**PSW**中的**ZF, CF, OF**和**SF**。

  若减数＞被减数则**CF=1**，否则**CF=0**；

  若两数符号相反，而结果符号与减数相同则**OF=1**,否则**OF=0**;
- 例： **SUB DH, [BP+4]**

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

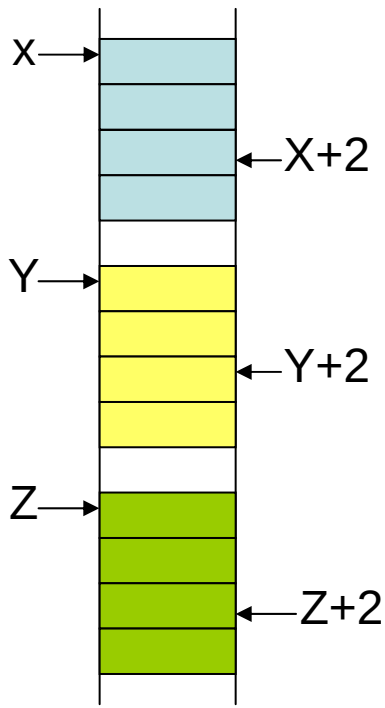
  在执行后，**(DH)=0E7H, SF=1,ZF=0,CF=1,OF=0**

```
   41              0100 0001              0100 0001
 − 5A      ⇨   −   0101 1010      ⇨   +   1010 0110
-------          -----------            -----------
                                        1110 0111
```

– **SBB** 带借位减法指令

- 指令格式：**SBB DST, SRC**
- 执行操作：**(DST) ← (DST) - (SRC) - CF**
- 说明：同**SUB**指令
- 例：设**X, Y, Z**均为双精度数，它们分别存放在地址**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，实现：**W ← X + Y + 24 – Z** 。结果存于 **W** 和 **W+2** 单元。

```
MOV   AX, X        ;
MOV   DX, X+2      ;        } X+Y
ADD   AX, Y        ;
ADC   DX, Y+2      ;
ADD   AX, 24  ;
ADC   DX, 0        ;        } + 24
SUB   AX, Z        ;        } –Z
SBB   DX, Z+2;
MOV  W, AX         ;        } 结果存入W, W+2
MOV  W+2, DX       ;
```



- **DEC 减1指令**
  - 指令格式：**DEC OPR**
  - 执行操作：**(OPR) ← (OPR) - 1**
  - 说明：同**SUB**指令,但不影响**CF**标志。
- **NEG 求补指令**
  - 指令格式：**NEG OPR**
  - 执行操作：**(OPR) ← －(OPR)**

    或　**(OPR) ← 0FFFFH – (OPR) +1**
  - 说明：影响**CF, SF, ZF, OF**。

    只有当**OPR=0**时结果使**CF=0**,其它情况下**CF=1.**
- **CMP指令**
  - 指令格式：**CMP OPR1, OPR2**
  - 执行操作：**(OPR1) – (OPR2)**
  - 说明：与**SUB**指令不同处在于不保存结果。后面常跟**1**条条件转移指令，根据比较结果产生不同的程序分支。

3.乘法指令

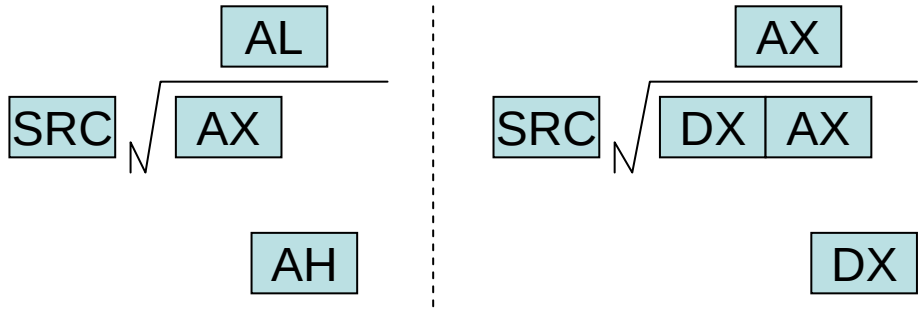
- **MUL** 无符号数乘法
  - 指令格式：**MUL SRC**　　；被乘数放在**AL**或**AX**中
  - 执行操作：字节操作　　**(AX) ← (AL) ＊ (SRC)**

    字操作　　**(DX, AX) ← (AX) ＊(SRC)**
  - 说明：如果执行结果的**AH**（或**DX**）为**0**，则**CF=OF=0,**否则均为**1**；其它标志位状态无法确定。
  - 例: 如**(AL) = 0B4H, (BL) = 11H,** 则执行：**MUL BL**

    0B4H－180D,11H－17D, (AX)=0BF4H=3060D.　CF=OF=1
- **IMUL** 带符号数乘法
  - 指令格式：**IMUL SRC**
  - 执行操作：与**MUL**指令相同，但必须是带符号数。
  - 说明：同**MUL**指令
  - 例：如**(AL) = 0B4H, (BL) = 11H,** 则执行：**IMUL BL**

    0B4H－－76D, 11H－17D, (AX) = 0FAF4H = -1292D. CF=OF=1

4.除法指令



- **DIV** 无符号数除法
  - 指令格式：**DIV SRC**
  - 执行操作:
    - 字节操作:执行前将 **(AX) ← 16b**被除数，**(SRC) ← 8b**除数;
      执行结果：**8b**商在**AL**中，**8**位余数在**AH**中。表示为：
      (AL) ← (AX) / (SRC)的商；
      (AH) ← (AX) / (SRC)的余数
    - 字操作:执行前将 **(DX, AX) ← 32b**被除数， **(SRC) ← 16b**除数;
      执行结果：**16b**商在**AX**中，**16**位余数在**DX**中。表示为：
      (AX) ← (DX,AX) / (SRC)的商；
      (DX) ←(DX,AX) / (SRC)的余数
  - 说明：对所有条件码无定义。
  - 例: 如**(AX) = 0400H, (BL) = 0B4H,** 则执行：**DIV BL**

0400H－1024D,0B4H－180D, (AH)=7CH=124D, (AL)=05H=5D

– **IDIV** 带符号数除法

- 指令格式：**IDIV SRC**
- 执行操作：与**DIV**指令相同，但必须是带符号数，商和余数均为带符号数，且余数符号和被除数符号相同。
- 说明：同**DIV**指令
- 例：如**(AX) = 0400H, (BL) = 0B4H,** 则执行：**IDIV BL**

  0400－1024D, 0B4H－－76D; (AH)= 24H = 36D,(AL)=0F3H=－13D

– 例：计算 **(V - (X * Y + Z – 540)) / X**

其中 **X, Y, Z, V** 均为**16**位带符号数，已分别装入 **X, Y, Z, V** 单元中，要求上式计算结果的商存入**AX**，余数存入**DX.**

```
MOV   AX, X
IMUL  Y          DX | AX
MOV   CX, AX
MOV   BX, DX  ;(BX,CX)=X*Y
MOV   AX, Z
CWD
ADD   CX, AX
ADC   BX, DX  ;(BX,CX)=X*Y+Z
SUB   CX, 540
SBB   BX, 0 ;(BX,CX)= X*Y+Z-540
MOV   AX, V
CWD
SUB   AX, CX
SBB   DX, BX ;(DX,AX)=V- (X*Y+Z-540)
IDIV  X    ;(DX):余数，(AX):商
```

5.十进制调整指令

- 压缩BCD码调整指令
  - DAA 加法的十进制调整指令
    - 指令格式：DAA
    - 执行操作：(AL) ←把AL中的和调整到压缩的BCD格式。该指令执行之前必须执行ADD或ADC指令，加法指令必须把两个压缩的BCD码相加，并把结果存放在AL中。
    - 说明：该指令对OF无定义，但影响所有其它条件标志。
    - 例：(1) ADD AL, BL
      DAA

      CF=0, AF=1

      如执行前(AL)=28, (BL)= 68, ADD后(AL)=90, DAA后(AL)=96

      (2) 如(BCD1)=1834，(BCD2)=2789,要求编程序实现：

      (BCD3) ←(BCD1)+(BCD2)

```
MOV     AL, BCD1  ;(AL)=34
ADD     AL, BCD2  ;34+89
DAA     ;(AL)=23,(CF)=1
MOV     BCD3, AL
```

```
MOV AL, BCD1+1; (AL)=18
ADC AL,  BCD2+1; 18+27+1
DAA       ;(AL)=46
MOV BCD3+1, AL
```

– **DAS 减法的十进制调整指令**

- 指令格式：**DAS**
- 执行操作：**(AL) ←把AL中的差调整到压缩的BCD格式。**

  该指令执行之前必须执行**SUB**或**SBB**指令，减法指令必须把两个压缩的**BCD**码相减，并把结果存放在**AL**中。
- 说明：同**DAA**指令。
- 例：**(1)**

```
SUB AL, AH
DAS
```

CF=0, AF=1

如执行前**(AL)=86, (AH)= 07, SUB**后**(AL)=7F, DAS**后**(AL)=79**

(2) 如**(BCD1)=1234，(BCD2)=4612,**要求编程序实现：

(BCD3) ←(BCD1)－(BCD2)

```
MOV     AL, BCD1  ; (AL)=34
SUB     AL, BCD2  ; 34-12
DAS               ; (AL)=22, CF=0
MOV     BCD3, AL
```

```
MOV AL, BCD1+1  ; (AL)=12
SBB AL,  BCD2+1  ; 12-46-0
DAS
MOV BCD3+1, AL
```

- 非压缩**BCD**码调整指令
  - **AAA** 加法的**ASCII**调整指令
    - 指令格式：**AAA**
    - 执行操作：**(AL)** ← 把**AL**中的和调整到非压缩的**BCD**格式。

      (AH) ← **(AH)**＋调整产生的进位值

      该指令执行之前必须执行**ADD**或**ADC**指令，加法指令必须把两个非压缩的**BCD**码相加，并把结果存放在**AL**中。
    - 说明：**AAA**指令影响**AF**和**CF**标志，对其余标志位无定义。
    - 例：
      ```
      ADD AL, BL
      AAA
      ```

      如执行前**(AX)=0535H, (BL)=39H, AL**和**BL**中分别为**5**和**9**的**ASCII**码。**ADD**后**(AL)=6E, DAS**后**(AX)=0604H**

      CF=0, AF=1　　**CF=1, AF=1**

– **AAS 减法的ASCII调整指令**

- 指令格式：**AAS**
- 执行操作：**(AL) ← 把AL中的差调整到非压缩的BCD格式。**

  (AH) ← (AH)－调整产生的借位值

  该指令执行之前必须执行**SUB**或**SBB**指令，减法指令必须把两个非压缩的**BCD**码相减，并把结果存放在**AL**中。
- 说明：**AAA指令影响AF和CF标志，对其余标志位无定义。**
- 例：编程序实现：**(DX) ← UP1 + UP2 - UP3**

  其中参加运算的数均为二位十进制数。

```
MOV     AX, 0
MOV     AL, UP1
ADD     AL, UP2
AAA
MOV     DL, AL
MOV     AL, UP1+1
ADC     AL, UP2+1
AAA
```

```
XCHG    AL, DL
SUB     AL, UP3
AAS
XCHG    AL, DL
SBB     AL, UP3+1
AAS
MOV     DH, AL
```

