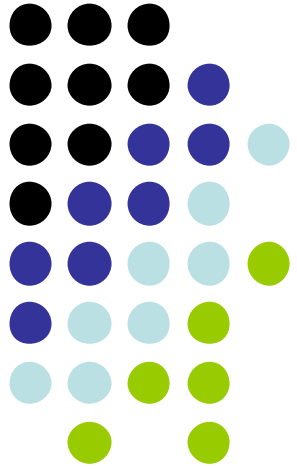
Southwest Jiaotong University

第三章

8086/8088的寻址方式和指令系统



本章学习目的

1. 掌握8086的指令格式和寻址方式;
2. 掌握数据传送、算术运算、逻辑运算与移位、串操作、控制转移、处理器控制等指令的功能、格式及各条指令执行后对标志F的影响;
3. 掌握基本DOS功能调用(键盘输入、显示输出、屏幕设置等)及返回指令的使用方法;
4. 能用学过的指令语句写出能完成一定功能的程序片段。

概　述

- 指令: 是计算机用以控制各部件协调动作的命令;
- 指令系统: CPU可执行的指令的集合;
- 机器指令:是CPU仅能识别的指令的二进制代码,也称机器码;
- 指令格式:由操作码和操作数两部分组成,有些指令无操作数。操作码规定了指令的操作性质,用助记符表示; 操作数规定了指令的操作对象。

第一节　指令的基本格式

指令的符号用规定的英文字母组成，称为**助记符**。每条指令都是由二进制代码(即指令码)组成的，计算机通过解释每一条指令的含义，来执行指令所规定的各种操作。为此，每条指令应包括下列基本信息：

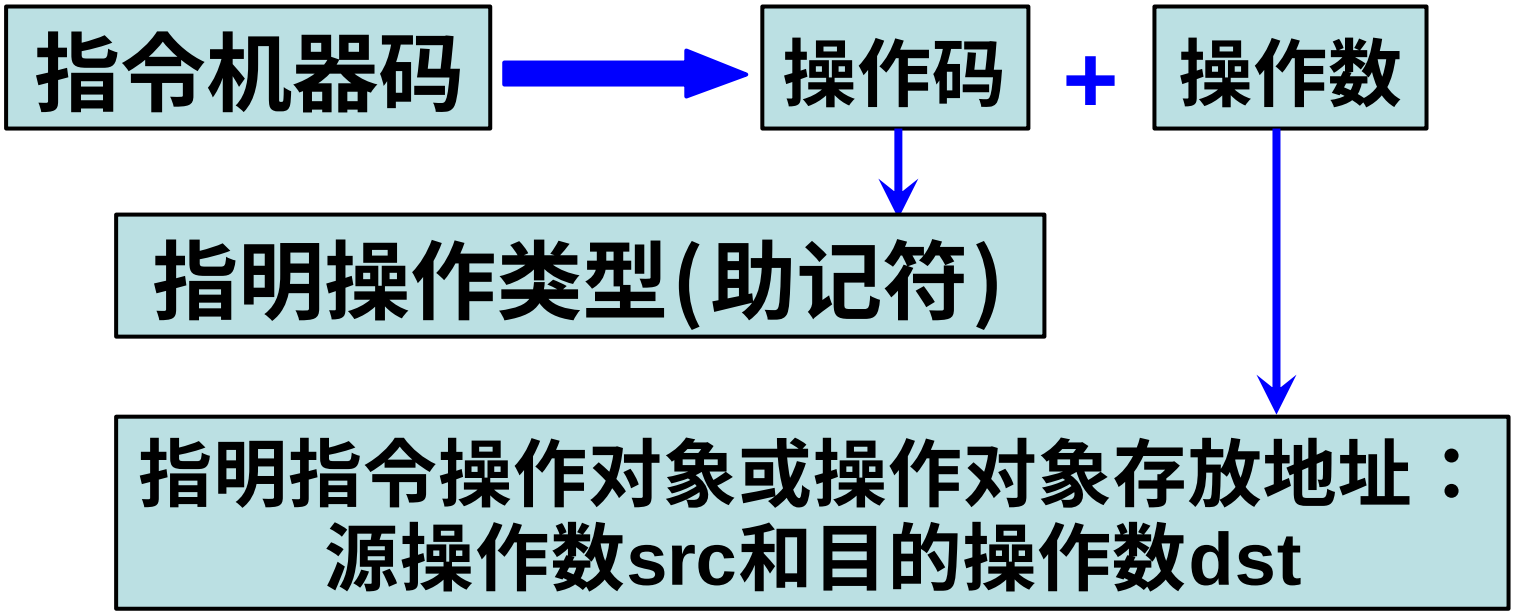
① 指明计算机的有关部件执行什么操作。

② 指明参加操作的是一些什么样的数。

③ 指明这些操作数存放在什么地方，以及通过什么方式才能够找到它们。

④ 指明后继指令从哪里取出。

一、机器码格式



8086指令的机器码是可变长的,有1～6个字节。

其排列次序如下：

操作码(OP)	操作数(OD)1	操作数(OD)2	……

第一个字节为操作码，后面的部分为操作数。

1. 操作码(Operation Code，简称OP)字段

该字段指明计算机所要执行的操作类型。在机器语言中由一组二进制代码表示，在汇编语言中用助记符代表。

例如，加法操作用“ADD”，乘法操作用“MUL”。

2. 操作数(Operated Data,简称OD)字段

指令执行的操作过程中所需的操作数,操作数字段可以是操作数本身、操作数地址或操作数地址的运算方法,还可以是指向操作数地址的指针或其它的有关操作数的信息。

8086指令可分为无操作数指令、单操作数指令、双操作数指令 。

一个操作数称为单操作数,两个操作数称为双操作数。源操作数src(Source)和目的操作数dst(Destnation)。在指令执行前,src和dst均可以是参加运算处理的操作数,指令执行后,dst中则存放运算处理的结果。

- 如果操作数直接在OD字段中给出，此数据称作立即数，指令把它拿来直接处理。
- 如果操作数存放在CPU的寄存器中，则只要给出寄存器号。
- 如果操作数存放在存储器中或I/O端口中，则要给出有效地址EA（Effect Address），或者给出形成有效地址的方法。
- 指令中操作数究竟以何种形式给出，以及操作数的地址如何形成，这就是靠寻址方式——寻找操作数的方式描述的。

二、8086/8088符号指令的书写格式

8086符号指令语句格式如下：

[标号:] 指令助记符 操作数 [; 注释]

其中带[ ]号的标号和注释可以缺省。

- 标号　后面必须跟冒号，它可缺省，是可供选择的自定义标志符。标识符遵循“:”以字母打头，不超过31个字符长度。
- 指令助记符　是指令功能的代表符号，它是指令语句的关键字，不可缺省。
- 操作数　是参加本操作的数据。
- 注释　是以“;”开头的说明部分，可以用英文或中文书写，注释允许缺省。

第二节 8086/8088 的寻址方式

- 指令中的操作数字段给出操作数的逻辑地址;
- 高级语言及其多种数据结构需要处理器提供有效而灵活的数据访问方法;
- 寻址方式的实质是由指令中的逻辑地址生成物理地址的方法(即PA与EA的关系)。

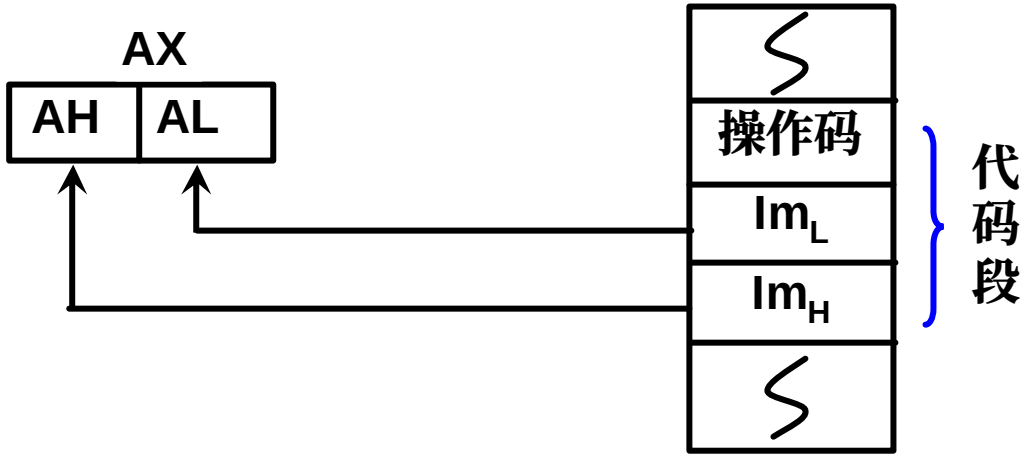
8086/8088的操作数可位于寄存器、存储器或I/O端口中。对位于存储器的操作数可采用多种不同方式进行寻址。

一条指令一般都有源操作数(src)和目的操作数(dst),要说明一条指令为何种寻址方式,应就源操作数(src)和目的操作数(dst)分别加以说明。

一、立即数寻址方式(Immediate Addressing)

在这种寻址方式下,操作数直接包含在指令中,它是一个8位或16位的常数,也叫立即数。这类指令翻译成机器码时,立即数作为指令的一部分,紧跟在操作码之后,存放在代码段内。如果立即数是16位数,则高字节存放在代码段的高地址单元中,低字节放在低地址单元中。

MOV AX，Im



它表示将立即数Im送到AX寄存器中，其中将Im的低8位（用Im_L表示）送到AL中，将Im的高8位（用Im_H表示）送到AH中。

立即数Im可以是8位的，也可以是16位的。

例如：

```
MOV  AX，1234H   ;执行后AX＝1234H，
                 ;其中AH＝12H,AL＝34H。

MOV  AX，1000    ;执行后AX＝03E8H，
                 ;其中AH＝03H,AL＝0E8H。

MOV  AL，80H     ;执行后AL＝80H。
```

以A～F打头的数字出现在指令中时,前面一定要加一个数字0。

```
MOV  AX，0FF00H  ;执行后AX＝0FF00H 。
```

特点及注意事项

1.执行速度快：操作数是直接从指令中取得，不执行总线周期，所以执行速度快；

2.立即数只能作源操作数，不能作目的操作数；

3.主要用来给寄存器或存储器赋初值。

源操作数与目的操作数类型必须匹配

8位（字节）立即数可装入8位或16位寄存器，16位（字）立即数只能装入16位寄存器。**例如：**

MOV AL，14H ®执行后，AL=14H

MOV AX，14H ®执行后，AX=0014H

MOV BP，1234H ®执行后，BP=1234H

但指令MOV AL，1234H 或 MOV BL，123H是错误的，因为立即数1234H或123H都不能用8位二进制数表示，而AL、BL都是8位寄存器。

二、寄存器寻址方式(Register Addressing)

操作数存放在寄存器中,由指令指定寄存器的名称。

16位寄存器：AX、BX、CX、DX、SI、DI、SP和BP等——存放16位操作数。

8位寄存器：AH、AL、BH、BL、CH、CL、DH和DL——存放8位操作数。

MOV CX，AX ;16位寄存器传送,AX→CX

假设该指令执行前AX＝1234H,CX＝5678H,**则指令执行后,**CX＝1234H,**而**AX**的内容保持不变。**

MOV DL，CL ;8位寄存器传送,CL→DL

寄存器寻址方式

特点

1. 执行速度快：因操作数就在CPU内部，不执行总线周期；
2. 源操作数和目的操作数都可使用寄存器寻址。

注意：源操作数的长度必须与目的操作数一致，否则会出错。

例如，MOV CX，AH ×

尽管CX寄存器放得下AH的内容，但汇编程序不知道将它放到CH还是CL。

三、固定寻址(Inherent Addressing)

8086的单操作数指令,其操作是规定在CPU中某个固定的寄存器中进行,这个寄存器又被隐含在操作码中,因此,这种寻址方式的指令大多为单字节指令。

加减法的十进制调整指令DAA,其操作总是固定在AL寄存器中进行;

还有一种寻址方式为双操作数指令,例如寄存器入栈和出栈指令:

```
PUSH  AX  ;入栈指令,源操作数为寄存器寻址,
            目的操作数地址固定为堆栈栈顶。
POP   AX  ;出栈指令,源操作数地址固定为堆栈栈顶,
            目的操作数为寄存器寻址。
```

❖ 还有乘、除法指令虽不是单字节指令，但被乘数、被除数总是放在AL或AX中。固定寻址指令中，对固定操作数是被隐含了的。

特点

1. 这种寻址方式的指令大多为单字节指令；
2. 执行速度快，因固定寻址的指令不需要计算EA(有效地值)，且部分操作发生在CPU内部。

四、存储器寻址

存储器寻址的指令,其操作数总是在代码段之外的**数据段**、**堆栈段**或**附加段**的存储器中,指令给出的是操作数寻址信息(逻辑地址),也就是操作数所在存储单元的地址信息。

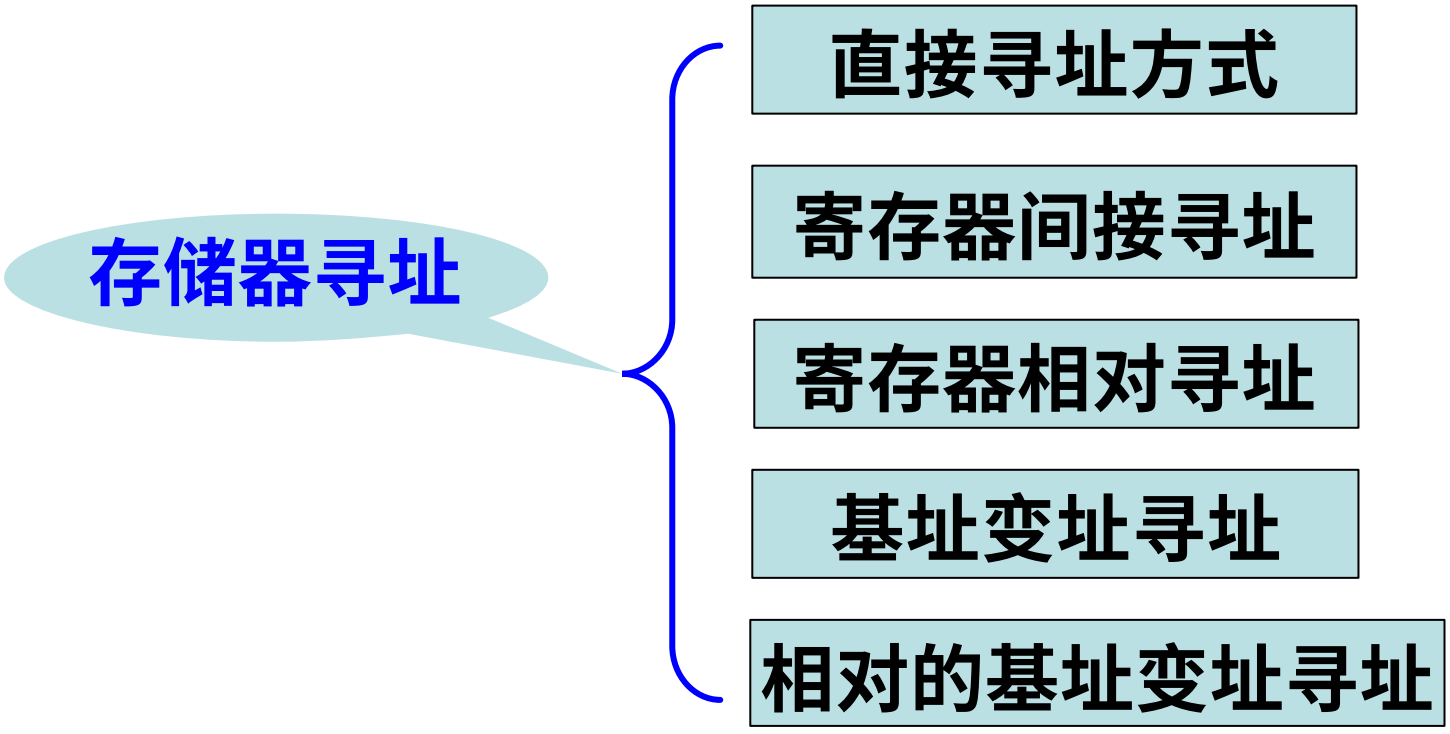
执行步骤

CPU首先根据操作数字段提供的地址信息,由执行部件EU计算出有效地址EA

再由总线接口单元BIU根据公蕞騱3 W_ 'l W_

一条指令中，最多只能有一个存储器操作数，或者是源操作数，或者是目的操作数。

存储器寻址方式共有24种，按EA的计算方法不同又可分为以下几种。



1. 直接寻址方式(Direct Addressing)

在直接寻址方式中,操作数所在存储单元的有效地址直接由指令给出。有效地址是机器码的一部分,它存放在代码段中指令的操作码之后。

要注意的是当采用直接寻址指令时,如果指令中没有用前缀指明操作数存放在哪一段,则默认为使用的段寄存器为数据段寄存器DS,因此,操作数的物理地址PA＝16×DS + EA,即10H×DS + EA。指令中有效地址上必须加一个方括号,以便与立即数相区别。

例： MOV AX，[2000H]

指令机器码为0A1H、00H、20H,执行过程如下:

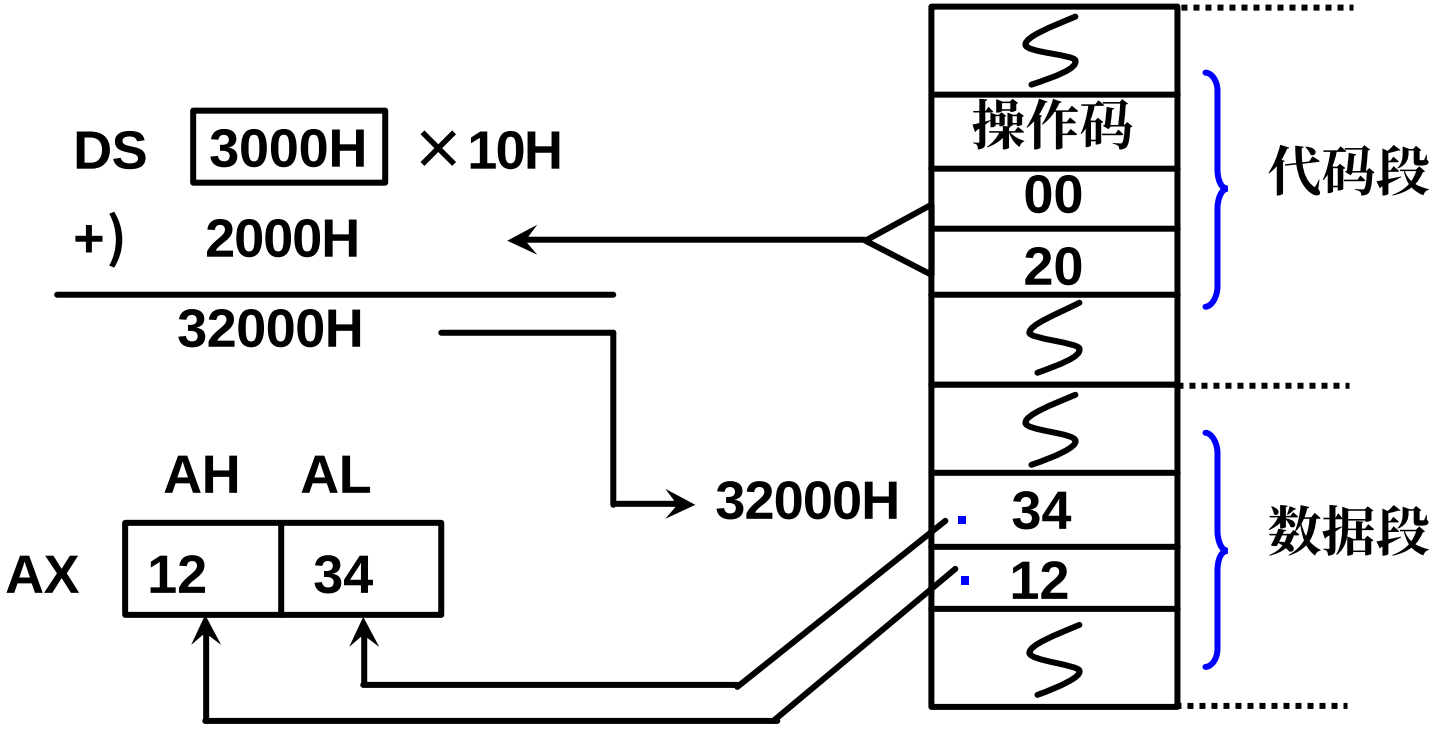
STEP1: 取指令机器码0A1H、00H、20H;

