CHAPTER 3

第3章

指令系统和寻址方式

任务和目的：

- 掌握基本指令的格式、使用方法和规则；
- 掌握基本寻址方式及其应用。

解决方案：

- 用搬家程序、ASCII 码变为压缩的 BCD 码等程序的设计，来说明汇编语言程序的基本结构和各种伪指令所起的作用；
- 通过对数组操作的实例说明各种寻址方式的作用；
- 用流程图说明编程思路，通过程序说明各种指令的使用规则；
- 用 DEBUG 演示各种指令运算处理的过程。

基本知识点：

- 数据传送指令、算术运算指令、逻辑运算指令、串操作指令和程序控制等指令的使用规则；
- 算法流程的编写思路。

以数据搬家为例说明程序的基本结构，说明伪指令、数据传送类指令、算术运算类指令、程序控制等类指令的使用规则，同时也简单引入寻址的概念。使学生无论对指令还是对程序设计都先有个基本的概念，同时也留下很多悬念，让他们带着问题去学习指令系统和寻址方式。

引例：把数据段中以 AREA1 标识的一组数据顺次搬到 AREA2 数据区中去。

解决任务的方法是用算法流程图解析处理过程，即描述编程思路。

任务是数据搬家，那么首先解决从哪儿往哪儿搬？搬多少？然后是怎么搬？当把问题任务交代清楚时，往往思路就出来了。所谓流程图就是算法思路的图形描述，即把每个思考点用一个框表示出来，然后把它们连接在一起构成一个完整的处理流程，这就是流程图。数据搬家流程图如图 3-1 所示，检查流程是否合理，然后对流程图中的每个框用一个语句或多个语句表达出来就构成了程序。只要流程图正确，语句运用得当，所编写的程序基本没有太大问题。

例 3-1 数据搬家程序。

程序代码如下：

```
DATA  SEGMENT
    AREA1  DB "1234567890"
```

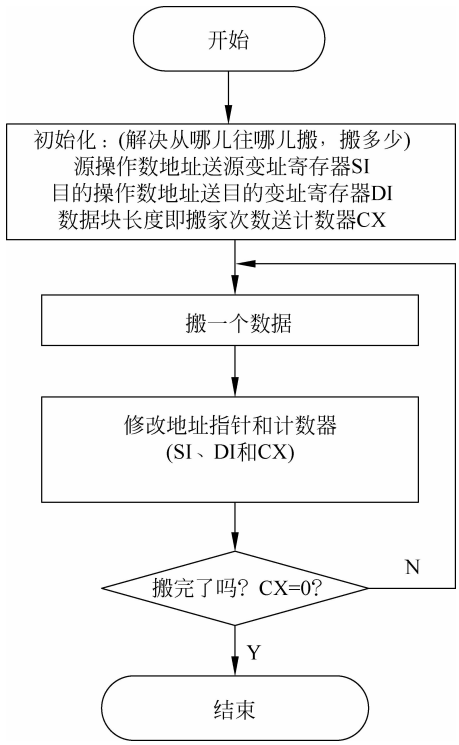

图 3-1　数据搬家流程图

```
    COUNT   EQU   $ - AREA1
    AREA2   DB    COUNT  DUP (0)
DATA  ENDS
STACK  SEGMENT  PARA STACK 'STACK'
    DB  100  DUP (0)
STACK  ENDS
CODE  SEGMENT
        ASSUME CS: CODE,DS: DATA,SS: STACK
START: MOV   AX,DATA
       MOV   DS,AX
       MOV   SI,OFFSET AREA1
       MOV   DI,OFFSET AREA2
       MOV   CX,COUNT            ; 初始化
       CLD
LOP:   MOV   AL,[SI]
       MOV   [DI],AL             ; 搬一个数据
       INC   SI
       INC   DI
       DEC   CX                  ; 修改地址指针和计数器
       JNZ   LOP                 ; 控制循环
       MOV   AH,4CH
       INT   21H                 ; 结束
CODE   ENDS
       END   START
```

如用串操作指令程序还可以简化成：

```
LOP: MOVS BYTE PTR [DI],[SI]
DEC  CX
JNZ  LOP
```

或

```
LOP: REP  MOVSB
```

这里用了 CLD 清方向标志处理器命令，也可以用 STD。前者使地址按增量方式变化；后者使地址按减量方式变化。当两个区域有重叠部分时，使用 STD 控制地址按减量方式变化，可以从后往前搬家。

分析整个程序会发现，用段定义、变量定义、符号定义等伪指令；用立即寻址方式、寄存器寻址方式和寄存器间接寻址方式；用大量的数据传送指令 MOV 和少量算术运算指令；此外还用了循环结构和系统功能调用，目的是让学生对汇编程序建立基本概念，有个大致了解，可以在以后的章节中逐步加深认识。