- **AAM 乘法的ASCII调整指令**
  - 指令格式：**AAM**
  - 执行操作：
    - **(AX) ← 把AL中的积调整到非压缩的BCD格式。**
    - 该指令执行之前必须执行**MUL**指令，把两个非压缩的**BCD**码相乘，并把结果存放在**AL**中。
  - 说明：影响**SF, ZF**和**PF**，对**OF, CF**和**AF**无定义。
  - 例：
```
MUL AL, BL
AAM
```
如指令执行前**(AL)=07H, (BL)=09H.**

MUL后**(AL)=3FH**，**AAM**后**(AH)=06H, (AL)=03H**。

- **AAD 除法的ASCII调整指令**
  - 指令格式：**AAD**
  - 执行操作：
    - 如果被除数是放在**(AX)**中的**2**位非压缩**BCD**码（**AH**中存放十位数，**AL**中存放个位数），且**AH**和**AL**中高**4b**均为**0**；除数是**1**位非压缩**BCD**码，且高**4b**均为**0**；**(AX)** ← 把**AL**中的和调整到非压缩的**BCD**格式。
    - 把两个数用**DIV**指令相除之前，必须先用**AAD**指令把**AX**中的被除数调整成二进制，并存放在**AL**寄存器中。
  - 说明：影响**SF, ZF**和**PF**，对**OF, CF**和**AF**无定义。
  - 例： **AAD**

    如指令执行前**(AX)=0604H**，

    则指令执行后**(AX)=0040H**。

2.2.3 逻辑指令

- 逻辑运算指令
  - **AND** 逻辑与
    - 指令格式：**AND DST, SRC** ；操作：**(DST) ←(DST) ∧(SRC)**
  - **OR** 逻辑或
    - 指令格式：**OR DST, SRC** ；操作：**(DST) ←(DST) ∨(SRC)**
  - **NOT** 逻辑非
    - 指令格式：**NOT OPR** ；操作：**(OPR) ←$\overline{(OPR)}$**
  - **XOR** 异或
    - 指令格式：**AND DST, SRC** ；操作：**(DST) ←(DST) ⊻(SRC)**
  - **TEST** 测试指令
    - 指令格式：**TEST OPR1, OPR2** ；操作：**(OPR1) ∧(OPR2);**

      结果不保存，只根据其特征设置条件码

– 说明：
  - 上5条指令中，NOT不允许使用立即数，其它4条指令除非源操作数是立即数，至少有1个操作数必须存放在寄存器中，另1个操作数则可以使用任意寻址方式。
  - 对标志位的影响：NOT不影响标志位，其它4种指令将使CF和OF为0，AF位无定义，而SF, ZF和PF则根据运算结果设置。
– 例：(1) 要求将AL中0, 1两位屏蔽　　(2)要求将BL中第5位置1

```
AND AL, 11111100B
```

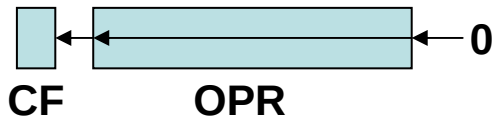
```
OR  BL, 00100000B
```

(3)要求将AX中0，1位取反

```
XOR AX, 00000011B
```

移位指令

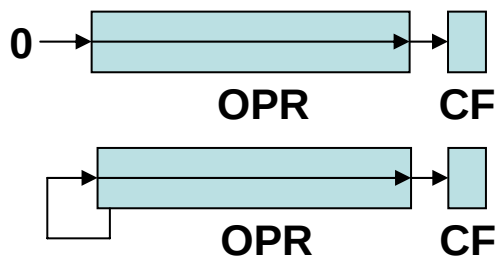
– SHL 逻辑左移指令



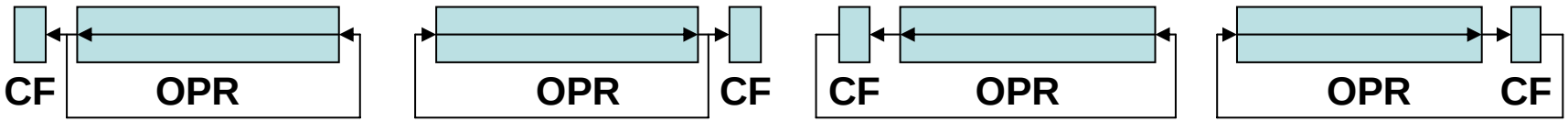
- 指令格式：**SHL OPR, CNT**
- 执行操作： **OPR**可以是除立即数以外的任何寻址方式，移位次数由**CNT**决定（**CNT**可以是**1**或**CL**）.
- 说明：对条件码的影响是，**CF**位根据指令规定设置，**OF**只有当**CNT=1**时才有效，在移位后最高有效位的值发生变化**(0**变**1**或**1**变**0)**时**OF**置**1**，否则**OF**置**0**；

– 其它移位指令

- **SAL OPR, CNT** ;算术左移指令（同**SHL**）
- **SHR OPR, CNT** ;逻辑右移指令
- **SAR OPR, CNT** ;算术右移指令



- **ROL OPR, CNT** ;循环左移指令
- **ROR OPR, CNT** ;循环右移指令
- **RCL OPR, CNT** ;带进位循环左移指令
- **RCR OPR, CNT** ;带进位循环右移指令



– 例：

(1) MOV CL, 5 ; 执行前**(DS)=0F800H,(DI)=180AH,(0F980A)=0064H**

SAR [DI], CL ; 则执行后**(0F980A)=0003H, CF=0**

(2) MOV CL, 2 ;执行前**(SI)=1450H**

SHL SI, CL ;执行后**(SI)=5140H, CF=0**

(3) 如**(AX)=0012H, (BX)=0034H,**要求将它们装配在一起形成**(AX)=1234H.**

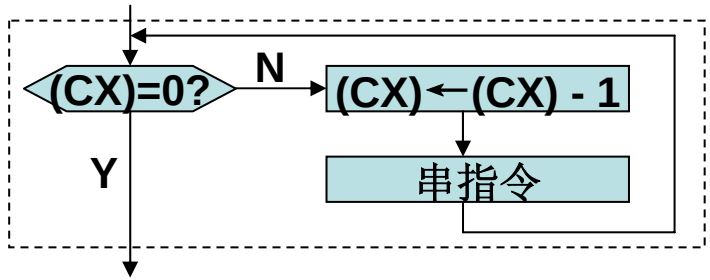
```
MOV CL, 8
ROL AX, CL
ADD AX, BX
```

2.2.4 串处理指令

- 串指令：
  - **MOVS**　串传送
  - **CMPS**　串比较
  - **SCAS**　串扫描
  - **LODS**　从串取
  - **STOS**　存入串
- 与上述基本指令配合使用的前缀指令：
  - **REP**　重复
  - **REPE/REPZ**　相等/为0则重复
  - **REPNE/REPNZ**　不相等/不为0则重复

– **REP** 重复串指令

MOVS, STOS, LODS



- 指令格式：**REP** 串指令
- 执行操作：

– **MOVS** 串传送指令

- 指令格式：**MOVS DST, SRC** ;在操作数中要表明字/字节操作

  MOVSB （字节）

  MOVSW （字）

- 执行操作: **(1) (DI) ←(SI)**

  DF=0时用＋,DF=1时用－

  (2) 字节操作: **(SI) ← (SI) ±1, (DI) ← (DI) ± 1**

  (3) 字操作: **(SI) ←(SI) ±2, (DI) ← (DI) ±2**

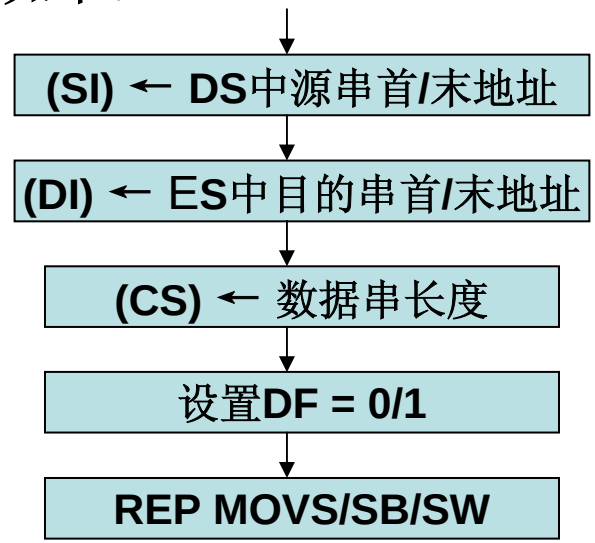
- 说明：

  A.该指令不影响条件码。

  B.该指令可以把由**(SI)**指向的数据段中的**1**个字/字节→由**(DI)**指向的附加段中的**1**个字/字节中，同时根据**DF**值及数据格式(字/字节)对**(SI)**和**(DI)**进行修改。

C. 该指令与REP联用，则可将数据段中整串数据→附加段中，但执行前要做准备工作，步骤如下：

- 例：将DS段中17个字符送到DS段中。




```
;----------------定义数据段----------------------
DATAREA SEGMENT
        MESS1   DB   'personal computer $'
DATAREA ENDS
;----------------定义附加段----------------------
EXTRA SEGMENT
        MESS2   DB   17   DUP(?)
EXTRA ENDS
;--------------------定义代码段--------------------------------
CODE SEGMENT
        ASSUME CS:CODE, DS:DATAREA, ES:EXTRA
        …
        MOV AX, DATAREA
        MOV DS, AX
```

```
        MOV AX, EXTRA
        MOV ES, AX
        …
        LEA SI, MESS1
        LEA DI, MESS2
        MOV CX,17
        CLD
        REP MOVSB
        …
CODE ENDS
        END
```

– **STOS 存入串指令**

- 指令格式：**STOS DST**

  STOSB（字节）

  STOSW（字）

- 执行操作：字节操作：**((DI)) ← (AL), (DI) ← (DI) ± 1**

  字操作：　**((DI)) ← (AX), (DI) ← (DI) ± 2**

- 说明：

  A.该指令不影响条件码。

  B.该指令把由**(AL)/(AX)**→由**(DI)**指定的附加段某单元中，并根据**DF**值及数据格式**(**字**/**字节**)**修改**(DI)**。

  C. 该指令与**REP**联用，则可将**(AL)/(AX)**存入一个串长度为**(CX)**的缓冲区中。

  D.该指令在初始化某一缓冲区时很有用。

– LODS 取出串指令

- 指令格式：**LODS DST**

  LODSB （字节）

  LODSW（字）

- 执行操作：字节操作：**(AL) ← ((SI)), (SI) ← (SI) ± 1**

  字操作：　**(AX) ← ((SI)), (SI) ← (SI) ± 2**

- 说明：

  A.该指令不影响条件码。

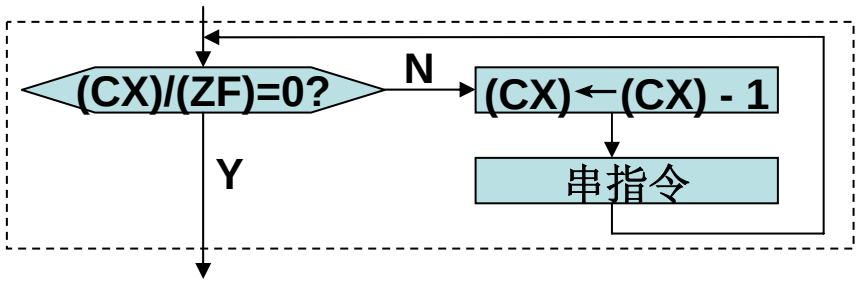
  B.该指令把由 **(SI)**指定的数据段某单元内容中→ **(AL)/(AX)**，并根据**DF**值及数据格式**(**字**/**字节**)**修改**(SI)**。