STEP2: CPU分析操作码0A1H后得知本指令功能为数据传送指令;

STEP3: 计算操作数所在内存单元物理地址PA＝16×DS+EA,EA由指令直接给出(即所谓直接寻址方式),DS为段基址寄存器,应事先设置好其值。指令中如没有特别说明,计算操作数物理地址所需段基址都由DS段寄存器给出。

STEP4: CPU执行总线操作,从PA地址单元取得数据送到AX寄存器。

设DS＝3000H,则源操作数的物理地址PA＝16 × 3000H + 2000H ＝ 32000H。若地址32000H中的内容为34H,32001H中的内容为12H,我们用(32000H)＝1234H来表示这个字。则执行指令后,AX＝1234H。



如果要对代码段、堆栈段或附加段寄存器所指出的存储区进行直接寻址,应在指令中指定段超越前缀。

例如,数据若存放在附加段中,则应在有效地址前加“ES:”,这里的冒号“:”称为修改属性运算符,计算物理地址时要用ES作基地址,而不再是默认值DS。

MOV AX，ES:[1000H]

该指令的源操作数的物理地址:

PA＝16×ES + 1000H。

在汇编语言中还允许用符号地址代替数值地址,实际上就是给存储单元取一个名字,这样,如果要与这些单元打交道,只要使用其名字即可,不必记住具体数值是多少。

MOV AX，AREA1

不过光从指令的形式上看,AREA1不仅可代表符号地址,也可以表示它是一个16位的立即数,两者之间究竟如何来区别呢?程序中还必须事先安排说明语句也叫做伪指令来加以说明。

使用这种寻址方式应注意：

①在汇编语言程序中，直接地址可用数值表示，被包括在方括号[ ]之中；直接地址也可用符号地址表示，例如，MOV AX，AREA2，这里的AREA2为符号地址。符号地址是有属性的，它由数据段中定义数据的伪指令确定。

②在直接寻址方式下，操作数地址也可位于数据段以外的其它段。此时，应在操作数地址前使用前缀指出段寄存器名，称这种前缀为段超越前缀。若上例中的AREA2位于附加段中，则应写为：

MOV AL，ES:AREA2 ；表示把附加段ES中的变量地址AREA2中的内容传送到AL。

③直接寻址方式用于处理存储器中的单个变量。

2. 寄存器间接寻址(Register Indirect Addressing)

操作数的有效地址EA是通过基址寄存器BX、BP或变址寄存器SI、DI中的任一个寄存器的内容间接得到的，即称这四个寄存器为间址寄存器：

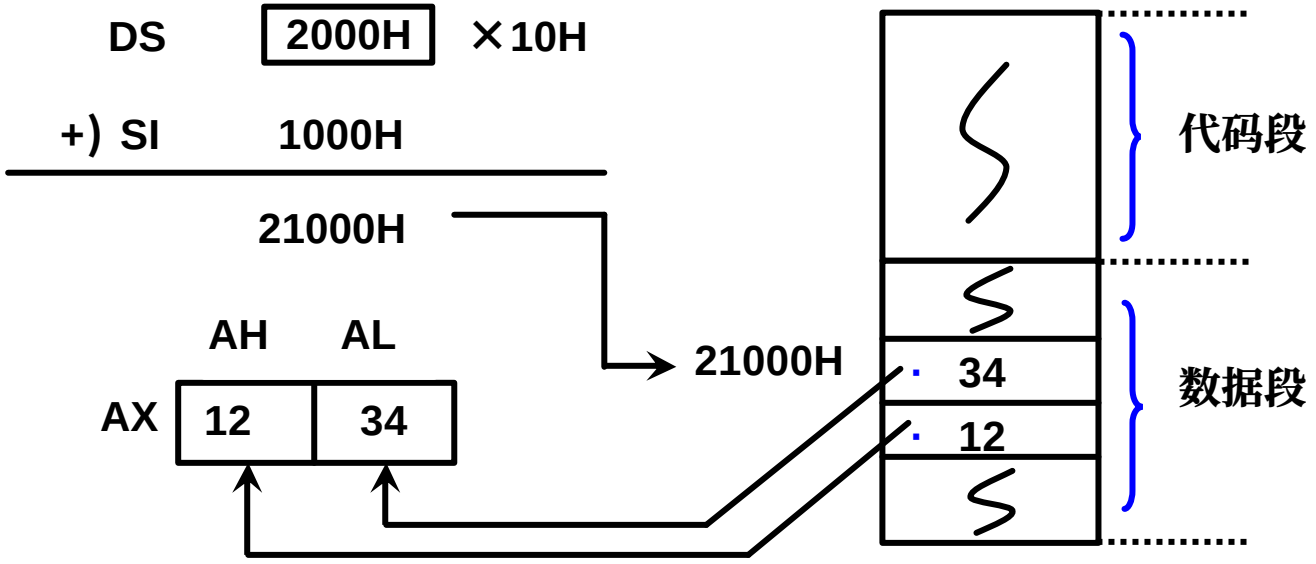
$$EA=\begin{Bmatrix}BX\\BP\\SI\\DI\end{Bmatrix}$$

① 指令中指定BX、SI、DI为间址寄存器，则操作数在数据段中。这种情况下用DS寄存器内容作段首址，操作数的实际地址为：

$$PA=16\times DS+\begin{Bmatrix}BX\\SI\\DI\end{Bmatrix}$$

MOV AX，[SI]

若已知DS＝2000H，SI＝1000H，则PA＝21000H，该指令将把数据段中21000H单元和21001H相邻两个单元的内容传送到AX，如下图所示。



② 指令中若指定BP为间址寄存器,则操作数在堆栈段中。这种情况下,用SS寄存器内容作段首址,操作数的实际地址为:

PA＝16×SS + BP

使用寄存器间接寻址方式时应注意:

在指令中,也可指定**段超越前缀**来取得其它段中的操作数。例如:

MOV AX，ES:[BX]

寄存器间接寻址方式可以用来对一维数组或表格进行处理,只要改变间址寄存器BX,BP,SI,DI中的内容,用一条寄存器间接寻址指令就可对连续的存储器单元进行存/取操作。

3.寄存器相对寻址(Register Relative Addressing)

指定BX,BP,SI,DI的内容进行间接寻址。但是,和寄存器间接寻址方式不同的是:指令中还要指定一个 8 位或 16 位的位移量 DISP(Displacement),操作数的有效地址EA则是等于间址寄存器内容和位移量之和,结果按16位归算。

$$EA = \begin{Bmatrix} BX \\ BP \\ SI \\ DI \end{Bmatrix} + \begin{Bmatrix} DISP_8 \\ DISP_{16} \end{Bmatrix}$$

对于寄存器为BX、SI、DI的情况,用段寄存器DS的内容作段首址；而对于寄存器BP,则使用段寄存器SS的内容作段首址。操作数的实际地址为:

$$PA = 16 \times DS + \left\{ \begin{matrix} BX \\ SI \\ DI \end{matrix} \right\} + \left\{ \begin{matrix} DISP_8 \\ \\ DISP_{16} \end{matrix} \right\}$$

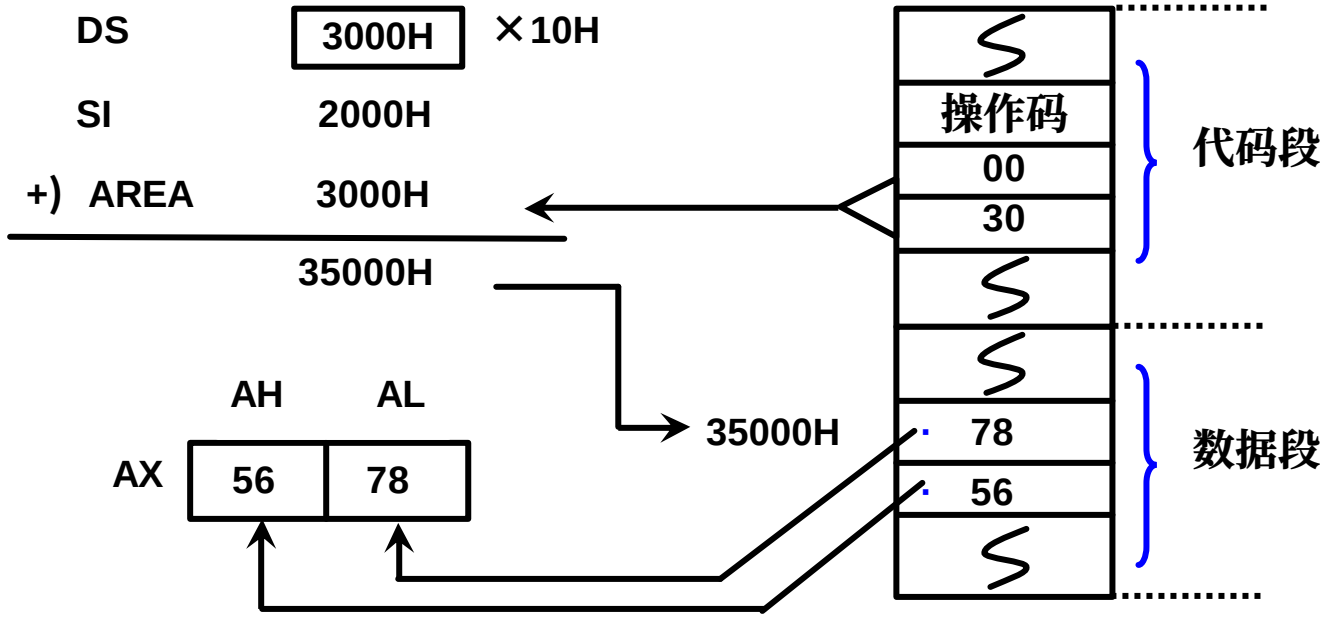
$$PA = 16 \times SS + BP + \left\{ \begin{matrix} DISP_8 \\ \\ DISP_{16} \end{matrix} \right\}$$

寄存器相对寻址通常用来访问数组中的元素。

MOV AX，AREA[SI]

;位移量用符号AREA代表的值表示。

[例]设DS＝3000H，SI＝2000H，AREA＝3000H，其寻址示意如图所示。



采用寄存相对寻址的指令,也可使用段超越前缀。

例如： MOV DL，DS:COUNT[BP]

BX、BP叫基址寄存器,因此用它们进行寻址的又叫基址寻址。SI、DI叫变址寄存器,用它们进行寻址的又叫变址寻址。在处理数组时,SI用于源数组的变址寻址。DI则用于目的数组的变址寻址。

[例] MOV AX，ARRAY1[SI]

MOV ARRAY2[DI]，AX

其中:ARRAY1和ARRAY2为不相等的位移量。若以这两条语句,配上改变SI和DI值的指令,构成循环,便可实现将源数组搬家到目的数组的操作。

在计算EA时，位移量若为8位$DISP_8$（以补码形式出现，数值范围为–128～+127），则应将其变成16位补码方能相加；位移量若为16位$DISP_{16}$，则直接相加。结果取低16位有效。

4. 基址变址寻址(Based Indexed Addressing)

通常将BX和BP看作基址寄存器,将SI、DI看作变址寄存器。

用这种寻址方式,存储器操作数的有效地址EA是由指令指定的一个基址寄存器和一个变址寄存器的内容之和确定。

$$EA = \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix}$$

结果按16位归算(只取结果的低16位有效)。

基址在BX,段寄存器使用DS；基址在BP中,段寄存器使用SS。

$$PA = 16 \times DS + BX + \begin{cases} SI \\ \\ DI \end{cases}$$

$$PA = 16 \times SS + BP + \begin{cases} SI \\ \\ DI \end{cases}$$

MOV AX，[BX][SI] 或写为：MOV AX，[BX+SI]

设 DS ＝ 2000H ， BX ＝ 8000H ， SI ＝ 90FEH ，
则 BX+SI ＝ 8000H+90FEH ＝ 110FEH，取低16位有效得：

EA＝10FEH

PA＝2000H×16 + EA＝210FEH

指令执行后，将把210FEH和210FFH相邻两个单元内容送到AX。而 AL＝(210FEH)，

AH＝(210FFH)。

注意事项：

① PA计算公式中，BX可与SI或DI组合，BP也可与SI或DI组合，但BX不可与BP组合。BX的缺省段基址寄存器为DS，而BP的缺省段基址寄存器为SS；

② 基址变址寻址方式也可使用段超越前缀；

例如：　MOV CX，ES:[BX][SI]

③ 基址变址寻址方式同样适合数组或表格的处理，由于基址和变址寄存器中的内容都可以修改，因而在处理二维数组时特别方便。

基址变址寻址时，也允许带一个8位或16位的位移量，带位移量的基址加变址寻址又称为相对的基址变址寻址。

5. 相对的基址变址寻址

操作数的有效地址EA是由指令指定的一个基址寄存器和一个变址寄存器的内容再加上8位或16位位移量之和，结果按16位归算。

$$PA = 16 \times DS + BX + \begin{Bmatrix} SI \\ \\ DI \end{Bmatrix} + \begin{Bmatrix} DISP_8 \\ \\ DISP_{16} \end{Bmatrix}$$

$$PA = 16 \times SS + BP + \begin{Bmatrix} SI \\ \\ DI \end{Bmatrix} + \begin{Bmatrix} DISP_8 \\ \\ DISP_{16} \end{Bmatrix}$$

例：　　MOV AX，MASK[BX][SI]

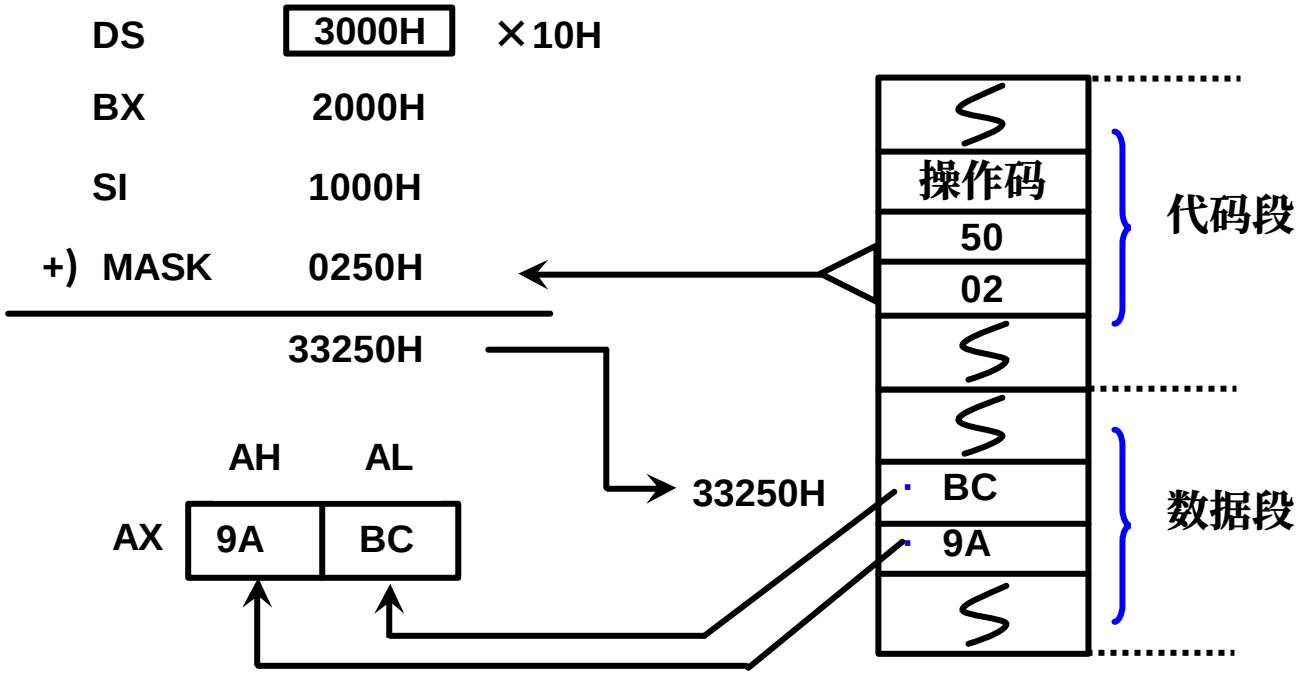
或写为：　MOV AX，MASK[BX+SI]

也可写为：MOV AX，[BX+SI+MASK]

设DS＝3000H，BX＝2000H，SI＝1000H，

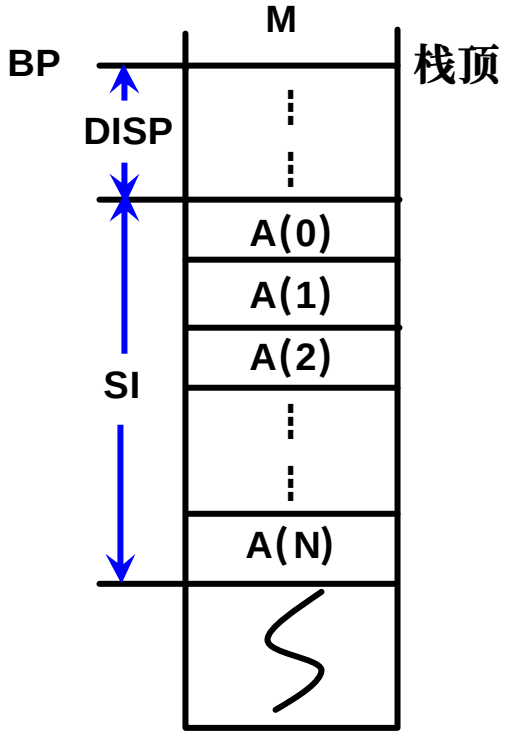
MASK＝0250H,则源操作数的实际地址为：

PA＝30000H+(2000H+1000H+0250H)＝33250H


DS
3000H
×10H
BX
2000H
SI
1000H
+) MASK
0250H
33250H
AH
AL
AX
9A
BC
33250H
操作码
50
02
BC
9A
代码段
数据段

若用段超越前缀,也可用其它段寄存器作为地址基准。

这种寻址方式为访问堆栈中的数组提供了方便,可以在基址寄存器BP中存放堆栈顶的地址,用位移量表示栈顶到数组第一个元素的距离,变址寄存器则用来访问数组中的每一个元素。



6. 串寻址(String Addressing)