  C. 一般说来，该指令不和**REP**联用。有时缓冲区中的一串字符需要逐次取出来测试时，可用本指令。

– **REPE/REPZ** 当相等/为0时重复串操作

CMPS, SCAS

- 指令格式：**REPE/REPZ** 串指令
- 执行操作：



– **REPNE/REPNZ** 当不相等/不为0时重复串操作

- 指令格式：**REPE/REPZ** 串指令
- 执行操作：退出条件为 **(CX)=0** 或 **ZF=1**，其它与**REPE**相同。

– **CMPS** 串比较指令

- 指令格式：　**CMPS SRC, DST**

  CMPSB（字节）;

  CMPSW（字）

- 执行操作：**((SI))－((DI))**

  字节操作：**(SI) ← (SI) ± 1, (DI) ← (DI) ± 1**

  字操作：　**(SI) ← (SI) ± 2 , (DI) ← (DI) ± 2**

- 说明：**(SI)**指向的**DS**段中的**1**个字**/**字节－**(DI)**指向的**ES**段中的**1**个字**/**字节，不保存结果，只根据结果设置条件码。

– **SCAS 串扫描指令**

- 指令格式：　**SCAS SRC, DST**

  SCASB （字节）;

  SCASW（字）

- 执行操作：字节操作：**(AL)－(DI), (DI) ← (DI) ± 1**

  字操作：　**(AL)－(DI), (DI) ← (DI) ± 2**

- 说明：把**(AL)**或**(AX)**与由**(DI)**指定的**ES**段中**1**个字节**/**字比较，不保存结果，只根据结果设置条件码。

- 以上**2**条指令与**REPE/REPZ**或**REPNE/REPNZ**结合可比较**2**个数据串或在**1**个串中查找**1**个指定字符。

- 串处理指令，几个需要注意的问题：
  - 用串处理指令在不同段之间传送或比较数据，如果需要在同一段内处理数据，可以在**DS**和**ES**中设置同样的地址，或者在源操作数字段使用段跨越前缀来实现。例如：**MOVS　[DI], ES:[SI]**
  - 注意循环次数的设置。对于字指令来说，**(CX)**中预置的值应该是字的个数而不是字节的个数。
  - 注意方向标志的设置。正向时设置**DF=0**，反向时设置**DF=1**。

2.2.5 控制转移指令

① 无条件转移指令
- **JMP**

② 条件转移指令
- 根据单个标志情况转移（**JZ/JE, JNZ/JNE; JS,JNS; JO,JNO; JP,JNP; JB/JC,JNB/JNC**）
- 比较两无符号数，由结果转移（**JB/JC, JNB/JNC, JBE/JNA, JNBE/JA**）
- 比较两带符号数，由结果转移（ **JL/JNGE, JNL, JLE/JNG, JNLE/JG** ）
- 测试**CX**的值，为**0**则转移（**JCXZ**）

③ 循环指令
- **LOOP, LOOPZ/LOOPE, LOOPNZ/LOOPNE**

④ 子程序调用和返回
- **CALL, RET**

⑤ 中断调用和返回
- **INT, INTO, IRET**

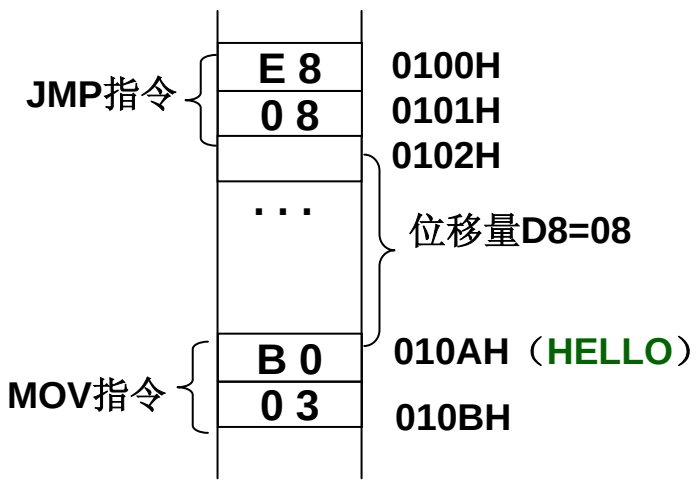
① 无条件转移指令: JMP指令

- 作用：无条件转移到指定的地址去执行从该地址开始的指令。
- 两类：段内转移。在同一个段范围内进行转移（修改**IP**）。
  段间转移。转到另一个段去执行程序（修改**CS**和**IP**）。
- 格式：（不影响条件码）

A. 段内直接短转移

- 指令格式：**JMP SHORT OPR**
- 执行操作：**(IP) ← (IP)＋8b偏移量　；-128~127字节范围内**
- 例：

```
        …
        JMP SHORT HELLO
        …
HELLO:  MOV AL, 3
        …
```



B. 段内直接近转移

- 指令格式：**JMP NEAR PTR OPR**
- 执行操作：**(IP) ← (IP)＋16b**偏移量

  位移量是**16b**，可以转移到段内任一个位置。

C. 段内间接转移

- 指令格式：**JMP WORD PTR OPR**
- 执行操作：**(IP) ← (EA)**

  有效地址**EA**由**OPR**的寻址方式确定，可以是除立即数以外的任一种寻址方式。

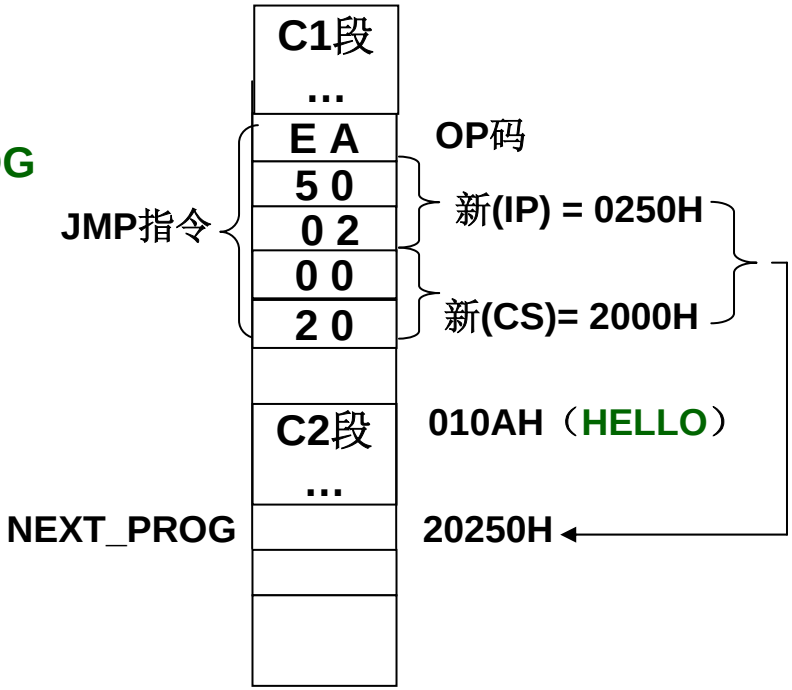
D. 段间直接（远）转移

- 指令格式：**JMP FAR PTR OPR**
- 执行操作：**(IP) ← OPR**的段内偏移地址

  (CS) ← OPR所在段的段地址

- 例：

```
C1 SEGMENT
        …
        JMP FAR PTR NEXT_PROG
        …
C1 ENDS
C2 SEGMENT
        …
        NEXT_PROG:
        …
C2 ENDS
```



E. 段间间接转移

- 指令格式：**JMP DWORD PTR OPR**
- 执行操作：**(IP) ← (EA)**

  (CS) ← (EA＋2)

  EA由寻址方式确定，可使用除立即数和寄存器方式外的任何存储器寻址方式
- 例： **JMP DWORD PTR ALPHA[SP][DI]**

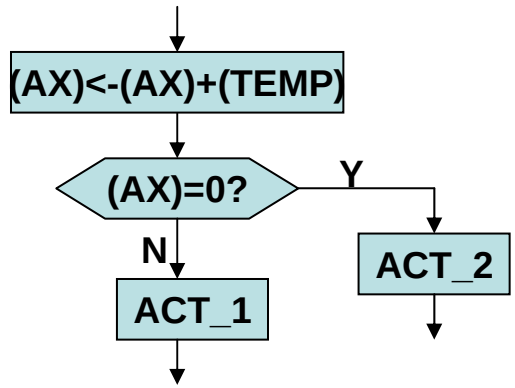
② 条件转移指令

– 根据单个标志的情况设置转移

操作符	指令格式	执行操作	测试条件
JZ / JE	**JZ OPR**	结果为0则转移	**ZF = 1**
JNZ/JNE	**JNZ OPR**	结果非0则转移	**ZF = 0**
JS	**JS OPR**	结果为负则转移	**SF = 1**
JNS	**JNS OPR**	结果为正则转移	**SF = 0**
JO	**JO OPR**	结果超过范围则转移	**OF = 1**
JNO	**JNO OPR**	结果不溢出则转移	**OF = 0**
JP/JPE	**JP OPR**	结果低8位中1个数为偶数则转移	**PF = 1**
JNP/JPO	**JNP OPR**	结果低8位中1个数为奇数则转移	**PF = 0**
JB/JNAC/JC	**JC OPR**	结果超出无符号数表示范围转移	**CF = 1**
JNB/JAE/JNC	**JNC OPR**	结果在无符号数表示范围内转移	**CF = 0**

- 例：
  - 如果需要根据一次加法运算的结果实行不同的处理，程序框图如右图所示，程序如下：

```
        …                          或      …
        ADD AX, TEMP                       ADD AX, TEMP
        JZ ACT_2                           JNZ ACT_1
        …          } ACT_1                 …          } ACT_2
ACT_2:                             ACT_1:
        …          } ACT_2                 …          } ACT_1
```



  - 比较2个数是否相等，如相等做动作1，否则做动作2。

```
        …                          或      …
        CMP AX, BX                         CMP AX, BX
        JE ACT_1                           JNE ACT_2
        …          } ACT_2                 …          } ACT_1
ACT_1:                             ACT_2:
        …          } ACT_1                 …          } ACT_2
```

– 比较两无符号数，根据比较结果转移

操作符	指令格式	执行操作	测试条件
JB/JNAE/JC	JB OPR	低于/不高于或等于/CF=1，则转移	CF = 1
JNB/JAE/JNC	JNB OPR	不低于/高于或等于/CF=0，则转移	CF = 0
JBE	JBE OPR	低于或等于/不高于，则转移	CF∨ZF = 1
JNBE	JNBE OPR	不低于或等于/高于，则转移	CF∨ZF = 0

– 比较两带符号数，根据比较结果转移

操作符	指令格式	执行操作	测试条件
JL/JNGE	JL OPR	小于/不大于或等于，则转移	SF∀OF = 1
JNL/JGE	JNL OPR	不小于/大于或等于，则转移	SF∀OF = 0
JLE/JNG	JLE OPR	小于或等于/不大于，则转移	(SF∀OF) ∨ZF = 1
JNLE	JNLE OPR	不小于或等于/大于，则转移	(SF∀OF) ∨ZF = 0