只用于数据串操作指令。数据串操作指令中,不能使用上述的存储器寻址方式存取所使用的操作数,而是应用了一种隐含的变址寄存器寻址。8086规定,当执行串操作指令时,用源变址寄存器SI指向源串的第一个字节或字,用目的变址寄存器DI指向目的串的第一个字节或字。在重复串操作中,CPU自动地调整SI和DI,以顺序寻址字节串或字串操作数。

```
MOVSB    ;隐含使用SI和DI分别指向源串和目的串,
         ;实现字节串的传送。
MOVSW    ;隐含使用SI和DI指向源串和目的串,
         ;实现字串的传送。
```

五、相对寻址

指令给出的形式地址与指令指针IP的内容相加,形成操作数的有效地址称为相对地址。实际上它是以指令指针IP作为变址寄存器,相对寻址方式是由寻址特征码指定。

相对寻址的过程与变址寻址过程类似,这是根据形式地址提供的偏移量DISP(可正可负),相对于当前地址进行上、下浮动,其有效地址的计算公式如下:

有效地址值EA＝当前的IP指针 + 偏移量DISP

这种寻址方式使程序模块可以采用浮动地址,编程时只要确定程序内部相对距离即可,很有实用价值。

六、I/O端口寻址

8086采用独立编址的I/O端口，可有64 K个字节端口或32 K个字端口，用专门的IN和OUT指令访问。

1. 直接端口寻址

用直接端口寻址的IN和OUT指令均为双字节指令，在第二字节中存放端口的直接地址。因此，直接端口寻址的端口数为0～255个。

```
IN  AL，60H    ;字节输入(60H)→AL
IN  AX，60H    ;字输入(60H)→AL,(61H)→AH
```

OUT指令和IN指令一样，都提供了字节和字两种使用方式。

直接端口地址可用两位16进制数值表示，但不被包括在任何括号中，不能理解为立即数；直接端口地址也可用符号表示。

```
OUT  PORT，AL     ;字节输出
OUT  PORT，AX     ;字输出
```

直接端口寻址的IN和OUT指令为双字节，和下面介绍的间接端口寻址的单字节IN和OUT指令比较，又称为长格式的I/O指令。

2. 寄存器间接端口寻址

当端口地址≥256时，长格式的I/O指令不能实现对此端口的寻址，先把端口号放到寄存器DX中，端口号可从0000～0FFFFH，间接寻址端口的寄存器只能用DX。寄存器间接寻址端口的IN和OUT指令均为单字节，故又称为短格式的I/O指令。

```
MOV  DX,383H  ;将端口号383H放入DX，
OUT  DX,AL    ;将AL输出到DX所指的端口中。

MOV  DX,380H  ;将端口号380H放入DX，
IN   AX,DX    ;从DX和DX+1所指的两个端口输入一
              ;个字到AX中，AL＝(380H)，AH＝(381H)
```

寻址方式小结

如何找到指令中的操作数呢?只需找到存放这些操作数的地址即可。在涉及到操作数的地址时,常常要在指令中使用方括号,有关带方括号的地址表达式必须遵循下列规则:

① 立即数可以出现在方括号内,表示直接地址,如[2000H]。

② 只有BX、BP、SI、DI这四个寄存器可以出现在[ ]内,它们可以单独出现,也可以由几个寄存器组合起来(只能相加)M或以寄存器与常数相加的形式出现,但BX和BP寄存器不允许出现在同一个[ ]内,SI和DI也不能同时出现。

③ 由于方括号有相加的含义,下面几种写法都是等价的:

6[BX][SI]

[BX+6][SI]

[BX+SI+6]

④ 若方括号内包含BP,则隐含使用SS来提供基地址,它们的物理地址的计算方法为:

物理地址PA＝16×SS + EA

包含BP的操作数有下面三种形式：

DISP[BP+SI]　　；EA＝BP+SI+DISP

DISP[BP+DI]　　；EA＝BP+DI+DISP

DISP[BP]　　；EA＝BP+DISP

其中,DISP表示8位或16位位移量,也可以为0。

也允许用段超越前缀将SS修改为CS、DS或ES中的一个,在计算物理地址时,应将上式中的SS改为相应的段寄存器。

其余情况均隐含使用DS来提供基地址,它们的物理地址的计算方法为: PA＝16×DS + EA

这类操作数可以有以下几种形式:

```
[DISP]              ; EA＝DISP
DISP[BX+SI]         ; EA＝BX+SI+DISP
DISP[BX+DI]         ; EA＝BX+DI+DISP
DISP[BX]            ; EA＝BX+DISP
DISP[SI]            ; EA＝SI+DISP
DISP[DI]            ; EA＝DI+DISP
```

同样,也可用段超越前缀将上式中的DS修改为CS、ES或SS中的一个。

注意: 计算EA时,相加以后的结果一定要按16位归算。

上面讲解的寻址方式基本上都是针对源操作数而言，而一条指令往往既有源操作数又有目的操作数，所以要说明一条指令的寻址方式，需将源操作数、目的操作数分别加以说明。若操作数为存储器操作数，在不太清楚它为那一小类时，干脆叫它为存储器寻址方式，并写出其EA和PA。

寻址方式说明举例


```
MOV  AX，CX
;源操作数为寄存器寻址，
;目的源操作数为寄存器寻址。

MOV  [BX+DI+6]，AL
;源操作数为寄存器寻址，
;目的操作数为存储器寻址，
;EA＝BX+DI+6,PA＝DS×16 + EA。
```

何种寻址方式?

```
PUSH  AX    ;
POP   BX    ;
```

指令的执行时间

- 执行每一条指令都需要一定的时间(用时钟周期T的倍数表示),见教材附录B。一条指令的执行时间由下式表示：
- 总时间＝EU基本执行时间+计算EA时间+执行总线R/W周期时间
- 不同功能的指令,其基本执行时间不同；就是同一类指令,因寻址方式不同,访问存储器次数不一样,执行时间也不同,参见教材附录B。
- EA的表达式越复杂,则计算EA的时间就越长,从而指令的执行时间也就变长。

- 若操作数在内存或I/O端口是规则存放,则访问只需一个总线周期(4个T)；若为非规则存放,则需另外增加一个总线周期。
- 对于给定的系统,已知T及寻址方式就能精确计算出每条指令的执行时间,进而得知一段乃至整个程序的执行时间。
- 现代计算机由于硬件的进步,一般不会精确计算指令的执行时间。

第三节 8086/8088指令系统

8086/8088指令系统包含133条基本指令，这些指令与寻址方式组合，再加上不同的数据形式——有的为字处理，有的为字节处理，可构成上千种指令。这些指令按功能可分为六类：

① 数据传送类；
② 算术运算类；
③ 逻辑运算与移位类；
④ 串操作类；
⑤ 控制转移类；
⑥ 处理器控制类。

和8位微处理器比较,功能有了很大扩充主要表现在:

① 有8个通用寄存器均可作累加器使用;

② 可进行字节或字的处理;

③ 有重复指令和乘除运算指令;

④ 扩充了条件转移,移位/循环指令;

⑤ 可进行带符号数的运算;

⑥ 有软中断和协调多处理器工作的指令。

- 学习指令系统，对每条指令的助记符要记忆，书写要正确；
- 对操作数的正确书写和认识其为何种类型；
- 指令执行后对标志F的影响要非常清楚。
- 要学好指令系统，必须亲自动手写指令和程序，实验时还要多上机调试。

为方便讲解指令而要用到的一些符号做一介绍:

R　如果是8位操作,则表示AH、AL、BH、BL、CH、CL、DH、DL;

若是16位操作,则表示AX、BX、CX、DX、SP、BP、SI、DI。

SP　堆栈指针。

段R　段寄存器CS、DS、ES、SS。

M　存储器操作数。

M/R　存储器或寄存器操作数。

port	一个输入/输出(I/O)端口,用数字或表达式表示,端口号≤255。
disp	8位或16位位移量,在汇编语言中常用符号地址来表示。
Im	8位或16位立即数。
src，dst	源和目的操作数。
( )	用来表示存储单元或I/O端口中的内容。

一、数据传送(Data Transfer)类指令

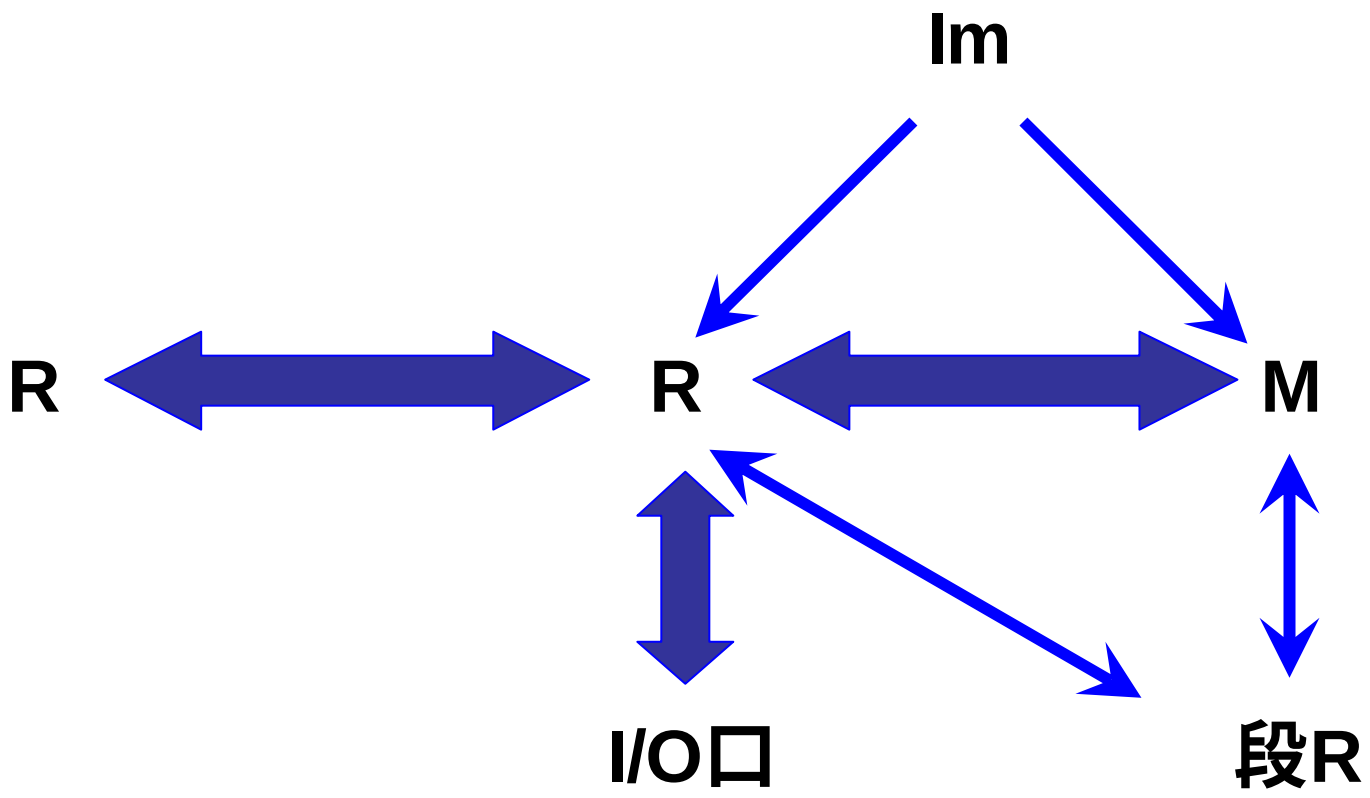
功　能：

实现CPU内部寄存器之间、CPU和存储器之间以及CPU和I/O端口之间的数据传送。

特 点：

① 含两个操作数；

② 除SAHF和POPF指令外，数据传送指令不影响标志寄存器F内容。


Im
R
R
M
I/O口
段R

1. 通用传送指令

西南交通大學 Southwest Jiaotong University

(1)传送指令

指令格式：MOV dst，src ；dst ← src

其中,源操作数src和目的源操作数dst均可采用多种寻址方式,不影响标志寄存器F。

- R « R ;实例1
- R « 段R ;实例2
- R ← Im ;实例3
- M ← Im ;实例4
- R « M ;实例5
- 段R « M ;实例6

NEXT

R « R传送实例


```
MOV  AX，BX        ;BX中16位数据传送到AX
MOV  SI，  BP      ;BP中16位数据传送到SI
MOV  AH，BL        ;BL中8位数据传送到AH
```

返回

R « 段R传送实例


```
MOV  DS，AX     ;AX中16位数据传送到DS
MOV  AX，ES     ;ES中16位数据传送到AX
```

返回

R ¬ Im传送实例


```
MOV  AX,03FFH     ;执行后AX＝03FFH
MOV  SP,2000H     ;执行后SP＝2000H
MOV  BX,1000      ;执行后BX＝03E8H
MOV  SI,057BH     ;执行后SI＝057BH
MOV  CL,'*'       ;执行后CL＝2AH(*的ASCII码)
```

返回

M ¬ Im传送实例

西南交通大学 Southwest Jiaotong University

MOV [2000H]，6789H

;执行后(2000H)＝89H,(2001H)＝67H

MOV WORD PTR[SI],1234H

;立即数1234H送DS段SI和SI+1所指的两存储单元

运算符PTR用于修改存储器操作数的类型属性,如例中的WORD PTR[SI]定义从SI所指单元开始的连续两个单元为字变量。

返回

R « M传送实例


```
MOV AL，BUFFER
    ;将DS段BUFFER单元的内容送AL
MOV AX，[SI]
    ;将DS段SI和SI+1所指单元的内容送AX

MOV [DI]，CX
    ;将CX的内容送DS段DI和DI+1所指的存储单元
MOV SI，BLOCK[BP]
    ;将SS段EA为BP+BLOCK字单元的内容送SI
```

返回

段R « M传送实例


```
MOV DS，[2000H]
   ;将2000H和2001H两存储单元的内容送DS
MOV [BX][SI]，CS
   ;CS内容送DS段BX+SI所指的字存储单元

MOV DS，DATA[BX+SI]
   ;将DS段BX+SI+DATA所指的字单元的内容送DS
MOV DEST[BP+DI]，ES
   ;ES内容送SS段BP+DI+DEST所指的字存储单元
```

返回

需要注意的问题:

对于MOV指令的使用,有几点需要注意:

① 操作数的类型要匹配,要么8位,要么16位,这决定于R是8位还是16位,也取于立即数的形式。下面的使用是错误的:

MOV BX，AL ×

MOV CL，1234H ×

② 不允许在段R之间直接传送信息。

③ 不能用CS和IP作目的操作数,CS和IP的内容可以了解,但不能随便修改。

④ 不允许用立即数作dst。

⑤ MOV指令中的dst和src不能同时为M。

如何实现M « M呢?可以借助R为桥梁:

```
MOV  AL，[SI]    ;通过AL实现两存储单元
MOV  [DI]，AL    ;间的8位信息传送。
```

⑥ 不能向段寄存器送立即数,因此当要对段寄存器初始化赋值时,也要通过R来过渡。例如:

```
MOV  AX，DATA        ;将段地址DATA通过
MOV  DS，AX          ;AX赋给DS
```

(2)堆栈操作指令

指令格式: PUSH src

指令功能:

SP ← SP–1,(SP) ← src_H,SP ← SP–1,(SP) ← src_L

标志寄存器:不影响

指令格式: POP dst

指令功能:

dst_L ← (SP),SP ← SP+1,dst_H ← (SP),SP ← SP+1

标志寄存器:不影响,POPF除外。

入栈和出栈指令都有4种格式:

① CPU通用寄存器进/出栈(PUSH/POP R)

```
PUSH   AX
POP    BX
```

② 段寄存器进/出栈(PUSH/POP 段R)

```
PUSH   CS
POP    DS
```

③ 存储器单元进/出栈(PUSH/POP M)

```
PUSH   [SI]
POP    [2000H]
```

④ 标志F的内容也可以进/出栈

```
PUSHF
POPF
```

进栈和出栈指令用于程序保存或恢复数据或用于转子或中断时保护现场和恢复现场。

堆栈操作注意事项：

堆栈操作指令格式简单，但使用时须注意：

- 堆栈栈顶隐含寻址，src、dst可以是R、段R、存储器（可使用各种存储器寻址方式）或标志寄存器F。
- 8086/8088堆栈操作都是字操作，而不允许对字节操作，因此，若写PUSH AL是错误的。
- 8086/8088栈顶是所谓的实栈顶。
- 每执行一条进栈指令，SP自动减2，进栈时，高字节先进栈，执行弹出时，正好相反，弹出一个字，SP自动加2。

- CS寄存器可进栈，但不能随意弹出一个数据到CS。
- POPF执行后，要影响F的当前状态。
- 在使用堆栈操作保存多个寄存器内容和恢复多个寄存器时，要按“先进后出”原则来组织进栈和出栈的顺序。
- 堆栈的容量有限，因此进栈和出栈要成对出现，否则将有数据残留在堆栈中，时间一长，堆栈会满的！

利用指令来实现

解法1：

```
MOV   DX, AX
MOV   AX, CX
MOV   CX, BX
MOV   BX, DX
```

需要一个中间变量

解法2：

```
PUSH  AX
PUSH  BX      ; 进栈
PUSH  CX
POP   AX
POP   CX      ; 出栈
POP   BX
```

无需中间变量

堆栈、堆栈区

▲堆栈是以“先进后出”,或者LIFO“后进先出”方式工作的一个存储区,它只有一个出入口。

▲堆栈指针SP指示堆栈的出入口,SP任何时候都指向当前的栈顶。

▲SS与SP给出堆栈栈顶的逻辑地址,计算栈顶物理地址PA＝SS ´ 10H + SP

(3)交换指令(Exchange)

指令格式：　XCHG dst，src

执行操作：　dst « src

交换指令可实现CPU内部R之间,或内部R与M之间的内容(字节或字)交换,不影响标志寄存器F。

```
XCHG  AH，AL    ;AL与AH间进行字节交换
XCHG  BX，CX    ;BX与CX间进行字交换
XCHG  [2000H]，DX
                ;DX « (2000H)、(2001H)
XCHG  AX，[SI+0400H]
```

XCHG dst，src 指令应注意:

① dst 和 src 不能同时为存储器操作数；

② 任一个操作数都不能使用段寄存器和IP,也不能使用立即数。

2. 累加器专用传送指令

(1)输入/输出指令

I/O指令按指令长度分为长格式和短格式。

① 输入指令：

- 长格式　IN AL，PORT
  　　IN AX，PORT
- 短格式　IN AL，DX
  　　IN AX，DX

② 输出指令：

- 长格式　OUT PORT，AL
  　　OUT PORT，AX
- 短格式　OUT DX，AL
  　　OUT DX，AX

均不影响标志寄存器F。

使用I/O指令时应注意：

- 这类指令只能用累加器AL/AX作I/O操作的机构，不能用其它寄存器；
- 长格式的I/O指令，PORT范围为00～0FFH，大于此范围要用短格式；
- 在使用短格式I/O指令时，应先将端口地址赋给DX，而且只能用DX。

例如，从端口2F0H输入数据的程序段为：

```
MOV  DX，02F0H
IN   AX，DX
```

(2)换码(Translate–Table)指令

格式:　　XLAT

指令功能: AL ← DS:[BX+AL];

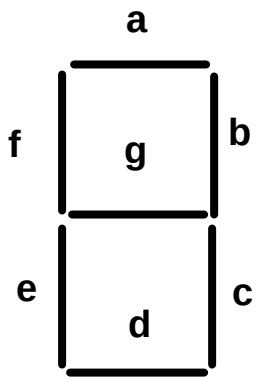
标志寄存器: 不影响;

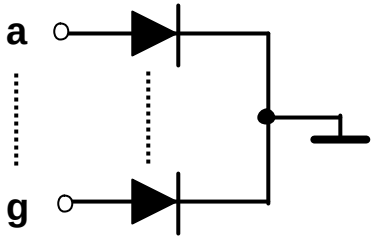
用途:用于查表,表首地址的偏移地址在BX中,表长度可达256字节。

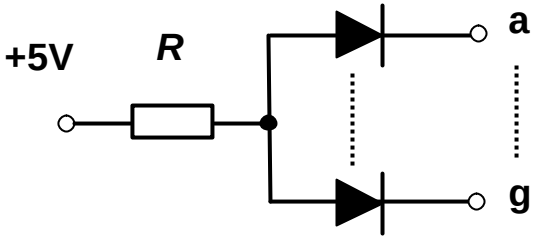
使用XLAT指令之前,要求BX寄存器指向表的首地址,AL中存放待查的码,用它表示表中某一项与表首址的距离。执行时,将BX和AL的值相加得到一个地址,最后将该地址单元中的值取到AL中,这就是查表转换的结果。

[例]十六进数0～9、A～F对应的ASCII码为:30H～39H、41H～46H,依次放在内存以TABLE开始的区域。将AL中某一位十六进数××H转换为对应的ASCII码,源程序如下：

```
MOV  BX，TABLE   ;TABLE为表首地址
MOV  AL，××H
XLAT             ;执行后,AL ¬ ASCII码
```



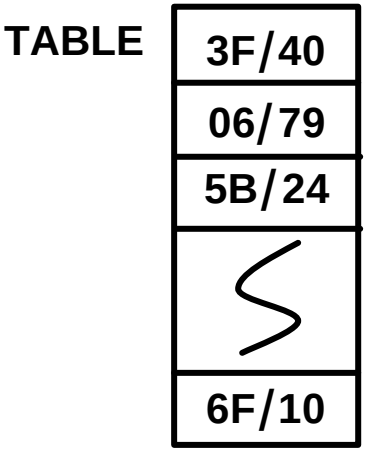




查七段显示码表：

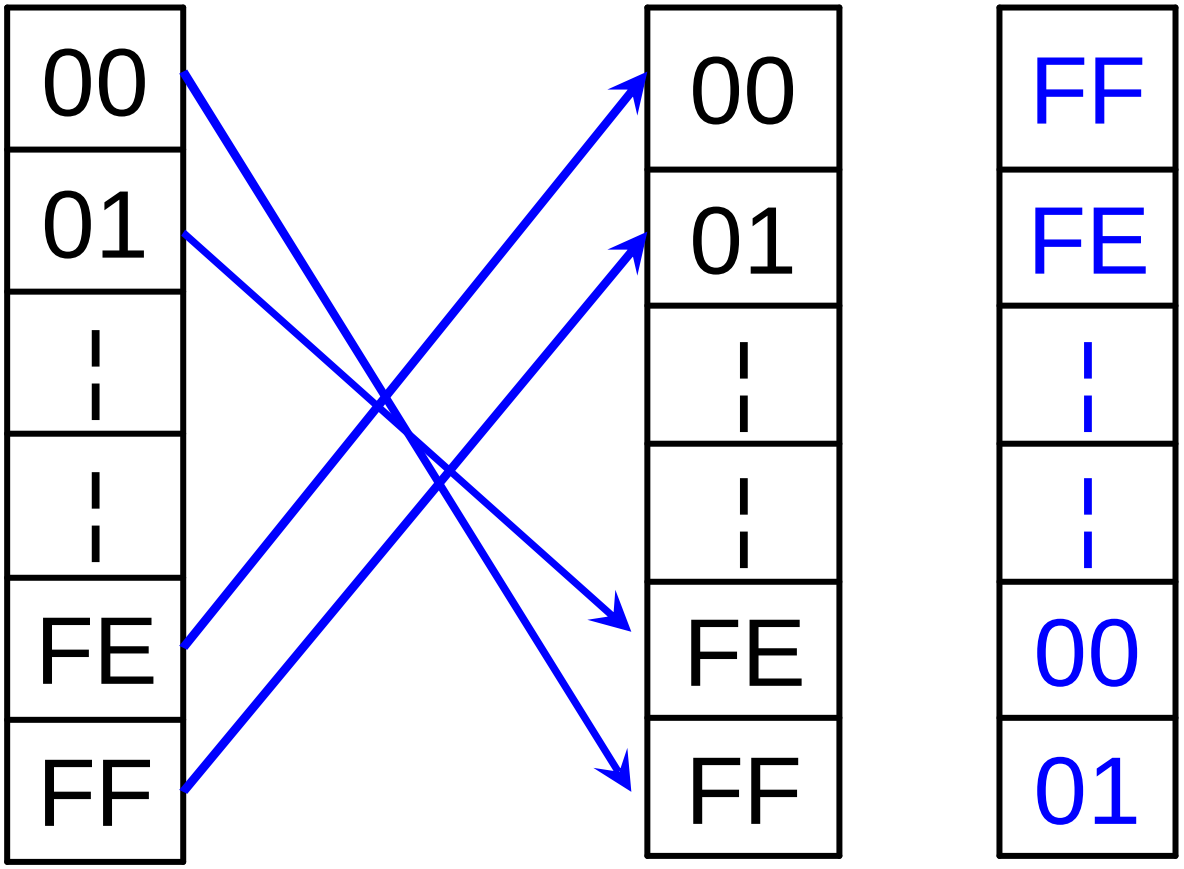
g f e d c b a(共阴极)	g f e d c b a(共阳极)
0 1 1 1 1 1 1→0<**3F**H>	1 0 0 0 0 0 0→0<**40**H>
0 0 0 0 1 1 0→1<**06**H>	1 1 1 1 0 0 1→1<**79**H>
1 0 1 1 0 1 1→2<**5B**H>	0 1 0 0 1 0 0→2<**24**H>
⋮ ⋮	⋮ ⋮
1 1 0 1 1 1 1→9<**6F**H>	0 0 1 0 0 0 0→9<**10**H>

```
MOV  BX，TABLE   ;TABLE为表首地址
MOV  AL，××H    ;××H为要转换的十进数
XLAT             ;执行后，AL←七段显示码
MOV  DX，2E0H    ;给DX赋端口地址
OUT  DX，AL      ;送端口显示
```



即可以通过“软件”的方法来实现“硬件”的功能。

此指令还可实现对信息的“加密”和“解密”功能。



3. 地址——目标传送指令

8086的地址——目标传送指令是用来对寻址机构进行控制的指令,这类指令传送到16位目标寄存器中的是存储器操作数的地址,而不是它的内容。这类指令有以下3条。

(1)有效地址送寄存器指令(Load Effective Address)

格式： LEA dst，src ；dst ¬ src的EA

其中,src必须是一个存储器操作数,dst是16位通用R,该指令常用来设置一个16位的寄存器R作为地址指针,不影响标志寄存器F。

功能：把src的16位偏移地址(有效地址)送到16位通用R中。

LEA BX，BUFR ;取BUFR的EA® BX，
与指令 MOV BX，OFFSET BUFR 等价。

LEA BX，[SI] ;将DS段偏移地址为SI的操作数的偏移地址→BX执行后，BX＝SI。

LEA BX，[BP+SI] ;将SS段偏移地址为BP+SI的操作数的偏移地址→BX，执行后，BX＝BP+SI的值。

LEA SP，[2000H]
;执行后，使堆栈指针SP＝2000H。

(2)指针送寄存器和DS的指令

格式： LDS dst，src

;dst←src的(EA),DS←src的(EA+2)

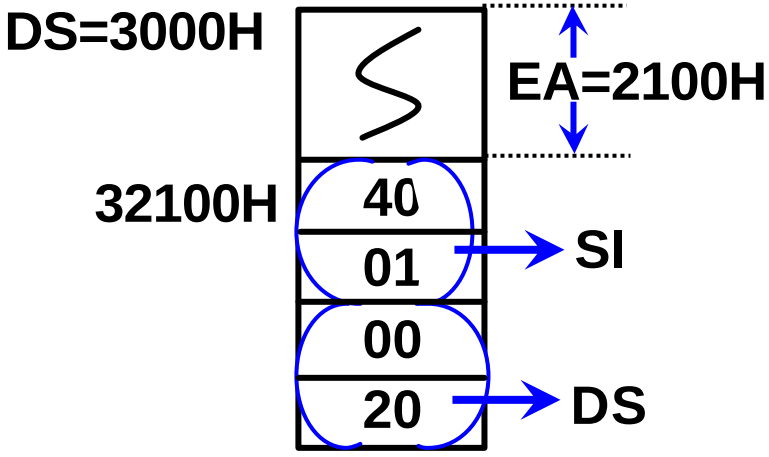
该指令完成一个32位地址指针的传送,地址指针包括段地址和偏移量两部分。指令把src所指内存的连续4字节内容(地址指针)传送到两个目标寄存器,其中:低字®dst所指通用R,高字®DS。该指令常指定SI作通用R。不影响标志寄存器F。

LDS SI，[2100H]

在指令执行前，设DS＝3000H，在DS段中，有效地址EA为2100H～2103H的4个字节，其中存放着一个地址指针，如图所示。则指令执行后：

SI＝0140H

DS＝2000H



(3)指针送寄存器和ES的指令

格式：LES dst，src ;dst ← src的(EA)

;ES ← src的(EA+2)

该指令和LDS dst，src功能类似,不同的只是用ES代替DS,这时常指定DI作寄存器R。不影响标志寄存器F。

使用LDS和LES指令时应注意：

① dst不能使用段寄存器;

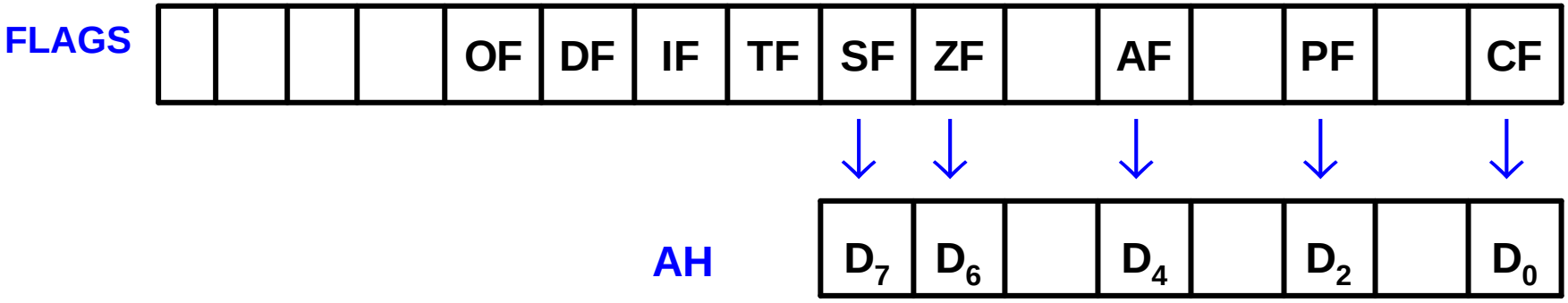
② src一定是存储器操作数,其寻址方式可以是24种当中的一种。

4. 标志传送指令

(1)读取标志指令(Load AH from Flags)

格式：LAHF ；AH←F_L，标志寄存器低8位送AH

指令执行后，将8086的16位标志寄存器的低8位取到AH中。这低8位包含了8位微处理器8080/8085所具有的SF、ZF、AF、PF和CF五个状态标志。



(2)设置标志指令(Store AH into Flags)

格式: SAHF ;F_L ¬ AH,AH的内容送F的低8位

SAHF指令和LAHF指令的操作正好相反,它是将AH寄存器的内容传送到标志寄存器的低8位,以对状态标志SF、ZF、AF、PF和CF进行设置。SAHF指令将直接影响标志位。

LAHF和SAHF两条指令只对低8位的标志寄存器操作,这样就保持了8086指令系统对8位8080/8085指令系统的兼容性。

二、算术运算(Arithmetic)类指令及应用

8086的算术运算类指令包括加、减、乘、除四种基本运算指令,以及为适应进行BCD码十进制数运算而设置的各种调整指令共20条。

操作数从数据形式来讲有两种,即8位和16位的操作数;从类型来讲也分两类,即无符号数和带符号数。

对无符号数和带符号数如何进行加、减、乘、除运算?是否可以采用相同的指令进行?对加法和减法可以采用同一套指令,而对乘法和除法则不能采用同一套指令。

用同一套指令是有条件的,这种条件有两个:

其一是两个操作数——被加数、加数或被减数、减数必须同为无符号数或同为带符号数;

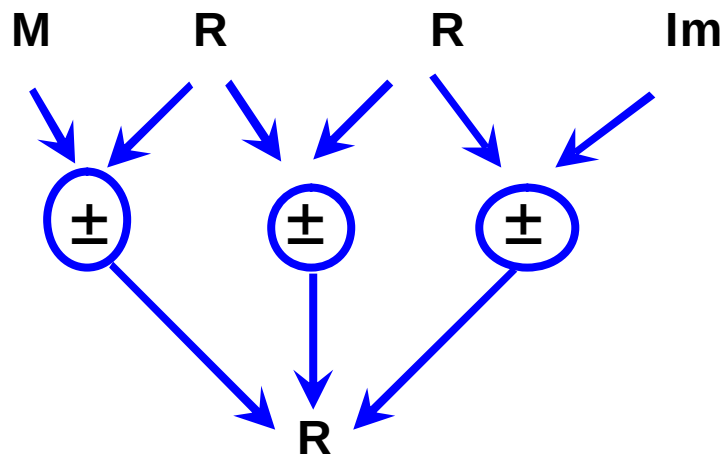
其二是检测无符号数或带符号数的运算结果是否发生溢出时,要用不同的状态标志。

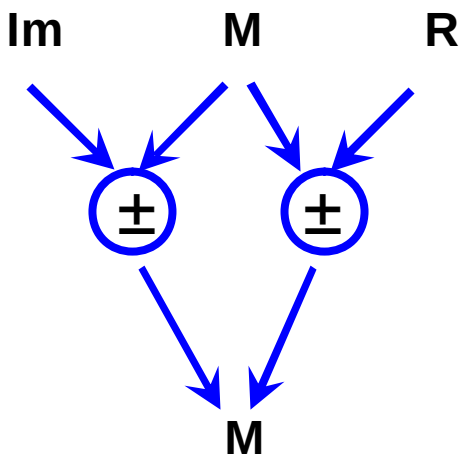
算术运算指令的特点有：

① 在加、减、乘、除基本运算指令中，除±1指令外，都具有两个操作数；

② 这类指令执行后，除±1指令不影响CF标志外，对CF、OF、ZF、SF、PF和AF等6位标志均可产生影响，由这6位状态标志反映的操作结果的性质如下：

- 当无符号数运算产生溢出时,CF＝1;
- 当带符号数运算产生溢出时,OF＝1;
- 当运算结果为零,ZF＝1;
- 当运算结果为负,SF＝1;
- 当运算结果中有偶数个1,PF＝1;
- 当操作数为BCD码,半字节间出现进位,需要进行BCD码调整,AF＝1。





1. 加法类指令(ADD)

加法类指令共有5条,其中3条为基本加法指令,2条为加法的十进制调整指令。

(1)不带进位加法指令

西南交通大学 Southwest Jiaotong University

ADD dst，src ；dst ¬ dst + src

ADD指令用来对源操作数src和目的操作数dst的字或字节数进行相加,结果放在目的操作数中,指令执行后对各状态标志OF、SF、ZF、AF、PF和CF均可产生影响。
dst和src还可使用多种寻址方式,如：

ADD AL，30 ;AL← AL+30 **,为R+Im→R(8位)**

ADD BX,3FFH ;BX←BX+3FFH,**为R+Im→R(16位)**

ADD AL,CL ;AL← AL+CL **,为R+R®R(8位)**

```
ADD DI,SI              ;DI ¬ DI+SI,为R+R® R(16位)
ADD AL,DATA[BX]        ;AL ¬ AL+(BX+DATA)
ADD  DX,DATA[BX+SI]
                       ;DX ¬ DX+(BX+SI+DATA)

ADD  BETA[SI],100      ;M+Im® M
ADD  [BX+2000H],AX     ;M+R® M
ADD  [BP],3AH          ;M+Im® M
```

(2)带进位的加法指令

ADC dst，src ；dst ¬ dst + src + CF

ADC指令和ADD指令功能基本类似,但区别在于ADC中进位标志的原状态一起参加运算,待运算结束,CF将重新根据结果置成新的状态。因此,ADC指令将用在两个多倍精度数的非最低字或非最低字节的相加。

例如：**ADC AX，DX**

执行的运算为：AX ¬ AX + DX + CF。

注意:第一次用带进位加法时,应将CF清零。

多字节的加法?