- 例：MOV AX, A

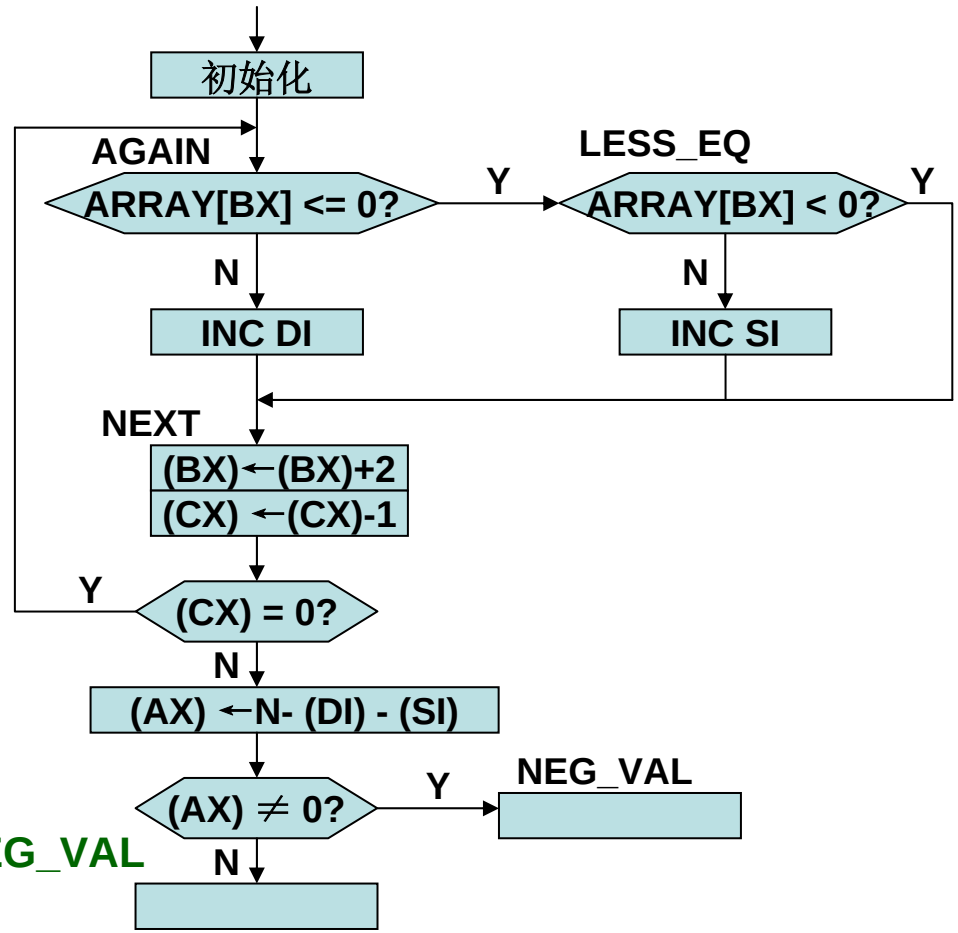
  CMP AX, B

  JL X ;如果(A) < (B)则转移到X去执行

- 测试**CX**的值为**0**则转移指令：**JCXZ**
  - 指令格式：**JCXZ OPR**
  - 测试条件：**(CX)＝0?**
  - **(CX)**常用来设置计数值，所以这条指令可根据**(CX)**的修改情况来产生两个不同的分支。
- 例：
  - 在存储器中有首地址为**ARRAY**的**N**字数组，要求测试其中正数、**0**及负数个数: **(DI)**←正数个数, **(SI)**←**0**个数, **(AX)**←**N- (DI) - (SI)**，如果有负数则转移到**NEG_VAL**中执行。

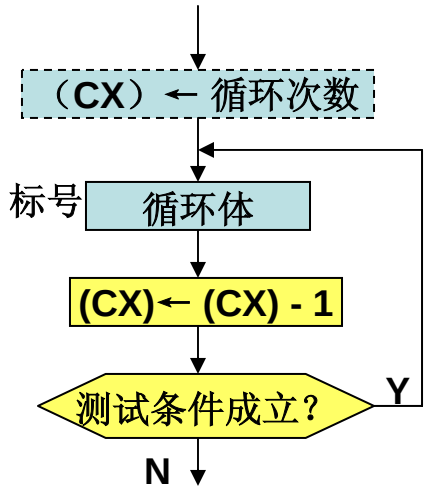
```
MOV CX, N          ; 初始化
MOV BX, 0
MOV DI, BX
MOV SI, BX
```

```
AGAIN:   CMP ARRAY[BX],0
         JLE  LESS_EQ
         INC DI
         JMP SHORT NEXT
LESS_EQ: JL NEXT
         INC SI
NEXT:    ADD BX, 2
         DEC CX
         JNZ AGAIN
         MOV AX, N
         SUB AX, DI
         SUB AX, SI
         JZ SKIP
         JMP NEAR PTR NEG_VAL
SKIP: …
NEG_VAL:…
```



③ 循环指令

操作符	指令格式	测试条件
LOOP	LOOP OPR	(CX) ≠ 0
LOOPZ / LOOPE	LOOPZ OPR	ZF=1 且 (CX) ≠ 0
LOONZ / LOONE	LOONE OPR	ZF=0 且 (CX) ≠ 0



– 执行步骤：右图

```
        MOV   CX, N
        …
AGAIN:
        …
        LOOP AGAIN
```

– 例：有一个首地址为**ARRAY**的**M**字数组，试编写程序：求该数组的内容之和（不考虑溢出），并把结果存入**TOTAL**中。

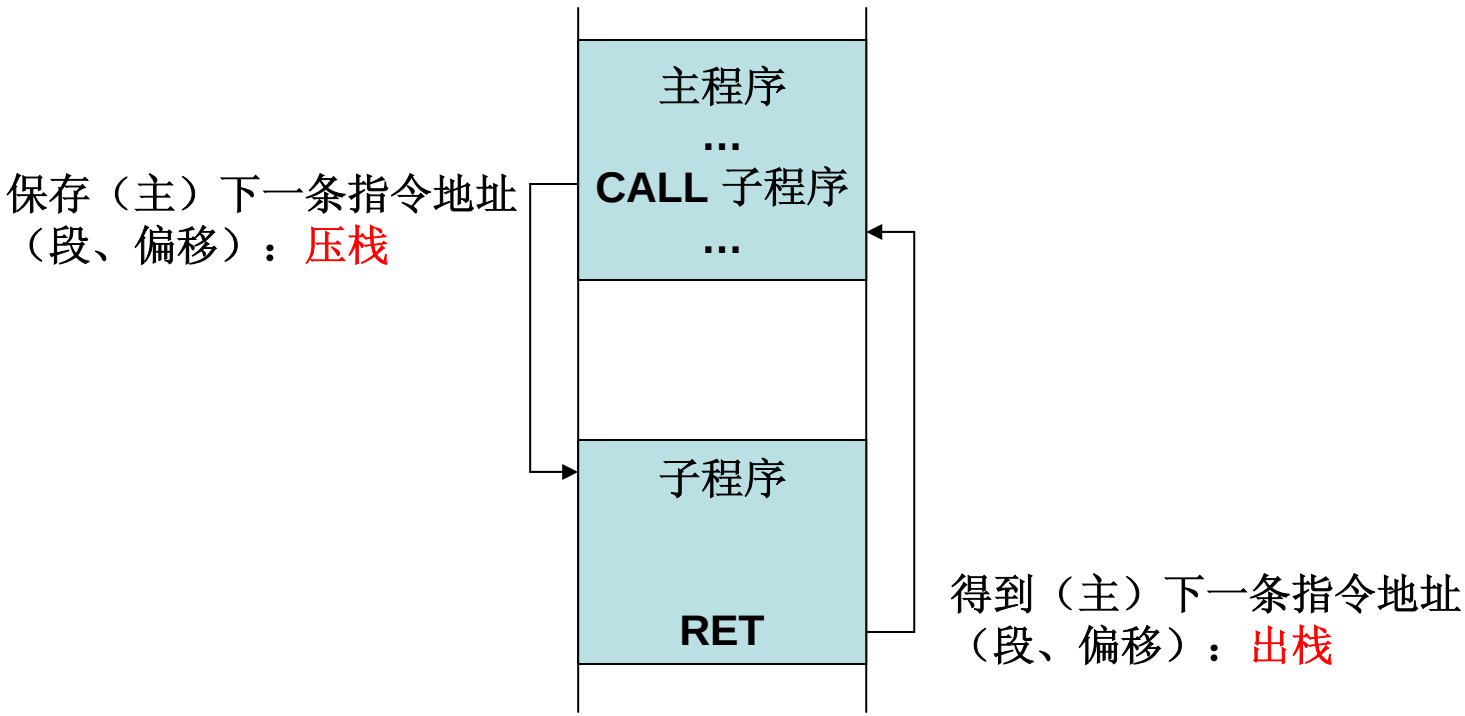
```
        MOV   CX, M     ；循环次数
        MOV   AX, 0
        MOV   SI, AX
AGAIN:  ADD   AX, ARRAY[SI]
        ADD   SI, 2
        LOOP AGAIN
        MOV   TOTAL , AX
```