(3)加1指令 INC src

指令只有一个操作数src。src可为R/M,但不能为立即数,该指令将src操作数当成无符号数,完成加1操作,所以又叫增量指令。它常用在循环结构修改指针或用作循环计数,例如：

```
INC  CX  ;将CX的内容加1后再送回CX中。
INC  SI   ;将SI的内容加1后再送回SI中。
INC  BYTE PTR[BX+100H]
         ;将BX+100H所指的单元内容加1后,送回。
```

注意：INC指令只影响OF、SF、ZF和PF,而不影响CF。

(4)压缩BCD码加法调整指令　DAA

指令用于对压缩BCD(又称组合BCD)码相加的结果进行调整，使结果仍为压缩BCD码。

微处理器中,运算器的核心是二进制加法器,逢二进一,当BCD码≤9时,遵循逢二进一；而＞9时,遵循逢十进一。因此,二进制数的1010、1011、1100、1101、1110和1111对BCD数都是非法码,必须进行调整。DAA指令应紧跟在加法指令之后,执行时,先对相加结果进行测试,若结果的低4位(或高4位)二进制数大于9(非法码)或大于15(即产生进位CF或辅助进位AF)时,DAA自动对低4位(或高4位)进行加6的调整。调整在AL中进行,因此加法运算后,必须把结果放在AL中。DAA指令执行后,将影响除OF之外的其它标志。

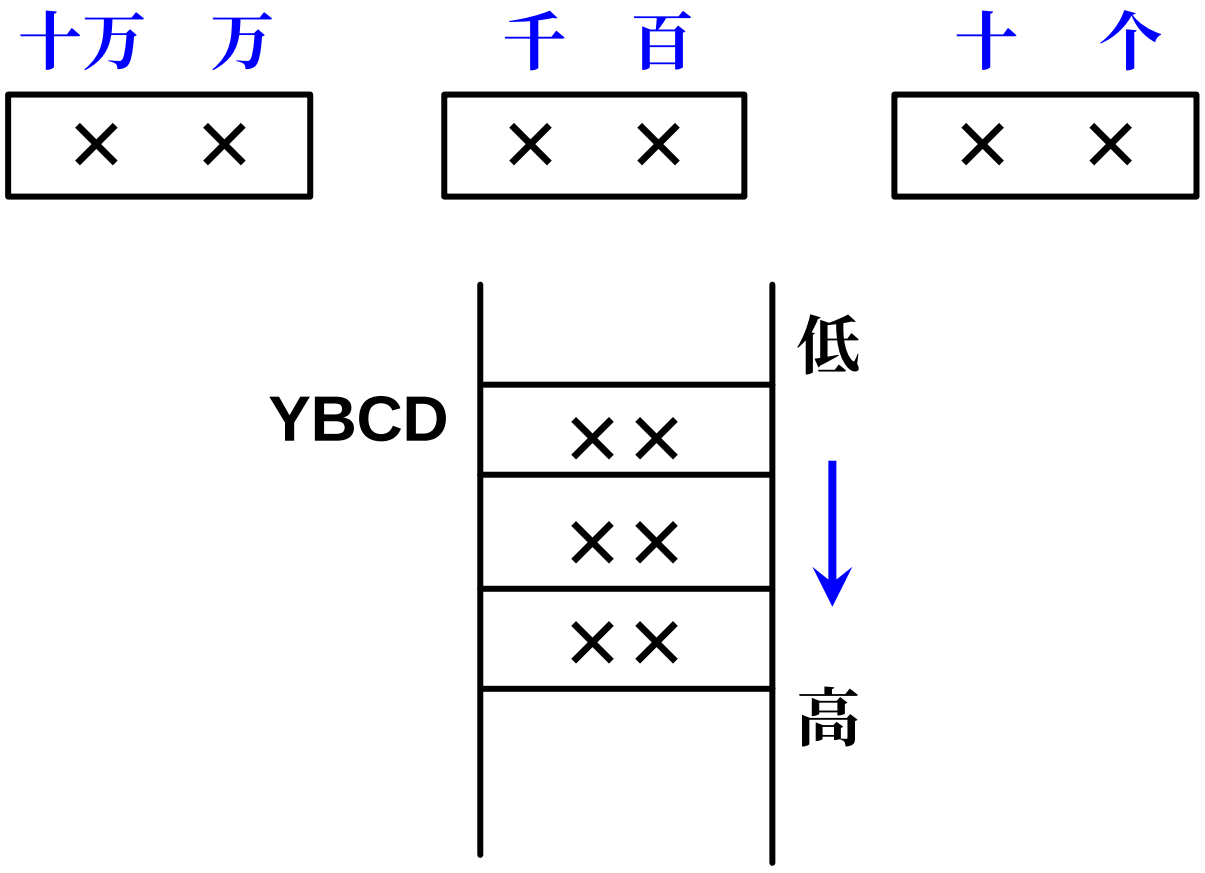
```
MOV  BL，08H          14      14H
MOV  AL，14H        +  8    +  08H
ADD  AL，BL           22      1CH
DAA                         +   6
                                22H
```

而 { MOV AL，1CH
  { DAA　　则是错误的。

注意：压缩BCD码每个字节表示两位十进制数，只能进行字节运算，加法可以不在AL中进行，但结果一定要放在AL中，DAA指令紧跟在加法指令之后，调整只能在AL中进行。

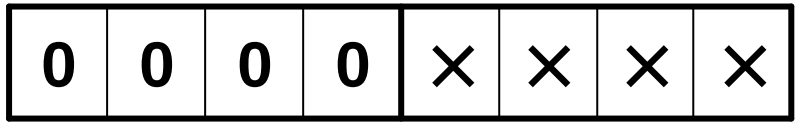
加6调整，可能加06H、60H或66H。

[例] 对多个字节表示的压缩BCD数进行增1。

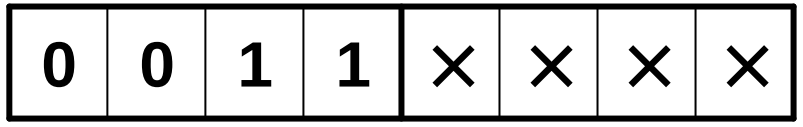


```
MOV   AL，YBCD          ；取个位、十位
ADD   AL，1             ；加1
DAA                     ；调整
MOV   YBCD，AL          ；存个位、十位
MOV   AL，YBCD+1        ；取百位、千位
ADC   AL，0             ；加低位的进位
DAA                     ；调整
MOV   YBCD+1，AL        ；存百位、千位
MOV   AL，YBCD+2        ；取万位、十万位
ADC    AL，0            ；加低位的进位
DAA                     ；调整
MOV   YBCD+2，AL        ；存万位、十万位
```

(5)非压缩BCD码加法调整指令　AAA



非压缩BCD码



ASCII码表示的数

非压缩BCD(又称非组合BCD)码,每个字节表示一位十进制数。

AAA指令用于对非压缩BCD码的相加结果进行调整,调整操作仍在AL中进行,调整后的结果在AX中。AAA指令的操作如下：

若AL＆0FH＞9或AF＝1,则

AL ¬ AL + 6

AH ¬ AH + 1

AL ¬ AL & 0FH

指令执行后,除影响AF和CF标志外,其余标志均无定义。

数字0～9的ASCII码是一种非压缩BCD码,因为其高4位为0011,而低4位才是以8421码表示的十进制数,这符合非压缩BCD码高4位无意义(定为0)的规定,这也是AAA指令又称为ASCII码加法调整指令的原因。

[例] 将两个具有16位的BCD数相加，被加数和加数分别放在从FIRST和SECOND开始的存储单元中，结果放在THIRD开始的单元。

采用循环结构，每循环一遍，对一个字节的BCD相加，相加之后，立即进行DAA调整，共循环8次，且采用ADC带进位加。当进行最低字节相加时，可预先设置CF为0(用CLC清进位)。程序段如下：

```
MOV  BX，OFFSET FIRST        ;指向被加数
MOV  SI，OFFSET SECOND       ;指向加数
MOV  DI，OFFSET THIRD        ;指向和
MOV  CX，8                   ;设置计数初值
CLC                          ;清进位标志
```

```
AGN:    MOV  AL，[BX]        ;加一字节
        ADC  AL，[SI]
        DAA
        MOV  [DI]，AL        ;存结果
        INC  BX
        INC  SI              ;修改指针
        INC  DI
        DEC  CX              ;计数
        JNZ  AGN
          ⋮                  ;结束
```

2. 减法类指令(Subtract)

减法指令共有7条,其中5条为基本减法指令,2条为十进制减法调整指令。

(1)不带借位减法指令

SUB dst，src ；dst←dst－src

SUB指令用来完成2个字/字节的相减,结果在目的操作数中,并根据结果影响标志位。

SUB AX，BX

(2)带借位的减法指令

SBB dst，src ；dst←dst－src－CF

SBB指令和SUB功能基本类似,但区别在于SBB在完成2个字或2个字节数相减的同时,还要减去借位CF的原状态,运算结束时,CF将被置成新状态。因此,SBB指令将用在两个多倍精数的非最低字或非最低字节的相减。

```
SBB  AX，2010H
        ;执行的运算为：AX ← AX－2010H－CF。
```

注意:第一次用带借位减法时,应将CF清零。

(3)减1指令　DEC　src　　;src ← src－1

减1指令只有1个操作数src,和INC指令类似,src可为R/M,不能为立即数,该指令实现对src中的内容减1,又叫减量指令。它常用在循环结构程序中修改指针(反向移动指针)和用作循环计数。例如：

```
DEC  CX       ;CX内容减1后,送回CX;
DEC  BYTE PTR [DI+2]
              ;将(DI+2)所指字节单元内容减1后,送回。
```

注意：DEC指令和INC指令一样,执行后对CF不产生影响。

(4)求变补指令

NEG src ;src ← 0 – src

;或 src ← $\overline{src}$+1

NEG指令将src中的内容求变补后,再送回src中。求变补相当于src ← 0 – src,所以NEG指令执行的操作也是减法操作。0 – src又相当于:$\overline{src}$+1,即将src的每一位变反,再在最低位加1。src可为8/16位R/M操作数。

在大多数情况下：NEG指令将src当成一个带符号数,若原来为“+”,则NEG后变成绝对值相等的负数(用补码表示);若原来为负数(用补码表示),则NEG后变成绝对值相等的正数。例如：

若AL＝00000100B＝+4，

NEG AL后,变为AL＝11111100B＝$[-4]_{补}$;

AL＝11101110B＝$[-18]_{补}$，

NEG AL后,变为AL＝00010010B＝+18。

在特殊情况下:若AL＝80H(即–128),则NEG　AL后,AL仍为80H;

若AX＝8000H(即–32768),则NEG　AX后,AX仍为8000H(即–32768)。

用途:在求Im–R（或M）时,用SUB Im,AL是错误的,但可用下面的指令组来实现:

“变补相加”

```
NEG  AL
ADD  AL，Im
```

- 使用NEG指令时应注意：NEG指令执行后，对OF、SF、ZF、AF、PF和CF均产生影响。但是对CF的影响总是使CF为1，这是因为0减某操作数，必然产生借位，只有当操作数为0时，CF才为0。

(5)比较指令

CMP dst，src

;dst－src,不回送结果,只影响标志位。

CMP指令和SUB指令类似,也是执行两操作数相减,但和SUB指令不同的是,不送回相减结果,只是使结果影响标志位OF、SF、ZF、PF和CF。换句括说,由受影响的标志位状态便可判断出操作数比较的结果。

src可以为8/16位R/M或Im,dst只可为8/16位R/M ,并且src与dst不能同时为M。

例如： CMP AX，BX

CMP AL，100

CMP CX，COUNT[BP]

CMP POINTER[DI]，BX

⋮

如何利用上述比较操作的标志判断两操作数的关系呢?两个数的比较首先要区分是无符号数还是带符号数。

两个无符号数的比较,如 CMP AX，BX

若ZF＝1,表明AX＝BX；

CF＝0,无借位,则AX≥BX；

若ZF＝0,表明AX≠BX

CF＝1,有借位,则AX＜BX。

两个带符号数的比较(以补码形式出现),仍以CMP AX，BX 为例:

若AX和BX中数同符号,即AX＞0、BX＞0或AX＜0、BX＜0,则AX－BX不会产生溢出,同时可根据SF标志判断两个数的大小：SF＝0,则AX≥BX; SF＝1,则AX＜BX。

若AX和BX中数不同符号,即AX＞0、BX＜0或AX＜0、BX＞0,那么AX－BX则可能产生溢出:

如果AX－BX没有产生溢出,则仍可由SF判断两个数的大小：SF＝0,则AX≥BX; SF＝1,则AX＜BX。

如果AX－BX产生溢出,则：当SF＝1时,AX≥BX; 当SF＝0时,AX＜BX。

可得出如下结论：

$$\begin{cases} OF \oplus SF = 0 & AX \geq BX \\ OF \oplus SF = 1 & AX < BX \end{cases}$$

综上所述,判断两个带符号数的大小,应由符号标志SF和溢出标志OF综合进行判断。

CMP指令对各状态标志的影响,为以后的条件转移指令创造条件。

两个数比较以后,对两个数本身没有任何影响,仅影响标志位；两个数比较时一定要分清是无符号数还是带符号数,两者判断的条件不同,对应的条件转移指令也不同。

[例]指令SUB　AL,BL执行后,AL＝83H,OF＝1,问AL、BL原来是正数还是负数?

解: 从OF＝1,知AL－BL有溢出,只有两个异号的数相减才会溢出，故：

AL＞0、BL＜0或AL＜0、BL＞0

又因指令SUB　AL，BL执行后,AL＝83H,结果为负,SF＝1,

∴　OF⊕SF＝0

从而有：AL≥BL

所以有：AL＞0、BL＜0,即AL原来是正数,BL原来是负数。

(6)压缩BCD码减法调整指令　DAS

DAS指令用于对压缩BCD码相减结果进行调整,紧跟在减法指令之后,调整后的结果仍为压缩BCD码。调整也只能在AL中进行。DAS与DAA对加法结果进行调整的作用相似,不同的是DAS对结果是进行减6调整。该指令执行后,对AF、CF、PF、SF和ZF均产生影响,但OF没有意义。

(7)非压缩BCD码减法调整指令　AAS

AAS指令用于对非压缩BCD码相减结果进行调整，也是紧跟在减法指令之后，调整后的结果仍为非压缩BCD码。AAS对减法结果的调整和AAA对加法结果的调整作用相似，但具体操作有两点不同：

一、AAA指令中的AL ← AL + 6的操作对应AAS中则应改为AL ← AL – 6；

二、AAA指令中的AH ← AH + 1的操作对应AAS则应改为AH ← AH – 1。

AAS指令执行后，只影响AF、CF标志，而OF、PF、SF和ZF都没有意义。

3. 乘法类指令(Multiplication)

乘法类指令共有3条,其中2条为基本乘法指令,包括对无符号数和带符号数相乘的指令,还有1条是非压缩BCD相乘调整指令。

进行乘法时,如果两个8位数相乘,乘积将是一个16位的数;如果两个16位数相乘,则乘积将是一个32位的数。

乘法指令中有两个操作数,但其中一个是隐含固定在AL或AX中,若是8×8,被乘数总是先放入AL中,所得乘积在AX中;若是16×16,被乘数总是先放入AX中,乘积在DX和AX两个16位的寄存器中,且DX中为乘积的高16位,AX中为乘积的低16位。

(1)无符号数的乘法指令

MUL src

MUL指令中的乘数src可以是R/M中的8位或16位的无符号数,不可为立即数Im,被乘数固定放在AL或AX中,也是无符号数。

若运算结果AH≠0(对应8×8)/ DX≠0(对应16×16),则CF和OF两标志同时置1,而不影响其它标志位(其它标志不确定,无意义)。

MUL CL ;AX ← AL*CL；

MUL CX ;DX,AX ← AX*CX,AX和CX中的16位数相乘,乘积在DX和AX中;

MUL WORD PTR [SI] ;AX和SI所指的字单元中的16位数相乘,结果在DX和AX中。

[例] MOV AX，5612H

MOV BL，66H

MUL BL ;积 AX＝072CH

因AH≠0,故CF＝1,OF＝1,而其它标志位不受影响(不确定,无意义)。

注意：对于8×8,无论AH原来的值是多少,都不参与运算,结束后,AH将被积的高8位所替代。

无符号数的乘法只有8×8和16×16两种,如何实现16×8呢?

很简单,向高的标准看齐,将16×8变成16×16,将8位的无符号数变成16位的无符号数(高8位补0)即可。

(2)带符号数的乘法指令 IMUL src

IMUL指令和MUL指令在功能和格式上类似,只有8×8和16×16两种,要求两个乘数均必须为带符号数。例如：

IMUL CL ;AL中和CL中的8位带符号数相乘,结果在AX中；

IMUL AX ;AX中和AX中的16位带符号数自乘,结果在DX和AX中；

IMUL BYTE PTR[BX] ;AL中和BX所指的字节单元中的8位带符号数相乘,积在AX中;

IMUL WORD PTR[DI] ;AX中和DI所指的字单元中的16位带符号数相乘,结果在DX、AX。

执行IMUL指令后,如果乘积的高半部分不是低半部分的符号扩展(不是全零或全1),则视高位部分为有效位,表示它是积的一部分,于是置CF＝1,OF＝1。若结果的高半部分为全零或全1,表明它仅包含了符号位,那么使CF＝0,OF＝0。利用这两个标志状态可决定是否需要保存积的高位字节或高位字。IMUL指令执行后,AF、PF、SF和ZF不确定(无意义)。

如何实现带符号数的16×8呢?请读者自己思考。

(3)非压缩BCD码乘法调整指令　AAM

先作乘法，后调整。对十进制数进行乘法运算,要求乘数和被乘数都是非压缩的BCD码。

调整过程为:把AL寄存器内容除以10,商放在AH中,余数在AL中,AH ← AL/10所得的商

AL ← AL/10所得的余数

指令执行后,将影响ZF、SF和PF,但AF、CF和OF无定义。

两个ASCII码数相乘之前,必须先屏蔽掉每个数字的高半字节。

因BCD码总是被当作无符号数看待,所以相乘是用MUL指令,而不是IMUL。

[例]求两个非压缩BCD码09和08之乘积。

可用如下指令实现：

```
MOV  AL，09H        ;赋初值
MOV  BL，08H
MUL  BL                    ;AL ¬ 09与08之乘积48H
AAM     ;调整得AH=07H(十位),AL=02H(个位)
```

最后可在AX中得到正确结果AX＝0702H,即十进制数72。

如何实现压缩BCD码的乘法呢?可采用变通办法,变压缩BCD码乘法为压缩BCD码加法。

4. 除法类指令(Division)

8086执行除法运算时,规定被除数必为除数的双倍字长,即除数为8位时,被除数应为16位,而除数为16位时,被除数为32位。

除法指令有两个操作数,其中被除数隐含固定在AX中(除数为8位时)或DX,AX中(除数为16位时)。在使用除法指令前,需将被除数用MOV指令传送到位。

(1)无符号数除法指令　DIV　src

对两个无符号数进行除法操作。src可以是字节或字，它可以是R/M,但不可为Im。

16÷8,被除数在AX中,操作后,AL ¬ 商,AH ¬ 余数

32÷16,被除数在DX,AX中,操作后,AX ¬ 商,

DX ¬ 余数

若除法运算所得的商超出AL/AX的容量,则系统将其当做除数为0处理,自动产生类型0中断,此时所得的商和余数均无效。

除法运算后,AF、ZF、OF、SF、PF和CF都是不确定的(无意义)。

DIV CX ;32÷16,被除数(在DX,AX中)/CX,所得商在AX中,余数在DX中。

DIV CL ;16÷8,实现AX/CL,所得商在AL中,余数在AH中。

DIV WORD PTR[DI] ;实现DX和AX中的32位数被除数与DI和DI+1所指的两个单元中的16位数相除,商在AX中,余数在DX中。

无 8÷8、16÷16指令。

特别注意:有些无符号数相除,被除数16位,除数8位,看起来很符合16÷8,但使用时往往要将其变成32÷16,究其原因是除数太小,商一个字节放不下。

(2)带符号数的除法指令，IDIV src

IDIV指令用于两个带符号数相除,其功能和对操作数长度的要求和DIV指令类似,例如:

IDIV CX ;带符号数32÷16,将DX,AX的32位数除以CX的16位数,商在AX中,余数在DX中

IDIV BYTE PTR[SI] ;带符号数16÷8,将AX中的16位数除以SI所指单元中8位数,商在AL中,余数在AH中。

指令执行后,所有标志都不确定(无意义)。

带符号数除法也只有16÷8和32÷16两种,要实现带符号数的32÷8、16÷16及8÷8,就必须对操作数长度进行扩展。

(3)扩展操作数长度指令

CBW ;将AL中的符号位扩展到AH中,从而使AL中的8位带符号数扩展到AX中的16位带符号数。当AL＜80H(为**正数**),执行CBW后,AH＝0;而当AL≥80H(为**负数**),执行CBW后,AH＝0FFH。

CWD ;将AX中的符号位扩展到DX中,从而得到DX,AX组成的32位双字带符号数。当AX＜8000H(为**正数**),执行CWD后,DX＝0;而当AL≥8000H(为**负数**),执行CWD后,DX＝0FFFFH。

此指令应安排在IDIV指令之前,且此两条指令执行后不影响标志位。

注意:CBW和CWD均只能用在带符号数扩展,而不能用作无符号数的扩展。而且扩展只能是AL到AX或AX到DX、AX,其它寄存器均不行。

无符号数的扩展:

AL中8位扩展到AX中16位,

用指令　MOV AH，0

AX中16位扩展到DX、AX中32位,

用指令　MOV DX，0

(4)非压缩BCD除法调整指令　AAD

先调整，后作除法。要求除数、被除数都用非压缩BCD码 。

在把AX中的两位非压缩格式的BCD数除以一个非压缩BCD数前,要先用AAD指令把AX中的被除数调整成二进制数,并保存到AL中,才能用DIV指令进行运算,以使除法得到的商和余数亦为非压缩BCD码。