④ 子程序调用和返回指令

- 子程序：相当于高级语言中的过程。为便于模块化设计，将程序中某些具有独立功能部分的程序模块，称之为子程序。

– **CALL** 调用子程序指令

– **RET** 子程序返回指令

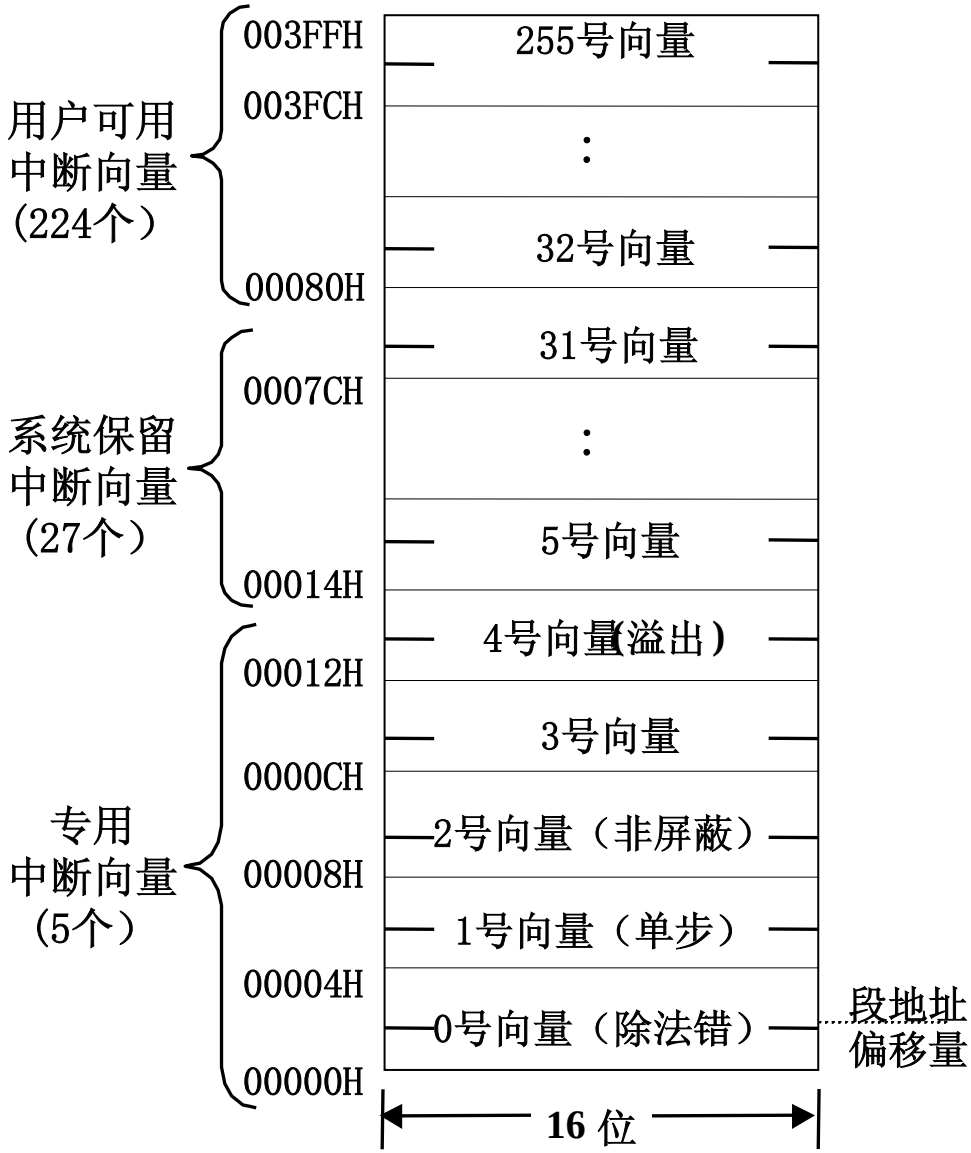


CALL	段内		段间	
	直接调用	间接调用	直接调用	间接调用
格式	CALL DST	CALL DST	CALL DST	CALL DST
执行操作	(SP) ←(SP)-2 ((SP)+1,(SP))←(IP) (IP) ← (IP)+D16	同直接调用(将主IP压栈) (IP) ← (EA)	(SP) ←(SP)-2 ((SP)+1,(SP))←(CS) (SP) ←(SP)-2 ((SP)+1,(SP))←(IP) (IP) ← (子)偏移地址 (CS) ← (子)段地址	同直接调用(将主IP和CS压栈) (IP) ← (子)(EA) (CS) ← (子)(EA+2)
RET	返回	带立即数返回	返回	带立即数返回
格式	RET	RET EXP	RET	RET EXP
执行操作	(IP)←((SP)+1,(SP)) (SP) ←(SP)+ 2	同左返回(将主IP出栈) (SP)←(SP)+D16	(IP)←((SP)+1,(SP)) (SP) ←(SP)+ 2 (CS)←((SP)+1,(SP)) (SP) ←(SP)+2	同左返回(将主IP和CS出栈) (SP) ←(SP)+D16

- 中断
  - 概念：
    - 当系统运行或者程序运行期间，在遇到某些紧急事件时，需要计算机暂停正在执行的程序，自动转去执行处理紧急事件的子程序，当紧急事件处理完毕后，恢复原来的状态，再继续执行原来的程序。这种对紧急事件的处理模式，称为中断。
    - 中断源和中断类型号：引起中断的事件称为中断源。中断源一般有多个（IBM-PC可有256个），为区分各个中断源，给每个中断源一个二进制编号，该编号称为该中断源的中断类型号。
    - 中断处理程序：处理紧急事件的子程序。
    - 中断向量：中断处理程序在内存中的入口地址。每个中断向量占4个字节（前2个字节存放偏移量，后2个字节存放段地址），一般存放在存储器的低地址区。中断向量入口地址＝4×中断类型号

– **8086中断向量表**

- 位于内存**0**段的**0～3FFH**，最多可以容纳**256**个中断向量。
- **0~4**：为专用中断指针，用户不能对其修改。
- **5~31**：为保留中断指针，这是**Intel**公司保留的中断指针，用户不应使用。
- **32~255**：为用户使用的中断指针，它可由用户指定。



– 中断指令（INT, INTO, IRET）

中断指令	INT	INTO （如溢出则中断）	IRET （从中断返回）
指令格式	INT TYPE 或 INT (对INT, 隐含的TYPE=3)	INTO	IRET
执行操作	(SP) ← (SP)-2 ((SP)+1,(SP)) ← (PSW) (SP) ←(SP)-2 ((SP)+1,(SP)) ← (CS) (SP) ← (SP)-2 ((SP)+1,(SP)) ← (IP) (IP) ← (TYPE*4) (CS) ← (TYPE*4+2)	若IF=1，则: (SP) ← (SP)-2 ((SP)+1,(SP)) ← (PSW) (SP) ←(SP)-2 ((SP)+1,(SP)) ← (CS) (SP) ← (SP)-2 ((SP)+1,(SP)) ← (IP) (IP) ← (10H) (CS) ← (12H)	(IP) ← ((SP)+1,(SP)) (SP) ← (SP)+2 (CS) ← ((SP)+1,(SP)) (SP) ← (SP)+2 (PSW) ← ((SP)+1,(SP)) (SP) ← (SP)+2

– **DOS中断**

- **BIOS中断**
- **DOS功能调用的使用方法：**

① 在**(AH)**中存入所要调用功能的功能号；

② 根据所调用功能的规定设置入口参数；

③ 用**INT 21H**指令转入子程序入口；

④ 相应的子程序运行完后，可以按规定取得出口参数。

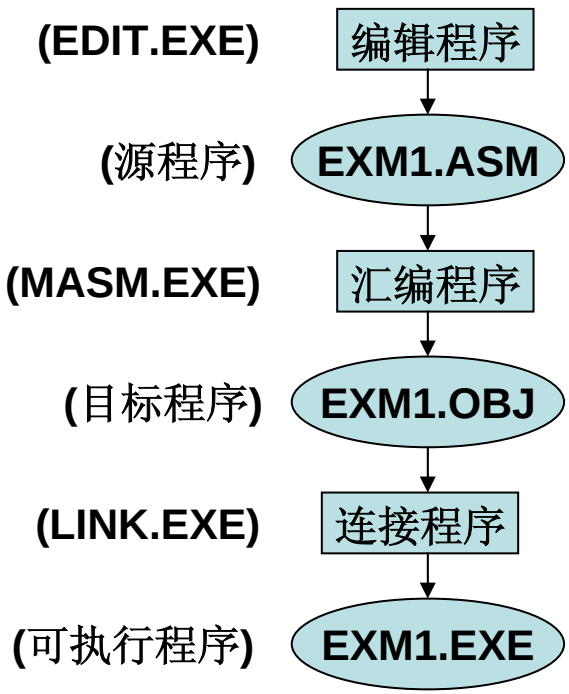
AH	功能	入口参数	出口参数
00H	程序终止	CS = 程序段前缀	
01H	键盘输入并回显		AL=输入字符
02H	显示输出	DL = 输出字符	
09H	显示字符串	DS:DX=串地址 ‘$’结束字符串	
25H	设置中断向量	DS:DX=中断向量，AL=中断类型号	
35H	取中断向量	AL=中断类型	ES:BX=中断向量

2.2.6 处理机控制指令

- **NOP**指令
  - 该指令不执行任何操作，在机器码中占**1**个字节单元。
- **HLT** 停机指令
  - 该指令可使机器暂停工作，使**CPU**处于停机状态以便等待**1**次外部中断到来，中断结束后可继续执行下面的程序。
- **WAIT**等待指令
  - 该指令可使**CPU**处于空转状态，它也可用来等待外部中断发生，但中断结束后仍返回**WAIT**指令继续等待。
- **ESC** 换码指令（与协处理器有关）
  - 格式：**ESC mem**
- **LOCK**封锁指令（与协处理器有关）

第3章 汇编语言程序设计

汇编程序上机过程



执行**EDIT.EXE**，在其中编写源程序，编写完成后保存（如：**EXM1.ASM**）

C:>MASM EXM1

（如果汇编时显示错误，根据提示的错误行，修改错误再汇编）

C:>LINK EXM1

（如果出错，一般是源程序出错，仔细检查源程序，修改错误后重新汇编、连接）

- 汇编语言源程序结构
  - 例：将**40**个字母**a**的字符串从源缓冲区传送到目的缓冲区。

```
data segment            ；数据段
        sour_buf   db   40 dup ('a')
data ends

extra segment           ；附加段
        dest_buff  db   40 dup(?)
extra ends

code segment            ；代码段
main proc far                       ;主程序部分
        assume cs:code, ds:data
   start: push ds                   ;保存旧数据段
        sub ax, ax
        push ax
        mov ax, data                ;装入数据段地址
        mov ds, ax
        mov ax, extra               ;装入附加段地址
        mov es, ax
        lea si, sour_buff           ;源串偏移地址送SI
        lea di, dest_buff           ;目的串偏移地址送DI
        cld                         ;清DF=0
        mov cx, 40                  ;传送次数送CX
        rep movsb                   ;串传送
        ret                         ;返回到DOS
main endp                           ;主程序结束
code ends               ；代码段结束
        end start       ；源程序结束
```

- 由上例，汇编语言源程序由若干个段构成。
  - 数据段，附加段，堆栈段：进行变量、常量定义。根据具体程序可以没有。
  - 代码段：任何源程序都必须有代码段。为了使程序更结构化，代码段一般由若干个过程构成，过程之间通过调用来协作。
  - 各个段程序由若干语句构成。汇编语言语句分为两类：
    - 指令性语句。产生目标代码。格式：

      【标号:】操作码 【操作数1】【,操作数2】【;注释】
    - 指示性语句。不产生目标代码，仅提供汇编信息。格式：

      【变量名】伪指令 【操作数1】【,操作数2,…】【;注释】

伪操作

- 数据定义和储存分配伪操作
  - 数据类型：
    - **DB**：用来定义字节，其后的每个操作数都占用**1**个字节。
    - **DW**：用来定义字，其后的每个操作数都占用**1**个字。
    - **DD**：用来定义双字，其后的每个操作数都占用**2**个字。
    - **DQ**：用来定义**4**个字，其后的每个操作数都占用**4**个字。
    - **DT**：用来定义**10**个字节，其后的每个操作数都占用**10**个字节，形成压缩的**BCD**码。
  - 例：

```
DATA_BYTE     DB    10, 4, 10H
MESSAGE       DB    'HELLO'          ；操作数可以是字符串
ABC           DB    0, ?, ?, 0       ；?表示分配存储空间，不存入值
DATA_WORD     DW    100, 10H, -5
DATA_DW       DD    3*20, 0FFFDH
```

– **DUP**

- 数据定义时，操作数可用**DUP**操作符来复制某个操作数，格式：

  count DUP (operand, …, operand)