AAD的操作：　AL ← AH×10 + AL

AH ← 00

无压缩BCD除法调整指令,如何实现压缩BCD码的除法呢?可以采用变通办法,变压缩BCD除法为压缩BCD减法 。

三、逻辑运算和移位(Logic & Shift)循环指令

1. 逻辑运算指令

		OF	SF	ZF	AF	PF	CF
AND dst,src	;dst←dst∧src	0	×	×	u	×	0
OR dst,src	;dst←dst∨src	0	×	×	u	×	0
XOR dst,src	;dst←dst⊕src	0	×	×	u	×	0
NOT src	;src←$\overline{src}$	-	-	-	-	-	-
TEST dst,src	;dst∧src	0	×	×	u	×	0

对F的影响说明:×为影响,u为无意义,- 为不影响。

src(除NOT外)可为R/M或Im,dst只能为R/M。因为NOT指令的src既是src,又是dst,故只能为R/M。

(1)AND逻辑“与”指令

指令格式：AND dst，src ；dst ← dst∧src
指令功能：对两个操作数进行按位逻辑“与”操作，结果回送目的操作数，要影响标志。

AND指令主要用于：使操作数的若干位维持不变，另外的位清0，维持不变的位跟“1”相“与”，而清0的位跟“0”相“与”。

从对标志的影响可知,AND操作使CF清0,故若要清CF,可用下面的指令：AND AX，AX;
也可用 AND AX，0FFFFH 来实现。

指令AND AX，AX 还有一个另外的作用:它对AX内容无影响,又可以用来判断AX是否为0,因为AND指令执行后要影响ZF。指令AND AX，AX还可用来检查数据的符号、奇偶性等。

(2)OR逻辑“或”指令

指令格式：OR dst，src ；dst ← dst∨src

指令功能:对两个操作数进行按位逻辑“或”操作，结果回送目的操作数,要影响标志。

OR指令主要用于:使操作数的若干位维持不变,另外的位置1,维持不变的位跟“0”相“或”,而置1的位跟“1”相“或”。

从标志的影响可知,OR操作都使CF清0,故若要清CF,可用下面的指令:

OR AX，AX

指令OR AX，AX 还有一个另外的作用:它对AX内容无影响,又可以用来判断AX是否为0,因为OR指令执行后要影响ZF。它还可用来检查数据的符号、奇偶性等。

(3)XOR “异或”操作指令

指令格式：XOR dst，src ；dst ← dst⊕src

指令功能:对两个操作数进行按位逻辑“异或”运算，结果回送目的操作数,要影响标志。

XOR指令主要用于：使操作数的若干位维持不变,另外的位取反,维持不变的位跟“0”相“异或”,而取反的位跟“1”相“异或”。

指令XOR　AX，AX 的一个重要作用：常用来使AX清0，此操作同时使CF清0。

逻辑运算(A⊕B)⊕B＝A，可用于信息的“加密”和“解密”。

B为密码，A⊕B相当于“加密”，将加密所得的结果再与密码B进行“异或”运算就是“解密”；反之，亦然。

(4)NOT取反指令

指令格式： NOT src ; src ¬ $\overline{src}$

指令功能：将源操作数取反,结果回送源操作数,不影响标志。

src可为8/16位的R/M,对于M,要说明其类型是字节还是字。

(5)TEST测试指令

指令格式：TEST dst，src ；dst∧src

指令功能：TEST指令和AND指令执行同样的操作，对两个操作数进行逻辑“与”操作，但TEST指令不回送操作结果，仅仅影响标志位，指令执行后，两个操作数都不变。

TEST指令一般用来测试指定位是1还是0。

TEST AL，80H

TEST [BX]，01H

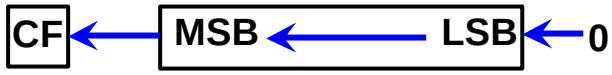
2. 移位类指令

指令格式：

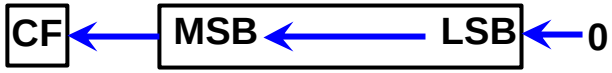
```
SAL  src，count   ;算术左移
SAR  src，count   ;算术右移
SHL  src，count   ;逻辑左移
SHR  src，count   ;逻辑右移
```

src可为R/M,操作数长度可为8/16位,count为移位次数,移位1次的,在指令中直接给出；移位n次的,则需预先将n送入CL寄存器中,这样,对8086的移位指令,移位次数的范围可以是1～255。

SAL src，count

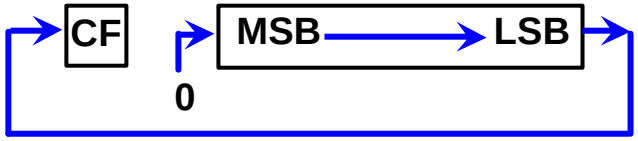


SHL src，count



SAR src，count

SHR src，count



逻辑移位指令用于无符号数的移位,左移时,最低位补0;右移时,最高位补0。算术移位指令用于对带符号数的移位,左移时,最低位补0;右移时,最高位参加移位,但保持不变。CF中总保留最后移出的一位的状态。

SHL和SAL的功能完全一样,因为对一个无符号数乘以2和对一个带符号数乘以2没有什么区别,每移一次,最低位补0,最高位移入CF。

在左移位数为1的情况下,移位结果使最高位(符号位)发生变化,则溢出标志OF＝1,这对于带符号数便可由此判断移位前后的符号位不同,反之,若移位后的最高位和CF相同,则OF＝0,这表示移位前后符号位没有改变。若移多位,则OF标志无效。

SHR和SAR的功能不同。SHR执行时最高位补0,因为它是对无符号数移位,而SAR执行时最高位保持不变,因为它是对带符号数移位,应保持符号不变。

左移1位,在无溢出的前提下,实现乘2运算(无论带符号数还是无符号数)；逻辑右移1位实现无符号数的除2运算；算术右移1位实现带符号数的除2运算，其余则依此类推。用左、右移位指令实现乘/除运算,比用乘、除指令实现所需的时间要短得多。

[例] 用移位指令实现A×10＝A×2＋A×8,A的长度为一个字节。

当A为无符号数时,则A×10的范围为0～2550,一个字节放不下。实现的程序如下：

```
MOV     AL，××H  ;给AL赋A的值
MOV     AH，00H   ;将A变成16位无符号数
SHL     AX，1     ; A×2
MOV     BX，AX
SHL     AX，1
SHL     AX，1     ; A×8
ADD     AX，BX    ; A×10结果在AX中
```

当A为带符号数时,则A×10的范围为-1280～+1270，一个字节放不下。实现的程序如下：

```
MOV     AL，××H ;给AL赋A的值
CBW              ;将A变成16位带符号数
SAL     AX，1    ;A×2
MOV     BX，AX
SAL     AX，1
SAL     AX，1    ;A×8
ADD     AX，BX   ;A×10结果在AX中
```

指令 SAR AL，1 相当于带符号数÷2;

指令 SHR AL，1 相当于无符号数÷2。

3. 循环移位指令

不带进位的循环左、右移指令(小循环) ROL、ROR和带进位的循环左、右移指令(大循环)RCL、RCR。

(1)ROL不带进位的左环移指令

指令格式： ROL src，count

(2)ROR不带进位的右环移指令

指令格式： ROR src，count

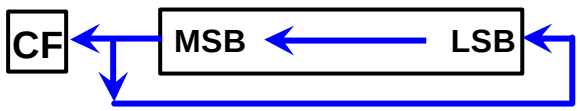
(3)RCL带进位的左环移指令

指令格式： RCL src，count

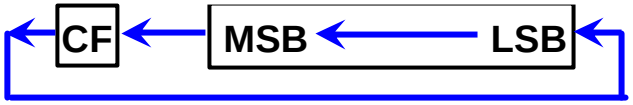
(4)RCR带进位的右环移指令

指令格式： RCR src，count

ROL src，count

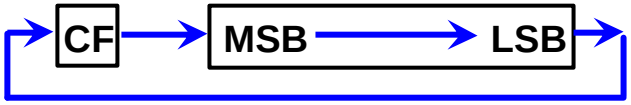


RCL src，count



ROR src，count

RCR src，count



循环移位指令,执行1次可移动1位,也可以执行1次移动n位，n仍然是预先放入CL寄存器中。

实现存于通用寄存器DX和AX的32位数,或存于连续四个存储单元中的32位数的联合左移:

```
SAL  AX，1
RCL  DX，1
```

寄存器DX和AX的32位带符号数整体右移一位:

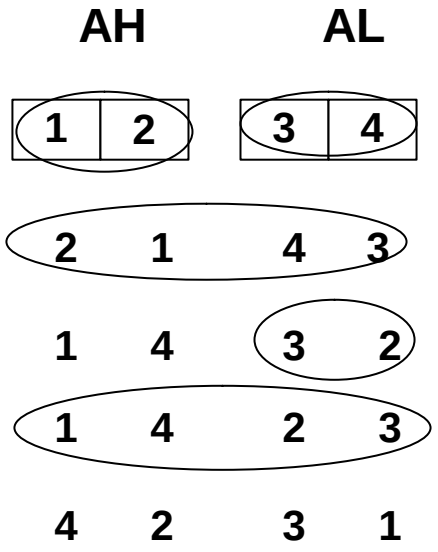
```
SAR  DX，1
RCR  AX，1
```

寄存器DX和AX的32位无符号数整体右移一位:

```
SHR  DX，1
RCR  AX，1
```

将AX中的最高4位与最低4位交换。

```
MOV  CL，4
ROL  AH，CL  ;变成2 1 3 4
ROL  AL，CL  ;变成2 1 4 3
ROL  AX，CL  ;变成1 4 3 2
ROL  AL，CL  ;变成1 4 2 3
ROL  AX，CL  ;变成4 2 3 1
```



四、信息串操作(String Manipulation)指令

它是8086唯一的一类源操作数和目的操作数都在存储单元的指令。

串操作指令执行时,须遵守以下的隐含约定：

① 地址：总是用DS:SI指向源串首址,源串允许使用段超越前缀来修改段地址；ES:DI指向目的串首址,但目的串不允许使用段超越前缀修改ES。如果要在同一段内进行串运算,必须使DS和ES指向同一段。

② **处理单位**：可以对字节串进行操作，也可以对字串进行操作。指令后面带有B，表示每次处理一个字节；带有W，表示每次处理一个字。

③ **处理方向**：串操作指令执行时，地址指针的修改与方向标志DF有关。DF＝0（用CLD指令设置），表示内存地址由低到高（递增），SI和DI自动增量修改；DF＝1（用STD指令设置），表示内存地址由高到低（递减），SI和DI进行自动减量修改。因此，在串操作指令执行前，需对SI、DI和DF进行设置。

④ **重复操作**：在串操作指令前加重复前缀(REP、REPE/REPZ、REPNE/REPNZ)时，可使串操作重复进行到结束，由CX表示重复次数。在每次重复之后，地址指针SI和DI分别为±1(字节操作)或±2(字操作)，但指令指针IP仍保持指向前缀的地址。

⑤ **处理的长度**：可达64 KB，将要处理的串长度(字节或字数)放在CX寄存器中，每处理一个单位，CX ¬ CX－1，但此CX减1并**不影响标志寄存器**F。

⑥ **重复前缀**:REP、REPE/REPZ、REPNE/REPNZ

REP	无条件重复
REPE/REPZ	相等/结果为零则重复
REPNE/REPNZ	不相等/结果非零则重复

❖无条件重复前缀指令REP常与串传送指令MOVS连用,连续进行字符串传送操作,直到整个信息串传送完毕,CX＝0为止；若CX的初值为0,则根本不进行重复操作。

- 重复前缀REPE和REPZ含义相同，它们常与串比较指令CMPS连用，连续进行字符串比较操作。当两个字符串相等ZF＝1和CX≠0时，则重复进行比较，直到ZF＝0或CX＝0为止。

- 重复前缀REPNE和REPNZ意义相同，它们常与串扫描指令SCAS连用，当结果非零ZF＝0和CX≠0时，重复进行扫描，直到ZF＝1或CX＝0为止。

- 串操作指令是一类高效率的操作指令，合理选用对程序的优化很有好处。

1. 信息串传送指令MOVS

指令格式： MOVS dst，src

简洁书写格式：

字节传送： MOVSB

字传送： MOVSW

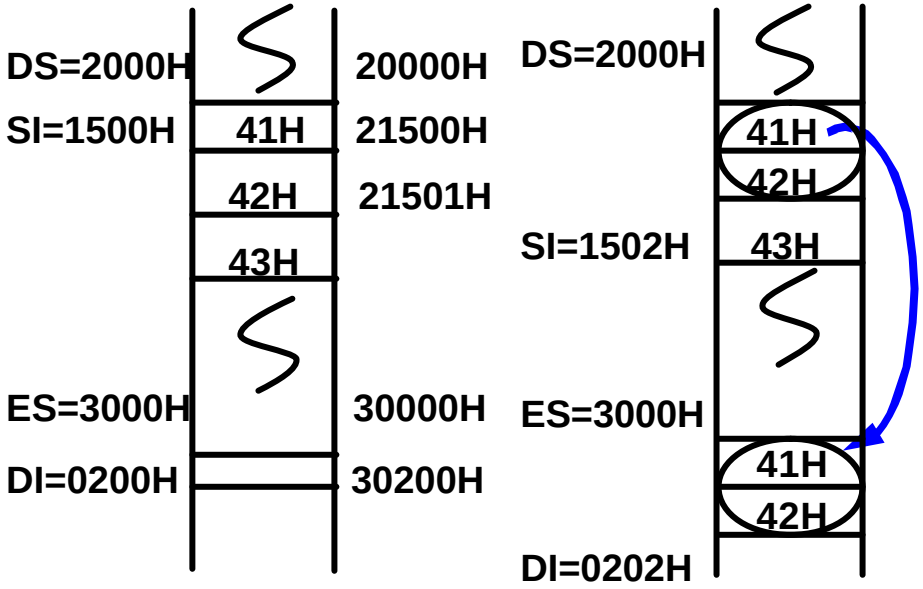
指令功能：把由DS:SI作指针的源串中的一个字节或字,传送到由ES:DI作指针的目的串中,且自动修改指针SI和DI。

MOVSB ;字节串传送,传完后SI和DI内容±1。

MOVSW ;字串传送,传送完后SI和DI内容±2。

将DS＝2000H、SI＝1500H中的一个字传送到ES＝3000H、DI＝0200H中。

```
CLD      ;DF＝0,由低到高
MOV  AX，2000H
MOV  DS，AX
MOV  AX，3000H
MOV  ES，AX
MOV  SI，1500H
MOV  DI，0200H
MOVSW
```



执行完后SI、DI指向下一个字节/字。

例如,传送3个字节：

```
STD                    ;DF＝1,由高到低
MOV  AX，2000H
MOV  DS，AX
MOV  AX，3000H
MOV  ES，AX
MOV  SI，1500H
MOV  DI，0200H
MOV  CX，3
REP   MOVSB
```

执行完后,SI、DI指向下一个字节,SI＝14FDH，DI＝01FDH。

MOVSB/MOVSW加上重复前缀或其它控制循环的指令,可用以实现信息块的搬家,若使用重复前缀,须先给CX赋值重复的次数。如果信息块的长度是多少个字节,用MOVSB传送时,则将此字节数赋给CX作重复次数;若用MOVSW传送时,则将此字节数÷2赋给CX作重复次数。

[例] 将100个字节从内存AREA1传送到AREA2。

```
MOV  SI,OFFSET AREA1  ;取AREA1的偏移地址
MOV  DI,OFFSET AREA2  ;取AREA2的偏移地址
MOV  CX,100           ;置重复次数
CLD                   ;设置传送方向
REP MOVSB             ;重复传送
```

若用指令REP MOVSW,则给CX赋的值应为50。

在内存中实现信息块的搬家,要特别注意信息的覆盖问题。也就是说源块与目的块有地址重叠。解决此问题的方法是正确地设置传送方向。如何正确地设置传送方向呢?可分以下几种情况:

① 同一段内,即DS＝ES,将SI和DI比较一下:

- 若SI＞DI,则应从低到高,设置DF＝0;
- 若SI＜DI,则应从高到低,设置DF＝1。

② 不同段内,DS≠ES,先将DS和ES比较一下,再做减法(大减小):

- 若DS和ES的差值≥1000H,则源块与目的块无任何地址重叠,传送方向DF可随意设置;

- 若DS和ES的差值＜1000H,则源块与目的块可能有地址重叠,要进行地址**折算**,转化成同一段内传送,转化的方法是大段基址向小段基址看齐,将段基址的不同转化同一段内地址的偏差,即DS和ES的差值乘以16再加上原来大段基址中的EA就得到新的同一段(小段基址)内的偏移地址,最后按同一段内传送处理。

  例如,将源串地址DS:SI为2000H:0100H,目的串地址ES:DI为2100H:0000H折算到同一个段内,(2100H－2000H)×16+0000H＝1000H,即目的串地址ES:DI变为2000H:1000H,于是源串和目的串就可以按同一段内传送处理。

2. 取信息串指令LODS

指令格式： LODS src

简洁书写格式：

取字节： LODSB

取字： LODSW

指令功能：从由DS:SI作指针的源串中取一个字节或字，传送到AL或AX中，同时修改指针SI，使其指向串中的下一个元素。SI的修改量由方向标志DF和源串的类型确定。

该指令加重复前缀没有意义。

[例]从2000H:0100H处取一个字放在AX中。

```
CLD
MOV  AX，2000H
MOV  DS，AX
MOV  SI，0100H
LODSW
```

执行后，AX＝1108H，**SI＝**0102H。

3. 信息串存储指令STOS

指令格式： STOS dst

简洁书写格式：

字节存储： STOSB

字存储： STOSW

指令功能：将AL或AX中的一个字节或字，存储到ES:DI为目标指针的目的串中，同时修改DI，以指向串中的下一个单元。DI的修改量由方向标志DF和目的串的类型确定。

STOS指令与REP重复前缀连用，即执行指令REP STOS，能方便地用AL/AX中的一个常数，对一个信息串进行初始化，例如初始化为全0的串。

[例] 往内存1000H:0500H处连续存储4个空格。

```
CLD
MOV  AX，1000H
MOV  ES，AX
MOV  DI，0500H
MOV  AL，20H
MOV  CX，4
REP   STOSB
```

执行后,DI＝0504H。

4. 信息串比较指令CMPS

指令格式： CMPS dst，src

简洁书写格式：

字节比较： CMPSB

字比较： CMPSW

指令功能：从由DS:SI作指针的源串中减去由ES:DI作指针的目的串数据，不回送结果，即不改变两个信息串的原始值，结果影响标志F。同时，操作后源串和目的串指针会自动修改，指向下一对待比较的串。

常用这条指令来比较两个串是否相同，并由加在CMPS指令后的一条条件转移指令，根据CMPS执行后的标志位值决定程序的转向。

在CMPS指令前可以加重复前缀，即：

REPE CMPS 或 REPZ CMPS

这两条指令功能相同，当ZF＝1且CX≠0时重复，直至ZF＝0(发现两串有不相同的)或CX＝0(比完了)为止。退出时，若ZF＝1，则两个信息串完全相同。不能用CX＝0(每重复一次CX减1，但并不影响标志)来判断两个信息串完全相同。此指令常用在“两信息串大部分相同找不同的”的情况。

也可以改用重复前缀REPNE或REPNZ,它们表示:当ZF＝0且CX≠0时重复,直至ZF＝1(发现两串有相同的)或CX＝0(比完了)为止。退出时,若ZF＝0,则两个信息串没有相同的。同样不能用CX＝0(每重复一次CX减1,但并不影响标志)来判断两个信息串没有相同的,如果这样做,就会出现将前面的都不同而最后一个相同的异常情况当作两个信息串没有相同的,这一点也应注意。此指令常用在“两信息串大部分不同找相同的”的情况。

[例] 比较两串各为6个字节的信息串,如发现有不同的字符,转至FOUND。

```
        CLD
        MOV  AX，2000H
        MOV  DS，AX
        MOV  AX，3000H
        MOV  ES，AX
        MOV  SI，1500H
        MOV  DI，0200H
        MOV  CX，6
```

```
        REPE CMPSB
        JNE    FOUND
          ⋮
FOUND:
          ⋮
```

找到后，SI、DI均指向下一个单元，要得到不相同字符的真实位置，尚需修改指针，利用指令DEC SI和DEC DI，即可得到不相同字符的真实位置。

5. 信息串扫描(搜索)指令SCAS

指令格式： SCAS dst

简洁书写格式：

字节扫描： SCASB

字扫描： SCASW

指令功能：从AL(字节操作)或AX(字操作)寄存器的内容减去以ES:DI为指针的目的串元素,不回送结果,但结果影响标志F。同时,操作后目的串指针会自动修改,指向下一个待搜索的串元素。

利用SCAS指令,可在内存中搜索所需要的信息,这个被搜索的信息也称为关键字。指令执行前,必须事先将它存在AL(字节)或AX(字)中,才能用SCAS指令进行搜索。SCAS指令前也可以加重复前缀REPE/REPZ(相等时重复)和REPNE/REPNZ(不等时重复),可以用来扫描某一个串,搜索其中是否有关键字。**具体作法**是:将ES:DI指向被扫描的串,DF表示搜索的方向,**扫描范围**放在CX中,将需扫描的**关键字**放在AL(或AX)中。

指令REPNE SCASB是从目标串找关键字,操作一直进行到ZF=1,表示找到了,此时DI指向下一个单元,CX为剩下的个数;若退出时仍为ZF=0,则表示未找到,此时CX=0。

[例] 编写一内存自检程序，检查从2000H:0000H开始的64KB是否损坏，若损坏转RAM_ERR，若完好转RAM_OK。

```
MOV  AX，2000H
MOV  ES，AX
MOV  DI，0
MOV  CX，8000H
MOV  AX，5555H
CLD
REP  STOSW          ;先给每一单元写入55H
MOV  CX，8000H
MOV  DI，0
REPE SCASW          ;扫描每个单元是否为55H?
JNZ  RAM_ERR        ;有错转RAM_ERR
```

```
        MOV  CX，8000H
        MOV  DI，0
        MOV  AX，0AAAAH
        REP  STOSW  ;再给每一单元写入0AAH
        MOV  CX，8000H
        MOV  DI，0
        REPE SCASW
                    ;扫描每个单元是否为0AAH?
        JNZ  RAM_ERR ;有错转RAM_ERR
RAM_OK:      ┆          ;内存完好处理程序段

RAM_ERR:     ┆          ;内存损坏处理程序段
```

五、控制转移(Control Jump)类指令

1. 无条件转移、调用和返回指令

(1)无条件转移指令(JUMP)

指令格式： JMP 目标标号(TARGET)

这类指令又分成两种类型:

第一种类型 段内转移或近(NEAR)转移。

第二种类型 段间转移,又称为远(FAR)转移。

不论是段内还是段间转移,又可分为两种方式:

①直接转移,指令码中直接给出转移的目的地址。

②间接转移,目的地址包含在某个16位R/M中,CPU必须根据R/M寻址方式,间接地求出转移地址。

① 段内直接转移。指令格式为：JMP 标号

建议写成：JMP NEAR PTR 标号 ;IP ¬ IP+16位位移量,NEAR PTR为运算符。

目标偏移地址＝当前IP值 + 指令中的位移量

位移量可以是8位也可以是16位。

若位移量是16位,最高位为符号位,此时指令为3字节,可在±32 KB范围内转移;

若位移量是8位,则为段内短转移,此时指令为2字节,转移范围在–128～+127之间。

段内短转移写成：JMP SHORT 标号

;IP ¬ IP+8位位移量,SHORT为运算符。

段内直接寻址方式既可用在无条件转移指令中,也可用在条件转移指令中,要特别注意:在条件转移指令中,只能用8位的位移量。

```
                                        ⋮
13BA:001B F3                    REPZ
13BA:001C A6                    CMPSB
13BA:001D 75 01                 JNZ    FOUND
13BA:001F CB            SAME:   RET
13BA:0020 41            FOUND:  INC    CX
13BA:0021 89 0E 06 00           MOV    [0006],CX
13BA:0025 EB F8                 JMP    SAME
                                        ⋮
```

② 段内间接转移。

这类指令要转向的段内目标地址存放于某16位通用R/M中的某两个连续地址中，指令中只需给出R/M地址即可。

用寄存器间接寻址的段内转移指令，要转向的有效地址存放在寄存器中，执行的操作为：IP ← 寄存器内容。

JMP BX

对于用存储器间接寻址的段内转移指令，要转向的有效地址存放在存储单元中，所以JMP指令先要计算出存储单元的物理地址，再从该地址处取一个字送到IP，即：IP ← 字存储单元内容。

JMP WORD PTR 5[BX]

这种指令的目的操作数前要加WORD PTR,进行字操作。

JMP WORD PTR [SI]

JMP TAB ;TAB为DS段中的字型变量

JMP TAB[BX+2]

③ 段间直接转移。

目标在其它代码段中,指令中用远标号直接给出了转向的段地址和偏移量,所以只要用指令中的偏移地址取代IP寄存器的内容,用指令中指定的段地址取代CS寄存器的内容,就可使程序从一个代码段转到另一个代码段去执行。

指令格式： JMP FAR PTR 标号

它是一个5字节指令,机器码的排列次序为:操作码、偏L、偏H、段L、段H 。

④ 段间间接转移。

将目的地址的段地址和偏移量事先放在存储器中的4个连续地址单元中，其中前两个字节为偏移量，后两个字节为段地址，转移指令中只需给出存放目标地址的存储单元的首字节的偏移地址即可。这种指令的目的操作数前常加运算符DWORD PTR，表示转移地址需取双字。低字取代当前的IP，高字取代当前的CS。

JMP TAB1 ;TAB1为DS段内的双字型变量

JMP DWORD PTR VAR

;VAR为DS段中非双字型变量

JMP DWORD PTR [BX] ;目的地址在DS段存放

JMP DWORD PTR [BP][DI] ;目的地址在SS段存放

JMP DWORD PTR [BX][SI] ;目的地址在DS段存放

注意:间接寻址的转移指令,从存储器中取出一个字或双字,默认的段地址为DS。

(2)调用指令CALL

在编写程序时,往往把某些能完成特定功能而又经常要用到的程序段,编写成独立的模块,并把它称为过程(Procedure),习惯上也称作子程序(Subroutine),然后在程序中用CALL语句调用这些过程,调用过程的程序称为主程序。CALL指令迫使CPU暂停执行后续的下一条指令(形成断点),转去执行指定的过程,待过程执行完毕再返回断点继续运行。

子程序和调用程序可以在同一个代码段内,也可在不同的段内,前者称为段内调用(近过程),用NEAR表示;后者称为段间调用(远过程),用FAR表示。

要保证正确返回断点,必须将断点保护起来(进栈),它由机器自动完成。

格式：CALL NEAR PTR 过程名 ;段内调用
CALL FAR PTR 过程名 ;段间调用

CALL指令执行时分两步进行：

① 第一步是保护断点，也就是将当前的IP值推入堆栈。对于段内近调用来说，执行的操作是：

SP ← SP－2
(SP＋1，SP) ← IP

对于段间远调用来说，执行的操作是：

SP ← SP－2
(SP+1，SP) ← CS
SP ← SP－2
(SP+1，SP) ← IP

② 第二步是转到子程序的入口地址去执行子程序。

有四种方式：段内直接调用、段内间接调用、段间直接调用和段间间接调用，无段内短调用指令。

```
CALL  1000H                  ;段内直接调用
CALL  NEAR PTR ROUT ;段内直接调用，“近”过程
CALL  BX                     ;段内间接调用
CALL  2500H:1400H      ;段间直接调用
CALL  FAR PTR SUBR    ;段间直接调用，“远”过程
CALL  DWORD PTR [DI]  ;段间间接调用
```

调用指令对标志F无影响。

若在过程运行中又去调用另一个过程,称为过程**嵌套**。

在模块化程序设计中,过程调用已成为一种必不可少的手段,它使程序结构清晰,可读性强,同时也能节省内存。

过程调用过程本身称为“**递归**”。

(3)返回指令RET

RET指令应安排在过程的出口处,它的功能是从堆栈中弹出由CALL指令压入的断点地址值,迫使CPU返回到主程序的断点去继续执行。RET指令为子程序最后执行的一条指令。

CALL与RET应配对使用。

返回指令的一般格式： RET 或 RET n

若为段内返回,RET指令的操作如下：

IP ← (SP+1,SP)

SP ← SP+2

若为段间返回,RET指令的操作如下：

IP ¬ (SP+1,SP)

SP ¬ SP+2

CS ¬ (SP+1,SP)

SP ¬ SP+2

段内和段间返回均用RET指令,其操作的区别由与之配用的CALL指令中的过程名确定。

RET指令还可带立即数n,如RET 4,该指令在返回地址出栈后,继续修改堆栈指针,将立即数4加到SP中,即SP ¬ SP+4,这一特性可用来冲掉在执行CALL指令之前推入堆栈的一些数据。

RET n

要注意：n一定为偶数。带立即数的返回指令RET n 一般用在这样的情况：调用程序为某个子程序提供一定的参数或参数的地址，在进入子程序前，调用程序将这些参数或参数地址先送入堆栈，通过堆栈传递给子程序；子程序运行过程中，使用了这些参数或参数地址；子程序返回时，这些参数或参数地址已经没有在堆栈中保留的必要，而“堆栈的容量是有限的”，因此，可在返回指令RET后面带上立即数n，使得在完成返回的同时，再多弹出n个字节作废，这一功能称作“清栈”。

RET n 的另外用途是,若子程序执行完后不想返回到断点处,而想转移到其它处,可用如下方法实现:

① 段内返回到××××H处,在子程序的尾部加以下指令:

```
MOV  AX，××××H      ;×××× H为偏移地址
PUSH  AX
RET   2
```

②段间返回到****H:××××H处,可在子程序的尾部加以下指令:

```
MOV  AX，****H       ;****H为段地址
PUSH  AX
MOV  AX，××××H      ;××××H为偏移地址
PUSH  AX
RET   4
```

2. 条件转移指令

指令格式： JCC 目标标号

CC为转移条件，根据上一条指令执行后标志寄存器F的状态决定是否转移。所有条件转移均为短转移，指令机器码为2字节，第一字节为操作码；第二字节为8位地址位移量，范围为–128～+127。

条件不满足，程序仍顺序执行；

条件满足，IP ← IP当前值 + 8位地址位移量，转移至目标地址去执行。

注意：8位地址位移量是用符号位扩展法扩展到16位才与IP值相加的。

条件转移指令通常用在比较指令或算术逻辑运算指令之后，根据比较或运算结果，转向不同的目的地址。**注意：所有条件转移指令均不影响标志位。**

条件转移指令共有18条，可以归类成以下几大类。

(1)直接标志转移指令

这类转移指令在指令助记符中，直接给出标志状态的测试条件，它们以CF、ZF、SF、OF和PF等5个标志的10种状态为判断的条件，共形成10条指令。

指令助记符	测试条件	指 令 功 能	
JC	CF＝1	有进位	转移
JNC	CF＝0	无进位	转移
JZ/JE	ZF＝1	结果为零/相等	转移
JNZ/JNE	ZF＝0	不为零/不相等	转移
JS	SF＝1	符号为负	转移
JNS	SF＝0	符号为正	转移
JO	OF＝1	溢出	转移
JNO	OF＝0	无溢出	转移
JP/JPE	PF＝1	奇偶位为1/为偶	转移
JNP/JPO	PF＝0	奇偶位为0/为奇	转移

(2)间接标志转移

类别	指令助记符	测试条件	指令功能	
无符号数比较测试	JA/JNBE	CF∨ZF=0	高于/不低于也不等于	转移
	JAE/JNB	CF=0	高于	转移
	JB/JNAE	CF=1	低于/不高于也不等于	转移
	JBE/JNA	CF∨ZF=1	低于等于/不高于	转移
带符号数比较测试	JG/JNLE	(SF⊕OF)∨ZF=0	大于/不小于也不等于	转移
	JGE/JNL	SF⊕OF=0或ZF=1	大于等于/不小于	转移
	JL/JNGE	SF⊕OF=1且ZF=0	小于/不大于也不等于	转移
	JLE/JNG	(SF⊕OF)∨ZF=1	小于等于/不大于	转移

条件转移指令转移的范围只有–128～+127,如果超出此范围,程序编译执行时就会报告出错,无法执行,那么如何解决条件转移超范围或明知转移超范围而又不得不使用条件转移指令的问题呢?方法有二:

(1)“接力法”,先用条件转移指令转移到它力所能及的范围,再用一条直接转移指令转移到目标地址;

(2)“反指令法”,将所使用的条件转移指令改为其反指令,反指令转移的目标地址为刚才使用的条件转移指令条件不满足时顺序执行的下一条指令,再添加一条直接转移指令转移到目标地址。

何为反指令呢?例如JC与JNC、JG与JLE互为反指令,以上讲到的条件转移指令两两互为反指令。

接力法

转移超范围：

```
AGAIN:  ┆                          AGAIN:  ┆
        ┆                                  ┆
        ┆                                  ┆
     JG AGAIN                          JLE  NEXT
NEXT:   ┆                             JMP  AGAIN
                                   NEXT:  ┆
```

转移超范围

反指令法

习惯用法：对于有反指令的超范围条件转移，多使用“反指令法”。

(3)根据CX中的值来决定是否转移

指令格式:　　JCXZ　目标标号

;不对CX的内容进行操作,只根据CX的内容控制转移,CX＝0时转移,否则顺序执行。

该指令转移的范围亦为–128～+127,而它又无对应的反指令,若转移超范围,则只能用“接力法”来实现。

若程序循环体为“先执行,后判断”结构,则最少要执行循环体一次,故本指令常用在循环体的入口处,构成“先判断,后执行”结构; 若CX＝0,则循环体一次也不执行,可以提高程序的效率。

3. 循环控制指令

控制一段程序的重复执行,重复次数由CX决定。

2字节指令,与条件转移类似,转移的目标地址等于当前IP值加上8位位移量,这类指令均不影响标志F。

(1) LOOP循环指令

指令格式： LOOP 目标标号

功能:控制重复执行一系列指令。先将重复次数放在CX中,每执行一次LOOP指令,CX自动减1(但不影响标志)。如果减1后CX≠0,则继续循环;若自动减1后CX＝0,则结束循环,转去执行后继指令。一条LOOP指令在功能上相当于执行以下两条指令的功能:

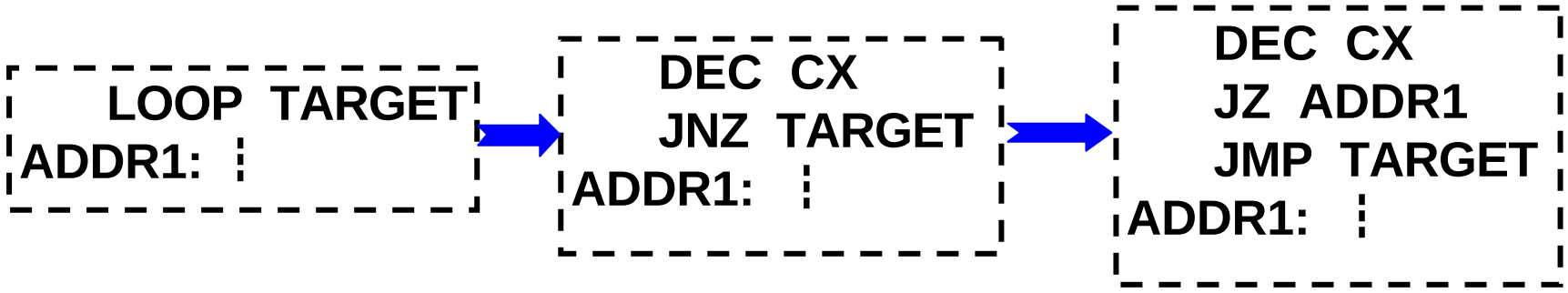
```
DEC   CX
JNZ   目标标号
```

将内存区BLOCK中N个字节的数据按+、–分开，分别送到两个缓冲区PLUS_DATA及MINUS_DATA。程序如下：

```
LEA   SI,BLOCK          ;SI指向内存数据区
LEA   DI,PLUS_DATA      ;DI指向正
LEA   BX,MINUS_DATA     ;BX指向负
CLD                     ;由低到高
MOV   CX,N              ;置循环次数
```

```
AGAIN:   LODSB              ;从数据区取一个字节
         TEST  AL,80H       ;判断正负
         JNZ   MIUS         ;为负转MIUS
         STOSB              ;存正数,同时修改指针
         JMP   NEXT_N       ;转下一个
MIUS:    XCHG  BX,DI        ;BX与DI交换,使DI指向负
         STOSB              ;存负数,同时修改指针
         XCHG  BX,DI        ;修改后的指针与BX交换
NEXT_N:  LOOP  AGAIN        ;未做完,继续
         ┆                  ;结束
```

LOOP指令循环转移的范围为–128～+127,若超范围,则可用下列指令来实现：给LOOP指令之后的那条指令加标号ADDR1,再用“反指令法”来完成。



(2)LOOPE/LOOPZ相等或结果为零时的循环指令

格式：LOOPE 目标标号 或 LOOPZ 目标标号

功能：LOOPE是相等时循环,LOOPZ是结果为零时循环,这是两条能完成相同功能,而具有不同助记符的指令,用于控制重复执行一组指令。先将重复次数放在CX中,每执行一次指令,CX自动减1。若减1后CX≠0且ZF＝1,则继续循环；若CX＝0或ZF＝0,则退出循环,转去执行LOOPE/LOOPZ指令之后的那条指令。

注意：CX自动减1后即使CX＝0时也不影响标志,ZF是否为1是由LOOPE/LOOPZ之前的指令执行所影响的。

LOOPE/LOOPZ指令循环转移的范围为–128～+127，若超范围，则可这样来实现：给LOOPE/LOOPZ指令之后的那条指令加标号ADDR1，再用“反指令法”来完成。

```
        LOOPZ  TARGET
ADDR1:
```

```
        JNZ   ADDR1
        DEC   CX
        JZ    ADDR1
        JMP   TARGET
ADDR1:
```