  含义：将括号中的操作数重复**count**次，**count**可以是一个表达式，其值应该是一个正数。

- 例：
```
ARRAY1   DB   2 DUP(0, 1, 2, ?)
ARRAY2   DB  100 DUP(?)
ARRAY3   DB  100 DUP(0, 2 DUP(1,2), 0, 3) ;DUP操作可嵌套
```

– **PTR**

- 汇编语言中，源和目的操作数类型应该相同，如果数据段中数据类型与源或目的操作数类型不同，则应该进行类型转换，格式：

  类型 **PTR** 变量 ± 常数表达式

  说明：类型可以是**BYTE, WORD, DWORD**

- 例：
```
OPER1   DB       1, 2
OPER2   DW       1234H, 5678H
        …
MOV  AX, WORD PTR OPER1+1        ; (AX) = 3402H
MOV  AL, BYTE PTR OPER2          ;(AL) = 34H
```

- **LABEL**
  - 汇编语言中，同一个变量可以具有不同类型，可用**LABEL**指令指定其类型。格式：

    变量名 **LABEL** 类型　　；类型:**BYTE,WORD,DWORD**
  - 例：　**DA1　LABEL　BYTE**

    DA2　DW　50　DUP(0)

    将数组首地址赋给字节类型变量**DA1**和字类型变量**DA2**，此时**DA1**可进行字节操作, **DA2**可进行字操作。

- 表达式赋值**EQU**
  - 当程序中多次出现同一表达式，为方便可以用赋值伪操作给表达式赋予一个名字，其格式：
  - 表达式名 **EQU** 表达式
  - 例：　**ALPHA　EQU　256**

    BETA　EQU　ALPHA+2

    ADDR　EQU　VAR+BETA

    P8　EQU　DS:[BP+ ADDR]

- 段定义伪操作
  - 段定义。格式：

    段名 **SEGMENT** 【定位类型】【组合类型】【'类别'】

    …

    段名 **ENDS**

    对段定义的说明：

    - 【定位类型】：指明段起始地址应有怎样的边界值。选项：
      - **PARA**：默认，段起始地址必须是**16**的倍数。
      - **BYTE**：该段可以从任何地址开始。
      - **WORD**：该段必须从偶地址开始。
      - **PAGE**：段起始地址必须是**256**的倍数。
    - 【组合类型】：说明程序连接时的合并方式。选项：
      - **PUBLIC, COMMON, AT** 表达式，**STACK**，**MEMORY**。

- 段定义时，还必须明确段与段寄存器的关系，格式：

  ASSUME 段寄存器名:段名, 段寄存器名:段名,…

  - **ASSUME**只指定某个段分配给某段寄存器，并不将段地址装入段寄存器，所以在代码段中还必须将段地址装入相应段寄存器中（对代码段无需装入），可通过**MOV**指令实现。如：

    ```
    ASSUME  CS:CODE_SEG, DS:DATA
    MOV AX, DATA
    MOV DS, AX
    ```

过程定义伪操作

- 过程定义用在过程（子程序）的前后，使整个过程形成清晰的，具有特定功能的代码块。
- 格式：

```
过程名 PORC 属性
   …
过程名 ENDP
```

– 说明：
  - 过程名：标识符，是子程序入口的符号地址。
  - 属性：可以是**NEAR**或**FAR**，属性确定原则：
    – 调用程序和过程在同一个代码段中则使用**NEAR**属性，
    – 调用过程和过程不在同一个代码段中则使用**FAR**属性
    – 当一过程被同一段和另一段调用时，使用**FAR**属性
    – 一般说来，主过程应定义为**FAR**属性。

– 例：调用程序和子程序在同一代码段中。

```
MAIN PROC FAR
   …
   CALL SUBR1
   …
   RET
MAIN ENDP
SUBR1 PROC NEAR
   …
   RET
SUBR1 ENDP
```

或

```
MAIN PROC FAR ；过程嵌套
   …
   CALL SUBR1
   …
   RET
SUBR1 PROC NEAR
   …
   RET
SUBR1 ENDP
MAIN ENDP
```

- 程序开始和结束伪操作
  - 程序开始可用**NAME**或**TITLE**为模块取名。格式：

    NAME 模块名　　；程序将以“模块名”作为模块的名字

    TITLE 说明文本　　；指定每**1**页上打印的标题，

    - 说明：
      - 若程序中无**NAME**伪操作，将以说明文本中前**6**个字符作模块名（说明文本最多可以有**60**个字符）；
      - 若程序中既无**NAME**伪操作又无**TITLE**伪操作，则以文件名作模块名。
  - 源程序结束伪指令。格式：

    END 【标号】

    - 标号指示程序开始执行的起始地址。若多个程序模块，则只需指明主程序开始时的标号。

对准伪操作

EVEN

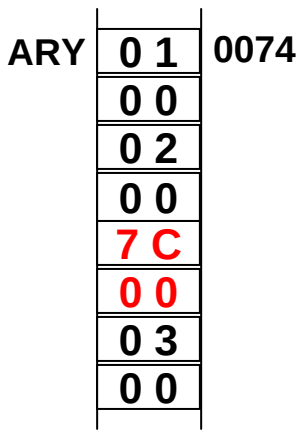
- 说明：该操作使下一个字节地址成为偶数。一个字的地址最好从偶地址开始，为保证此点，可以在其前面用**EVEN**操作来达到这一目的。
- 例：

```
D_SEG  SEGMENT
        …
        EVEN
        WARY DW 100 DUP(?)
        …
D_SEG ENDS
```

ORG

- 格式：**ORG** 常数表达式
- 说明：该操作可使下一个字节的地址为常数表达式的值（**0～65536**）。
- 例：

```
VECTOR SEGMENT
        ORG 10
        VECT1 DW 47A5H
        ORG 20
        VECT2 DW 0C96H
        …
VECTOR ENDS
；VECT1偏移地址为0AH,
；VECT2偏移地址为14H
ARY DW 1, 2, $+4, 3
```



- 基数控制伪操作

高级汇编语言技术

- 宏定义、宏调用和取消宏

  宏是源程序中一段有独立功能的程序代码。它只需在源程序中定义一次，就可以多次调用。

  – 宏定义
    - 格式：宏名 **MACRO [参数1, 参数2, …]**

      …

      ENDM

    说明：参数可以没有，可以是操作数，操作码，等等。

  – 宏调用
    - 格式： 宏名 **[参数1, 参数2, …]**
    - 说明：源程序被汇编时，将对每个宏调用作宏展开（即用宏定义体取代源程序中宏指令名，用实参取代形参）。

  – 取消宏
    - 格式：**PURGE** 宏名

– 例: **(1)** 用宏定义完成**2**个字操作数相乘，得到**1**个**16**位的第**3**个操作数，作为结果。

- 宏定义：

```
M2 MACRO OP1,OP2, RET
        PUSH DX
        PUSH AX
        MOV AX, OP1
        IMUL OP2
        MOV RESULT, AX
        POP AX
        POP DX
ENDM
```

- 宏调用：

```
…
M2 CX, VAR, XYZ[BX]
…
```

宏展开：

```
        …
+       PUSH DX
+       PUSH AX
+       MOV AX, CX
+       IMUL VAR
+       MOV XYZ[BX], AX
+       POP AX
+       POP DX
        …
```

– 例: (2)宏定义中包含操作码

- 宏定义：

```
FOO  MACRO  P1, P2, P3
        MOV AX, P1
        P2  P3
ENDM
```

- 宏调用：

```
FOO WORD_VAR, INC, AX
```

- 宏展开：

```
+       MOV AX, WORD_VAR
+       INC AX
```

重复汇编

- 当需要连续地重复完成（几乎完全）相同的一组代码，可使用重复汇编。
- 重复伪操作
  - 格式：**REPT** 表达式 ;表达式说明重复此数

    …（重复块）

    ENDM
  - 例：把字符**A**到**Z**的**ASCII**码填入数组**TABLE**。

```
CHAR = 'A'
TABLE LABEL BYTE
        REPT 26
          DB CHAR
          CHAR=CHAR+1
        ENDM
```

经汇编产生

```
+   DB   61H
+   DB   62H
    …
+   DB   7AH
```

- 不定重复伪操作
  - **IRP伪操作**
    - 格式：**IRP　参数，<自变量1，自变量2，…>**
      …（重复块）
      ENDM
    - 例:**(1)**

```
IRP X, <1, 2, 3, 4, 5, 6, 7>
    DB X
ENDM
```

```
+   DB 1
+   DB 2
    …
+   DB 7
```

(2)

```
IRP REG, <AX, BX, CX>
    PUSH REG
ENDM
```

```
+   PUSH AX
+   PUSH BX
+   PUSH CX
```

  - **IRPC伪指令**
    - 格式：**IRPC 参数，字符串（或<字符串>）**
      …（重复块）
      ENDM
    - 例：

```
IRPC K, A B C
    PUSH K&X
ENDM
```

```
+   PUSH AX
+   PUSH BX
+   PUSH CX
```

- 条件汇编
  - 格式：

```
IF × ×          ; × ×表示条件，如下表。
   …            } 自变量满足给定条件则汇编此块
[ELSE]
   …            } 自变量不满足给定条件则汇编此块
ENDIF
```

IF 表达式	表达式的值不为**0** 则满足条件
IFE 表达式	表达式的值为**0** 则满足条件
IFDEF 符号	若符号已在程序中定义或者已用**EXTRN**说明该符号是在外部定义的，则满足条件。
IFNDEF 符号	若符号已未定义或者未通过**EXTRN**说明为外部符号则满足条件。
IFB 自变量	如果自变量为空则满足条件
IFNB 自变量	如果自变量不为空则满足条件
IFIDN 字符串**1**，字符串**2**	如果字符串**1**和字符串**2**相同则满足条件
IFDIF 字符串**1**，字符串**2**	如果字符串**1**和字符串**2**不相同则满足条件

– 例：宏指令**MAX**把三个变元中的最大值放在**AX**中，而且使变元数不同时产生不同的程序段。

- 宏定义：

```
MAX     MACRO K, A, B, C
        LOCAL NEXT, OUT
        MOV  AX, A
        IF K-1
            IF K-2
            CMP  C, AX
            JLE   NEXT
            MOV  AX, C
            ENDIF
 NEXT:  CMP  B, AX
        JLE  OUT
        MOV  AX, B
        ENDIF
OUT:    ENDM
```

- 宏调用：

```
MAX  1, P
MAX  2, P, Q
MAX  3, P, Q, R
```

宏展开：

```
+           MOV AX, P
+??0001:
-------------------------------
+           MOV  AX, P
+??0002:    CMP  Q, AX
+           JLE  ??0003
+           MOV  AX, Q
+??0003:
-------------------------------
+           MOV  AX, P
+           CMP  R, AX
+           JLE   ??0004
+           MOV  AX, R
+??0004:    CMP  Q, AX
+           JLE   ??0005
+           MOV  AX, Q
+??0005:
-------------------------------
```

汇编语言程序格式

- 汇编语言语句格式：

- 名字：
  - 构成：名字可以是标号或变量。由字母、数字和专用字符（**?,.,@,_,$**）构成。
  - 标号：有三种属性（段地址、偏移地址和类型）
  - 变量：有三种属性（段、偏移和类型）
- 操作：
  - 操作可以是指令、伪指令或宏指令的助记符。
- 操作数

- 算术操作符
  - 有：＋，－，*，/，MOD(取余数，如19 MOD 7为5)
  - 例：把首地址为BLOCK的字数组的第6个字传送到DX.