(3)LOOPNE/LOOPNZ不相等或结果不为零循环指令

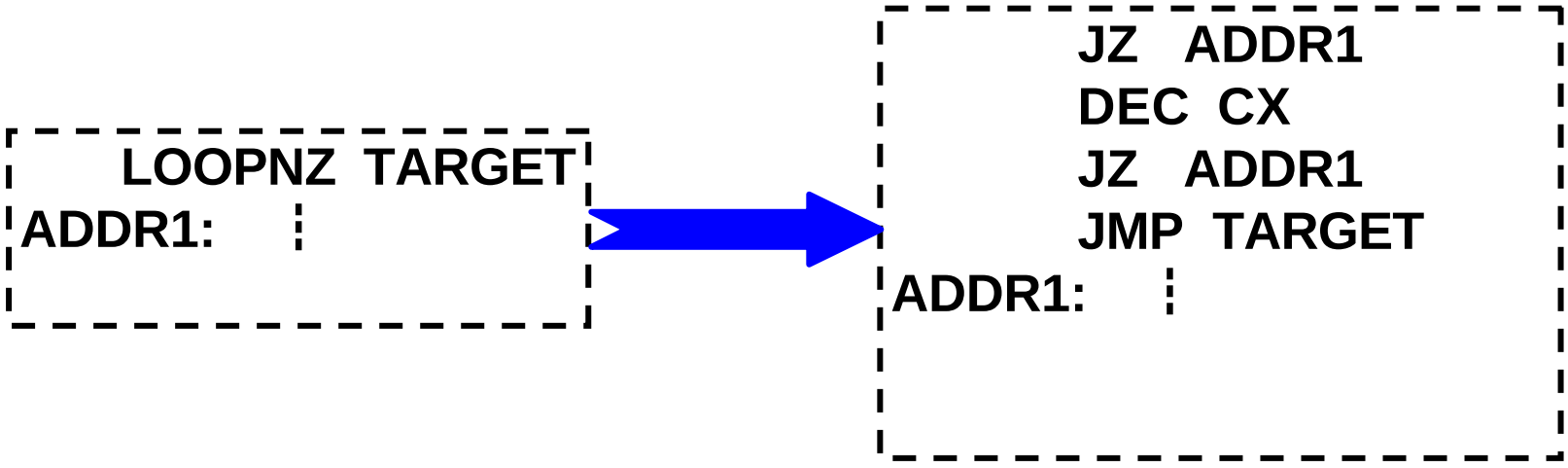
格式：　　LOOPNE　目标标号

　　或　　LOOPNZ　目标标号

功能：LOOPNE是不相等时循环,而LOOPNZ是结果不为零循环,它们也是一对功能相同但形式不一样的指令。指令执行前,应将重复次数送入CX,每执行一次,CX自动减1。若减1后CX≠0且ZF＝0,则转移到标号所指定的地方重复执行; 若CX＝0或ZF＝1,则退出循环,顺序执行下一条指令。

注意： ZF不受CX减1的影响,ZF是否为0是由LOOPNE/LOOPNZ之前的指令执行所影响的。

LOOPNE/LOOPNZ指令循环转移的范围为：–128～+127，若超范围，则可这样来实现：给LOOPNE/LOOPNZ指令之后的那条指令加标号ADDR1，再用“反指令法”来完成。



六、处理器控制指令

1. 标志位操作指令

除了指令执行后会影响标志外，8086/8088还提供了一组标志操作指令，可直接对CF、DF和IF等标志位进行设置或清除操作，但不包含TF标志。执行后不影响其它标志，只影响本指令指定的标志。

指令助记符	功	能
CLC	CF ← 0	进位标志清0
CMC	CF ← $\overline{CF}$	进位标志取反
STC	CF ← 1	进位标志置1
CLD	DF ← 0	方向标志位清0
STD	DF ← 1	方向标志位置1
CLI	IF ← 0	中断标志位清0
STI	IF ← 1	中断标志位置1

2. 其它控制指令

(1)暂停指令HLT

当CPU执行HLT指令时,暂停程序运行。用软件方法使CPU处于暂停状态等待硬件中断,CS和IP指向HLT后面的一条指令地址。一旦外部有一个硬件中断产生,只要IF为1,CPU便可用2个总线周期响应中断,在处理完中断而返回之后,CPU接着执行HLT后面的一条指令。

此外,对系统进行复位,也会使CPU退出暂停状态。

(2)空操作指令NOP

单字节指令,空耗3个时钟周期,而不完成任何操作,不影响标志位。NOP指令通常被插在其它指令之间,在循环等操作中增加延时。还常在调试程序时使用NOP指令。例如用3条NOP指令代替一条调试暂不执行的3字节指令,待调试好后再用这条指令退换掉NOP指令。

NOP指令有一个更重要的用途就是利用其机器码占用一个字节单元,在适当的地方预留一些备用单元,以便修改时插入指令。尤其是条件转移和循环控制指令,编程时无法准确知晓转移是否超范围,更要多用NOP指令。

3. 同步控制指令

8086 CPU具有多处理机的特征,为了充分发挥硬件的功能,设置了3条使CPU与其它协处理器同步工作指令,以便共享系统资源。这些指令执行后均不影响标志位。

(1)交权指令ESC

指令格式： ESC 外部操作码，源操作数

指令功能： ESC指令用来实现8086对协处理器的控制。

(2)等待指令WAIT

(3)总线锁定前缀LOCK

第四节　中断指令及DOS功能调用

一、中断

当系统运行时或程序运行期间,遇到特殊情况,需要CPU停止执行当前程序,产生断点,转去处理例行程序,这种情况称为**中断**(Interrupt),这种例行程序称为中断处理程序。在中断处理程序的末尾设置一条返回指令,称作**中断返回**(Interrupt Return),其功能有点像调用子程序,但和调用子程序又有所不同,调用子程序是人为安排的,中断是随机发生的(当然也可以人为安排);**调用子程序只需保护断点**,中断由于其随机性不仅要保护断点还需保护现场,统称**保护现场**。

在8086的中断机构中，包含两类中断源:一类是**外部中断**，它是外部事件通过8086的引脚INT和NMI向CPU发出中断请求信号而引起中断的，这类中断称为硬中断，或者称为外部中断;另一类是**内部中断**，内部中断是CPU执行中断指令而产生的，常称为**软中断**。

1. 中断指令 INT n

该指令为双字节指令，n为中断类型号，占用一个字节，其值在0～255之间的任何数，这个范围包括了所有可能的**硬中断**和**软中断**，8086共有256级中断。

当某一外部事件发生而CPU又允许其中断时，或执行一条**软中断**时，中断处理程序的入口地址由**中断向量表**（内存最低端PA＝00000H～003FFH的1KB空间，即逻辑地址0000H:0000H～0000H:03FFH）里相应的中断向量来决定。每级中断占用四个字节单元用以存放中断向量，前两个单元存放中处程序入口地址的偏移量（即IP），后两个单元存放中处程序入口地址的段地址（即CS）。

INT n 指令执行时完成如下操作:

```
SP ← SP－2                    ;SP－2
(SP+1,SP) ← F                 ;标志寄存器F入栈
IF ← 0                        ;中断允许标志IF清零
TF ← 0                        ;陷阱标志TF清零
SP ← SP－2                    ;SP－2
(SP+1,SP)← CS                 ;CS入栈
SP ← SP－2                    ;SP－2
(SP+1,SP)← IP                 ;IP入栈
IP ←(n×4+1,n×4)               ;取中处程序偏移地址
CS ←(n×4+3,n×4+2)             ;取中处程序段地址
```

于是CPU就转去执行该中断处理程序。

INT n 中断指令中,类型号0～4,共五种中断属于8086的专用中断：

(1)除法错中断(类型0)

(2)单步中断(类型1)

(3)不可屏蔽中断(类型2)

当8086的NMI引脚上接收到由低变高的电平变化时,将自动产生类型号为2的不可屏蔽中断。

(4)断点中断(类型3)

是为调试程序而设置的,它的类型号为3。

(5)溢出中断(类型4)

如果溢出标志OF置1,则可由溢出中断指令INTO产生类型为4的中断。

2. 断点中断指令 INT

该指令为单字指令,相当于n＝3的INT 3指令,但类型号3被省去了,称为断点中断,是8086提供给用户的一种调试手段,一般用在DEBUG中。

3. 溢出中断指令 INTO

4. 中断返回指令 IRET

它总是被安排在中断处理程序的出口处,用以退出中断,返回到中断断点处继续执行原来被中断的程序,其操作如下:

IP ←(SP+1,SP)　　;弹出IP
SP ← SP+2　　;SP+2
CS ←(SP+1,SP)　　;弹出CS
SP ← SP+2　　;SP+2
F ←(SP+1,SP)　　;弹出F
SP ← SP+2　　;SP+2

二、DOS的系统功能调用和基本I/O子程序调用

1. 8086中断类型号的分配

在8086允许的256级(即00H～0FFH)中断类型中，除00H～04H级规定为专用的中断外，PC机把类型号为08H～1FH的分配给主板和扩展槽上的基本外设的中断服务子程序和BIOS ROM中的I/O子程序调用指令;把类型号为20H～0FFH中的一些分配给DOS中的功能子程序调用指令，其中的40H～7FH留给用户，作为开发使用的中断类型号。

2. DOS系统功能调用(Function CALL)

80多个子程序按服务功能可分为三个方面：

① 磁盘的读写管理；

② 内存管理；

③ 基本I/O管理(包括对键盘、打印机、显示器和磁盘等的管理)以及对时间、日期的处理子程序。

DOS所有的功能子程序调用都利用INT 21H 中断指令。

在调用这些子程序时,应给出以下三方面的内容:

① 入口参数(有些子程序不需要入口参数,但大多数子程序需要将有关参数送入指定地点);

② 子程序的功能号送入AH寄存器;

③ INT 21H。

使用DOS系统功能调用还需注意以下几点：

从键盘输入的字符在机器内由系统自动转换为ASCII码，所以对它们必须进行反转换后才能进行数据处理；

需要在屏幕上显示的字符，必须在程序中把它们转换为ASCII码，然后再送到该调用所指定的单元中去；

在对09H号调用时，要在被显示字符串后加上字符串结束标志'$'，以通知系统字符串显示完毕；

调用结束后，如果要返回DOS或DEBUG状态，应在程序后加入退出语句：

```
MOV  AH，4CH
INT   21H
```

3. 基本I/O子程序调用

(1)键盘输入并回显(功能号01)

```
MOV  AH,1
INT   21H
```

(2)显示输出(功能号02)

```
MOV  AH,02
MOV  DL,'$'
INT   21H
```

(3)打印机输出(功能号05)

```
MOV  DL,'*'
MOV  AH,05
INT   21H
```

(4)直接控制台I/O(功能号06)

该功能调用既可执行输入操作,也可以执行输出操作。如果DL的数值是除0FFH以外的其它数值,则6号调用执行显示输出操作,把DL寄存器内容对应的字符在屏幕上显示,例如:

```
AGAIN:   MOV  DL,0FFH
         MOV  AH,6
         INT   21H
        JZ  AGAIN  ;等待从键盘输入一个字符。
```

而执行程序:

```
        MOV  DL,35H
        MOV  AH,6
        INT   21H     ;在屏幕上显示5。
```

(5)键盘输入无回显(功能号07)

7号功能调用和1号功能调用相类似,等待从标准输入设备(键盘)输入字符,然后将其对应的ASCII码值送入AL寄存器。和1号调用不同的是,7号调用的执行不在屏幕上显示出输入的字符,同时,它不检查字符代码是否为Ctrl-Break。

(6)键盘输入无回显(功能号08)

(7)显示以'$'结尾的字符串(功能号09)

9号功能调用可以方便地用来显示多个字符,即显示字符串。存储器中需要显示的字符串应以'$'作为结束标志,非显示字符(如回车、换行符等)也可以包含在字符串内,但不能把'$'作为要显示的字符包含在串内。因此,前面提到的2号功能调用可以显示'$',可作为9号功能调用的一个补充。

DS:DX指向字符串所在存储缓冲区首地址,字符串以ASCII值存放,其后紧跟结束符$的ASCII码24H。

把字符串How are you?显示在屏幕上。

程序如下：

```
        ⋮
STR1    DB    'How are you?',13,10,'$'
        ⋮
        MOV   AH,9
        MOV   DX,SEG  STR1
        MOV   DS,DX
        MOV   DX,OFFSET  STR1
        INT   21H
        ⋮
```

(8)字符串键盘输入(功能号0AH)

从键盘接收字符串存入内存,屏幕无显示。要求事先定义一个输入缓冲区,它的始址放于DS:DX

第一个字节指出缓冲区能容纳的最大字符数(1～255)不能为零,该值由用户设置;

第二个字节保留以用作由DOS返回实际读入的字符数(回车除外);

从第三个字节开始存放从键盘上接收的字符。

实际输入少于定义的字节数,缓冲区补零;若多于定义的字节数,则丢掉并且响铃。DOS还自动在字符串的末尾加上回车字符,而这个回车字符未被计入实际输入的数目之中。因此,缓冲区最大尺寸要比所希望输入的字节数多一个字节。

(9)返回DOS(功能号4CH)

终止当前程序,并把控制权交给调用程序。

```
MOV   AH,4CH
INT    21H
```

这两条指令应放在程序的结束处。

(10)取日期(功能号2AH)和设置日期(功能号2BH)

```
MOV   AH,2AH
INT    21H
```

提取系统的年、月、日、星期。调用返回后,年®CX,月®DH,日®DL。它们均为二进制数。AL中存放星期号(0为星期天,1为星期一,…)。

(11)取得时间(功能号2CH)和设置时间(功能号2DH)

```
MOV   AH,2CH
INT    21H
```

本调用提取系统的时间。调用返回后，
小时®CH,分®CL,秒®DH,秒/100®DL。
它们均为二进制数。
该调用不需要入口参数。

三、BIOS中断调用

格式：　　INT　n

n＝8～1FH,每一个n代表一种I/O硬件的中断调用，每一种中断调用;又以功能号区分控制功能。

用法与DOS功能调用类似：功能号®AH；准备入口参数;然后写中断指令。

1. 键盘I/O中断调用　(INT　16H)
2. 打印机I/O中断调用（INT　17H)
3. 时间处理中断调用　（INT　1AH)

4. 显示中断调用(INT 10H)

① AH＝0,设置显示方式。

入口参数：AL＝显示方式号

AL＝ 3　80列×25行彩色文本方式

AL＝ 5　320列×200行彩色图形方式

② AH＝2,设置光标位置。

入口参数：BH＝页号,通常取0页;

DH＝行号,取值0～24;

DL＝列号,对于40列文本,取值0～39,

对于80列文本,取值0～79。

③ AH＝6,屏幕上滚。

入口参数：AL＝上滚行数，

当AL＝0时,清除屏幕矩形方框，

CH、CL＝矩形方框左上角行、列号;

DH、DL＝矩形方框右下角行、列号;

BH＝增加空行的属性。

④ AH＝7,屏幕下滚。与功能号6滚动方向相反。

⑤ AH＝9,在当前光标处写字符和属性。

入口参数：BH＝页号，

AL＝显示字符的ASCII码，

BL＝属性

CX＝重复显示次数

⑥ AH＝0BH,设置图形方式显示的背景和彩色组。

入口参数：当BH＝0时,BL＝背景颜色,范围0～15;

当BH＝1时,BL＝彩色组,范围0～1,

0表示绿/红/黄,1表示青/品红/白。

⑦ AH＝0CH,写光点。

入口参数：DX＝行号,

CX＝列号,

AL＝彩色值(当AL第7位为1时,原显示彩色与当前彩色作按位加运算)。

⑧ AH＝0DH,读点。该功能与功能号0C操作相反。

四、返回DOS的中断调用

1. 程序结束中断INT 20H

执行这条指令后,退出当前程序,返回操作系统。

2. 程序驻留结束中断INT 27H

INT 27H和INT 20H都是实现退出程序,但它们也有不同,即INT 27H执行中断后,将程序看作COMMAND.COM常驻部分的属性,不会被其它调入的程序所覆盖;另外,INT 27H入口时,需输入参数到DX中。DX中包含程序最末地址后的第一字节的位移量(段地址在CS中),即入口参数:

DX＝程序最末端地址+1,例如,

```
MOV   DX,OFFSET LAST_INSTRUC
INC    DX
INT    27H
```

程序驻留于内存,要求后缀为 .COM。特征是程序无堆栈段,数据、代码在同一个段内,且定位于100H。

若驻留程序超过64 KB时,则使用DOS系统功能调用的31H号功能调用,其用法为:

```
MOV   AH, 31H
MOV   DX,驻留字节数
INT     21H
```

3. 结束当前程序并返回DOS的中断

```
MOV   AH, 4CH
INT    21H
```

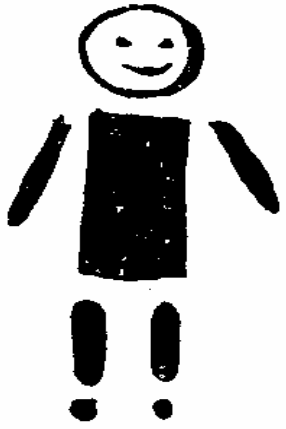
利用“IBM PC机的字符集”中的方块图形字符也能产生一些简单图形,由多个方块图形字符还能装配成一个较复杂的图形,若再加延时和对原图形的清除,还能使字符图形移动,构成动画。

为了在屏幕上显示一个方块图形,程序中要给出两种信息:

① 该图形的ASCII值,

② 给出一个8位的显示属性,该字节中包含指定字符或背景的颜色,显示的亮度以及显示是呈连续还是呈闪烁。

例:编一个程序在屏幕上定点显示左图的小人图形。



多字符图形

该小人图形由五种方块字符图形构成,并将头定在参考点10行40列上,指定头部方块的图形的参考点座标为(0,0),其余各方块座标用相对于头的偏移量表示,这样,形成一个字符表CHRTAB存于数据段中。每个方块字符含三个数据:它的ASCII和相对于参考点的行、列偏移量,并将此三个数据分别取入AL,DH和DL中。

```
PDATA       SEGMENT
CARTAB      DW    5
            DB    01,0,0,0DBH,1,0,13H,1,1
            DB    2FH,-1,-1,5CH,0,2
PDATA       ENDS
;-------------------------------------------------------------------
STACK       SEGMENT  PARA  STACK 'STACK'
            DB    100  DUP(?)
STACK       ENDS
;-------------------------------------------------------------------
CODE        SEGMENT
            ASSUME   CS:CODE,DS:PDATA
PICTURE     PROC  FAR
START:      PUSH  DS
            MOV   AX，0
            PUSH  AX
            MOV   AX，PDATA
            MOV   DS，AX
;-------------------------------------------------------------------
            STI
            MOV   AL，02          ;设置显示方式: 80 * 25黑白
            MOV   AH，0
            INT   10H
```

```
;-------------------------------------------------------------------
                MOV     DI，OFFSET CHRTAB
                MOV     CX，[DI]
                MOV     DH，10          ;定参考点于10行40列
                MOV     DL，40
                ADD     DI，2
;-------------------------------------------------------------------
NEXT:           ADD     DH，[DI+1]
                ADD     DL，[DI+2]
                MOV     AH，2
                INT     10H
                MOV     AL，[DI]
                PUSH    CX
                MOV     CX，1
                MOV     AH，10
                INT     10H
                POP     CX
                ADD     DI，3
                LOOP    NEXT
;-------------------------------------------------------------------
                RET
PICTURE         ENDP
CODE            ENDS
                END     START
```

动画显示

若在定点显示图形的程序段再套上一层外循环，循环次数由活动范围定；并不断修改DX中参考点座标，定时清除原位置上的图形，便可构成动画，具体步骤如下：

(1)在屏幕上显示图形；
(2)延迟一个时间；
(3)清除图形；
(4)改变图形的行列座标；
(5)返回第一步，重复上述过程。