    MOV DX, BLOCK+(6-1)*2
- 逻辑操作符（ AND, OR, XOR, NOT ）
  - 例：（1）IN AL, PORT　　（2）AND DX, PORT AND 0FEH

    OUT PORT AND 0FEH, AL
- 关系操作符
  - 有：EQ(相等)，NE(不等)，　　LT(小于)，

    GT(大于)，LE(小于等于)，GE(大于等于)
  - 说明：关系操作符的两个操作数必须都是数字或是同一段内的两个存储器地址，结果逻辑值为真表示0FFFFH，为假表示0。
  - 例：MOV BX, ((PORT LT 5) AND 20) OR ((PORT GE 5)AND 30）

    当PORT<5时汇编结果为:MOV BX, 20；否则汇编结果为: MOV BX, 30

- 数值回送操作符
  - **TYPE** （回送变量/标号的类型值）
    - 格式：**TYPE** 变量/标号
    - 若为变量，值：**DB**为**1**，**DW**为**2**，**DD**为**4**，**DQ**为**8**，**DT**为**10**。
    - 若为标号，值：**NEAR**为－**1**，**FAR**为－**2**。
    - 例：**ARRAY DW 1, 2, 3**

      则对于指令**ADD SI, TYPE ARRAY**,汇编后为**ADD SI, 2**
  - **LENGTH**
    - 格式：**LENGTH** 变量
    - 若变量中使用**DUP**,汇编程序将回送分配给该变量的单元数，对于其它情况则回送**1**。
    - 例：**FEES DW 100 DUP(0)**

      则对于指令**MOV CX, LENGTH FEES**,汇编后为**MOV CX, 100**
  - **SIZE** （回送分配给该变量字节数）
    - 格式：**SIZE** 变量　　；其值＝**LENGTH**值×**TYPE**值
    - 例：**MOV CX, SIZE FEES**　　;汇编后形成**MOV CX, 200**

- **OFFSET** （回送变量/标号偏移地址）
  - 格式：**OFFSET** 变量/标号
  - 例：**MOV BX, OFFSET OPER_ONE**
- **SEG** （回送变量/标号段地址）
  - 格式：**SEG** 变量/标号
  - 例：**MOV BX, SEG OPER1**

属性操作符

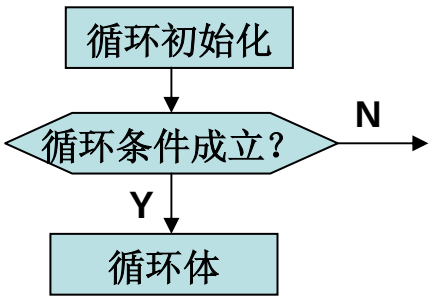
- **PTR**
  - 格式：类型 **PTR** 表达式
  - 例：
```
MOV BYTE PTR[BX], 5      ;送05H到[BX]
MOV WORD PTR[BX], 5      ;送0005H到[BX]
```
- **SHORT**
  - 用来修饰**JMP**指令中转向地址的属性，指出转向地址是在下一条指令地址的**127**个字节范围之内。
  - 例：
```
        JMP SHORT TAG
        …
TAG:    …
```

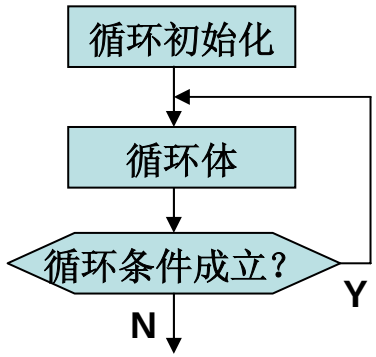
- **THIS**
  - 格式：**THIS** 属性/类型
- **HIGH和LOW** （字节分离操作符）
  - 说明：它接收**1**个数或地址表达式，**HIGH**取其高位字节，**LOW**取其低位字节。
  - 例： **CONST EQU 0ABCDH**
  - 则 **MOV AH, HIGH CONST** 将汇编成**MOV AH, 0ABH**

汇编语言程序设计方法

- 汇编语言程序设计步骤：
  - 分析所要解决的问题，确定适当的算法。
  - 根据算法设计整个程序的逻辑结构（子程序，顺序结构，选择结构和循环结构），画出程序框图。
  - 根据框图编写程序，正确运用指令、伪指令以及**DOS, BIOS**功能调用。同时写出简洁的说明和注释。
  - 上机调试程序

- 循环程序设计
  - 结构形式：

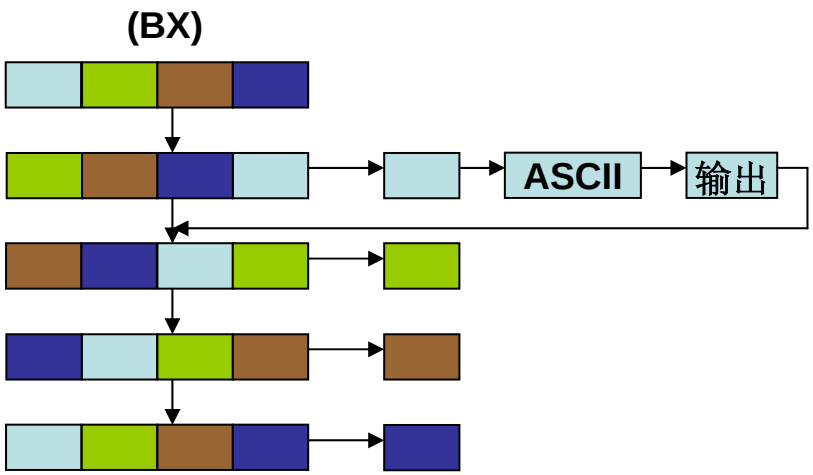


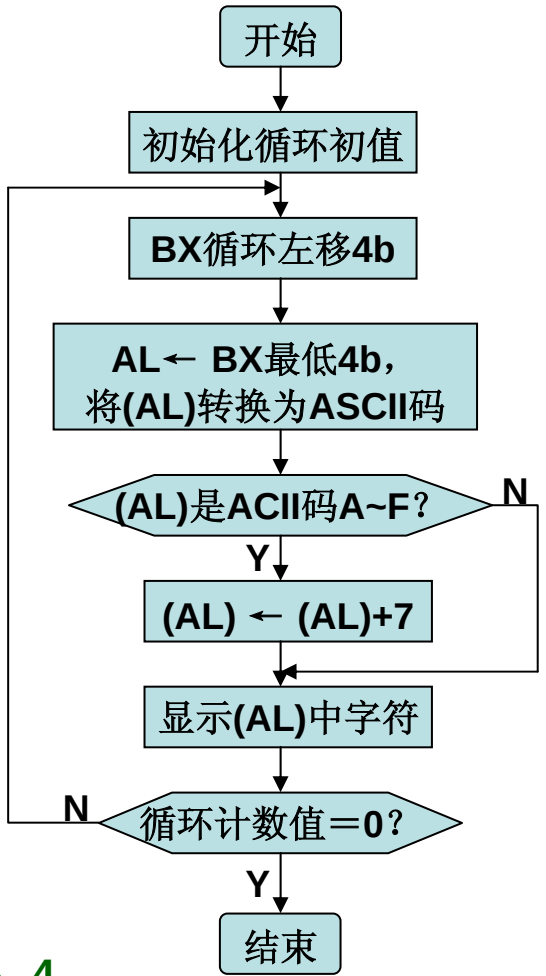


  - 循环控制指令：
    - 串/数组重复指令（**REP,REPZ/REPE**）
    - 条件转移指令（根据**PSW**中标志位状态来转移）
    - 重复控制指令**(JCXZ, LOOP, LOOPZ/LOOPE,LOONZ/LOONE)**
    - 当循环次数已知时，用 **LOOP**指令容易实现；当循环次数已知但有可能使用其它条件/特征来使循环提前结束时，用**LOOPZ/ LOOPNZ**指令容易实现。

– 例：试编写程序将**BX**寄存器内的二进制数用十六进制数的形式在屏幕上显示出来。

- 分析：





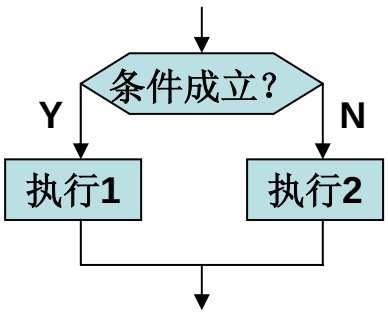

```
PROGNAM SEGMENT
MAIN PROC FAR
        ASSUME  CS:PROGNAM
START:PUSH DS
        SUB AX,AX
        PUSH AX
        MOV CH, 4          ；设置循环次数：4
```

```
ROTATE:MOV CL, 4          ；(AL) ←(BL)中低4b
       ROL BX, CL
       MOV AL, BL
       AND AL, 0FH
       ADD AL, 30H        ；将(AL)转换成ASCII码
       CMP AL, 3AH
       JL PRINTIT         ；(AL)中不为A~F，则转向输出
       ADD AL, 7H
PRINTIT: MOV DL, AL       ；输出(AL)中字符
       MOV AH, 2
       INT 21H
       DEC CH             ；循环次数－1
       JNZ ROTATE         ；循环次数<>0，则继续循环
       RET
MAIN ENDP
PROGNAM ENDS
       END
```

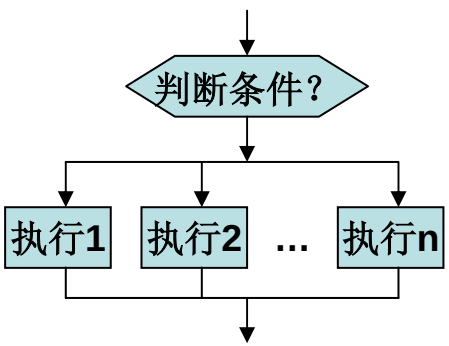
分支程序设计

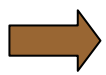
– 结构形式：

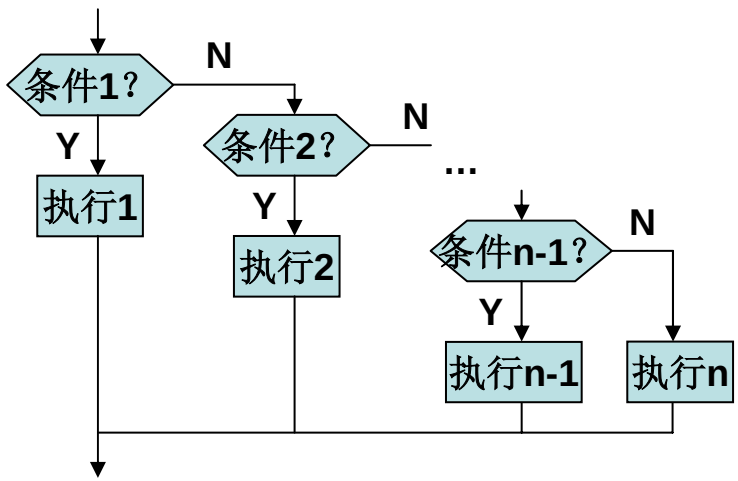
单分支结构



多分支结构

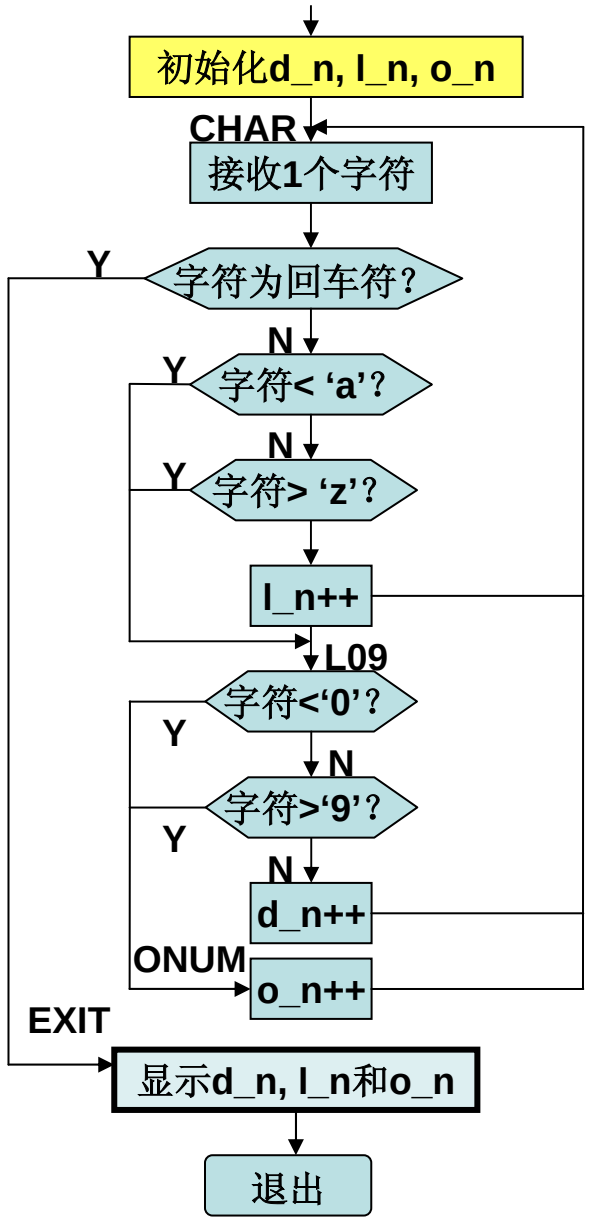




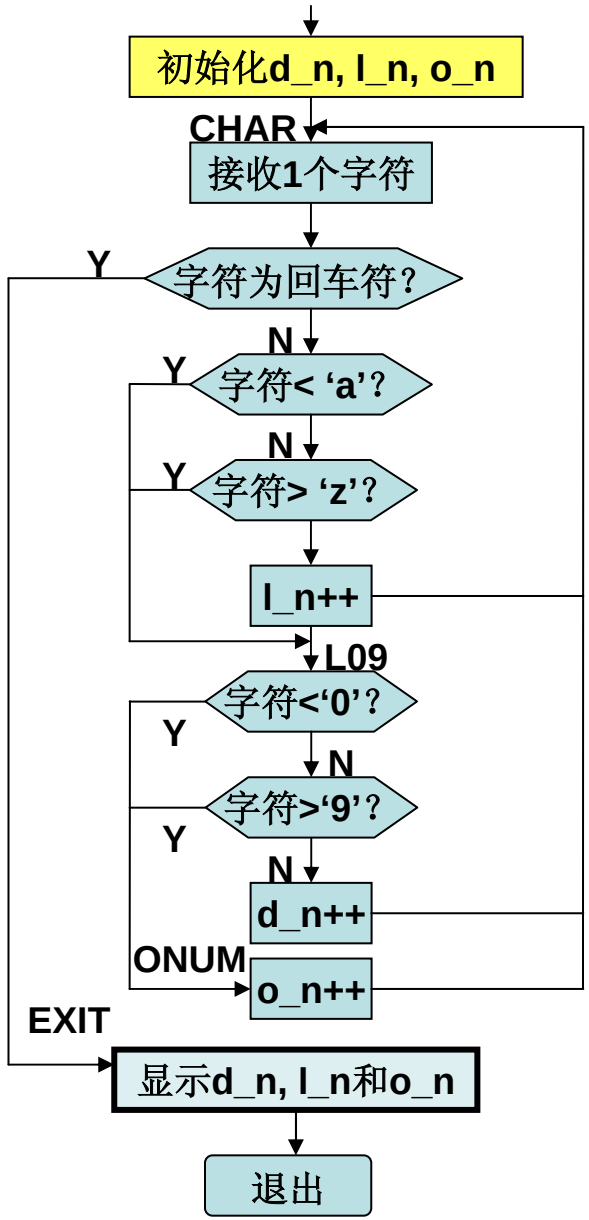


– 例：从键盘输入一系列字符（以回车符结束），并按小写字母、数字字符及其它字符分类计数，最后显示出这三类的计数结果。（假定每类字符数不超过99个）

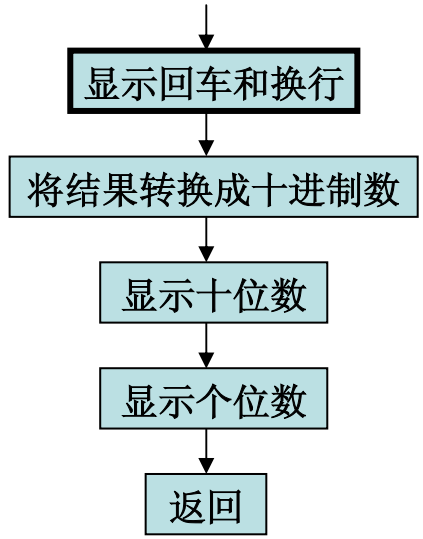
```
DATA SEGMENT
        D_N     DB      0   ；数字个数
        L_N     DB      0   ；字母个数
        O_N     DB      0   ；其它字符个数
DATA ENDS

CODE SEGMENT
MAIN PROC FAR
    ASSUME  CS:CODE, DS:DATA
        PUSH DS
        SUB AX, AX
        PUSH AX
        MOV AX, DATA
        MOV DS, AX
```



```
CHAR:   MOV AH, 1          ；接收字符到AL
        INT 21H
        CMP AL, 0DH        ；判断字符是否为回车符
        JZ EXIT
        CMP AL, 61H        ；判断字符是否<‘a’
        JL L09
        CMP AL, 7AH        ；判断字符是否>‘z’
        JG L09
        INC L_N            ；将字母数＋1
        JMP CHAR
L09:    CMP AL, 30H        ；判断字符是否<‘0’
        JL ONUM
        CMP AL, 39H        ；判断字符是否>‘9’
        JG ONUM
        INC D_N            ；将数字数＋1
        JMP CHAR
ONUM:   INC O_N            ；将其它字符数＋1
        JMP CHAR
EXIT:   MOV AL, D_N        ；显示结果
        CALL SHOWVAL
        MOV AL, L_N
        CALL SHOWVAL
        MOV AL, O_N
        CALL SHOWVAL
MAIN ENDP
```

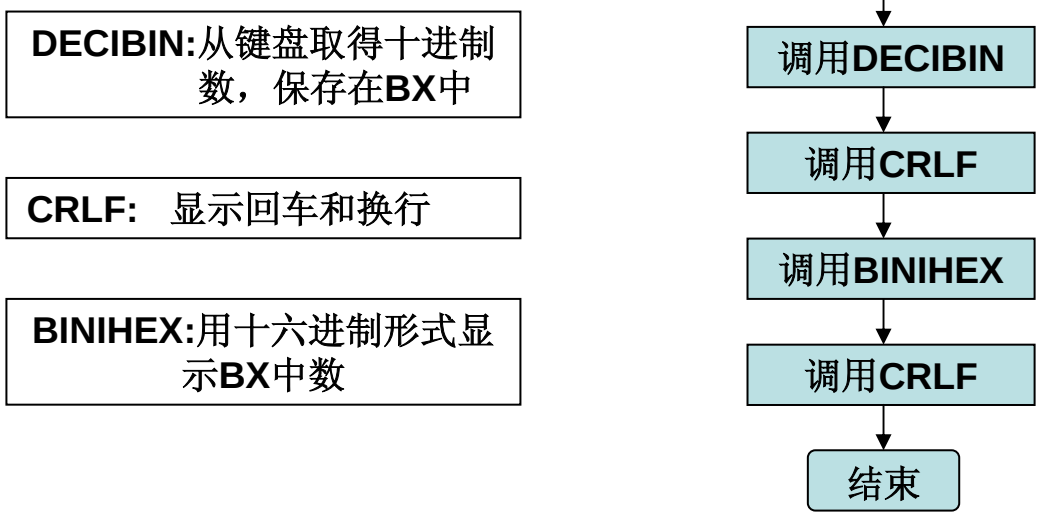


```
PROC SHOWVAL NEAR          ;子程序：显示结果
        CALL CRLF          ；显示回车和换行
        MOV AH,0           ；将结果转换为十进制数
        MOV BL, 10
        IDIV BL
        MOV BX, AX
        ADD BX, 3030H      ；将结果转换为ASCII码
        MOV DL, BL         ;显示十位数
        MOV AH, 2
        INT 21H
        MOV DL, BH         ;显示个位数
        INT 21H
        RET
SHOWRET ENDP
CRLF PROC NEAR             ；子程序：显示回车和换行
        MOV DL, 0DH        ；显示回车
        MOV AH, 2
        INT 21H
        MOV DL, 0AH        ；显示换行
        MOV AH,2
        INT 21H
        RET
CRLF ENDP
CODE ENDS
    MAIN END
```



子程序设计

- 调用：**CALL** 子程序名
- 子程序返回：**RET**
- 例：十进制数转换程序。程序要求从键盘取得一个十进制数，然后把该数以十六进制形式在屏幕上显示出来。



子程序（过程）　　主程序

```
DECIBIN SEGMENT
    ASSUME CS:DECIHEX
MAIN PROC FAR                ;主程序
AGAIN:  CALL DECIBIN
        CALL CRLF
        CALL BINIHEX
        CALL CRLF
        JMP AGAIN
MAIN ENDP
CRLF PROC NEAR               ；子程序：显示回车和换行
        MOV DL, 0DH                  ；显示回车
        MOV AH, 2
        INT 21H
        MOV DL, 0AH                  ；显示换行
        MOV AH,2
        INT 21H
        RET
CRLF ENDP
```

```
DECIBIN PROC NEAR        ；子程序：从键盘接收十进制数到BX中
        MOV BX , 0
CHAR:   MOV AH, 1                ；接收字符到AL
        INT 21H
        SUB AL, '0'              ；字符转换成数据
        JL EXIT                  ；数据<0则退出
        CMP AL, 9
        JG EXIT                  ；数据>9则退出
        CBW
        XCHG AX, BX              ；(BX) ← (BX) *10 + 数据
        MOV CX,10
        MUL CX
        XCHG AX, BX
        ADD BX, AX
        JMP CHAR                 ；接收下一个字符
EXIT:   RET
DECIBIN ENDP
```

```
BINIHEX PROC NEAR
        MOV CH, 4          ；设置循环次数：4次
ROTATE:MOV CL, 4           ；(AL) ← (BL)低4b
        ROL BX, CL
        MOV AL, BL
        AND AL, 0FH
        ADD AL, 30H        ；转换(AL)为ASCII码
        CMP AL, 3AH
        JL PRINTIT
        ADD AL, 7H
PRINTIT: MOV DL, AL        ；输出(AL)中字符
        MOV AH, 2
        INT 21H
        DEC CH             ；循环次数－1
        JNZ ROTATE
        RET
BINIHEX ENDP
DECIBIN ENDS
        END MAIN
```