例 3-2　把数据段中的数组数据逐个由 ASCII 码形式变成压缩的 BCD 码形式，然后存储在另一个区域中，实现数据搬家和压缩。

解决思路依然是用流程图描述，现在把任务变成将数据段中的数据先读出来，然后变成压缩的 BCD 码，最后再存储到相应的存储区中去，其流程图如图 3-2 所示。

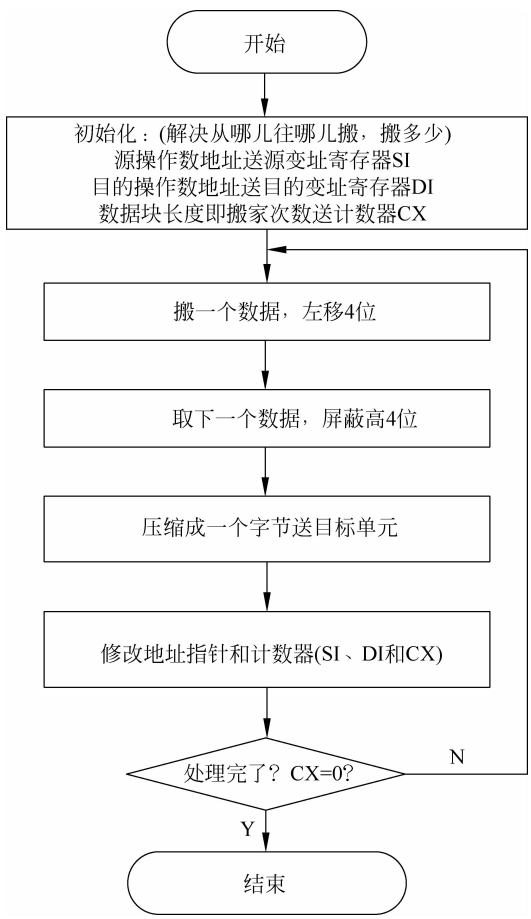

图 3-2　ASCII 码变压缩 BCD 码流程图

依据流程图写出汇编源程序代码如下：

```
DATA   SEGMENT
    AREA1  DB '1234567890'
    COUNT   EQU   $ - AREA1
    AREA2   DB   COUNT/2  DUP (0)
DATA   ENDS
STACK  SEGMENT  PARA  STACK  'STACK'
  DB   100 DUP(0)
STACK  ENDS
CODE   SEGMENT
       ASSUME CS: CODE,DS: DATA,SS: STACK
       START:MOV AX,DATA
       MOV   DS,AX
       MOV   CX,COUNT/2
       LEA   SI,AREA1
       LEA   DI,AREA2
       CLD                          ; 初始化
LOP: MOV  AL,[SI]                   ; 搬一个数据
       SHL   AL,1
       SHL   AL,1
       SHL   AL,1
       SHL   AL,1                   ; 左移 4 位
       INC   SI
       MOV   BL,[SI]
       AND   BL,0FH                 ; 取下一个数据,屏蔽高 4 位
       OR    AL,BL
       MOV   [DI],AL                ; 压缩成一个字节送目标单元
       INC   SI
       INC   DI                     ; 修改地址指针
       DEC   CX                     ; 修改计数器
       JNZ   LOP                    ; 控制循环
       MOV   AH,4CH
       INT   21H                    ; 结束
CODE  ENDS
       END   START
```

本例子是在前例的基础上增加了逻辑运算指令 AND、OR 和移位指令 SHL。在本程序里增加这些指令的目的是处理压缩的 BCD 码。所谓压缩的 BCD 码就是用一个字节存放两个 BCD 码。BCD(Binary/Coded Decimal)是二-十进制码，即用 4 位二进制数表示一位十进制数。它与相对应的 ASCII 码只差高位 4 位二进制 0011B，所以处理时先把高位的 4 位二进制数去掉就变成了未压缩的 BCD 码。例如，1 的 ASCII 码是 31H，写成二进制码是 00110001B。去掉高 4 位后得 1(二进制 BCD 码 00000001B)，前面的 4 个 0 空占位置，如果把另一个 ASCII 码所对应的 BCD 也去掉向 4 位无用的 0，然后两者组合在一个字节里，就成了压缩的 BCD 码，这样可以成倍地提高存储空间的利用率，有一定实用价值。

3.1 指 令 系 统

所谓指令就是控制计算机进行操作的指示和命令，而指令系统则是一种机器全部指令的集合。不同种 CPU 芯片的指令系统不一定一样，但大致可以把汇编语言的指令系统分

为 6 类：

- 数据传送类指令；
- 算术运算类指令；
- 逻辑处理类指令；
- 串操作类指令；
- 控制转移类指令；
- 处理器类指令。

由于计算机的数据是在传输的过程中加工处理的，所以各种汇编语言程序里数据传送类指令占的比重最大。数据传送类指令主要如下：

(1) 数据传送指令 MOV(Move)。

(2) 数据交换指令 XCHG(Exchange)。

(3) 入栈出栈指令 PUSH(Push Onto The Stack)、POP(Pop From The Stack)。

(4) 有效地址送寄存器指令 LEA(Load Effective Address)。

(5) 取地址指针到 DS 指令 LDS(Load DS With Pointer)。

(6) 取地址指针到 ES 指令 LES(Load ES With Pointer)。

(7) 标志位传送指令。

① LAHF (Load AH With Flag)标志位寄存器低 8 位送 AH。

② SAHF(Save AH With Flag)AH 内容送标志位寄存器低 8 位。

(8) 输入输出指令 IN(Input)、OUT(Output)。

(9) 查表转换指令 XLAT(Translate)。

(10) 标志位处理指令。

① PUSHF(Push The Flag)标志位进栈指令。

② POPF(Pop The Flag)标志位出栈指令。

本书重点介绍前三种数据传送指令，其他数据传送指令安排在适当的地方作引用。

3.1.1　数据传送类指令

本小节主要对数据传送类指令中的数据传送指令、数据交换指令、地址传送指令、堆栈操作指令、标志传送指令作简单介绍。前面提及的查表指令和输入输出指令放到其他章节中介绍。

1. MOV 指令

其格式如下：

```
MOV   DST,SRC
```

功能：将(SRC) → DST。

MOV(Move)是操作码，是助记符，DST(Destination)是目的操作数，SRC(Source)是源操作数。该语句的功能是将源操作数 SRC 中的数据传送给目的操作数 DST，源操作数可以是一个字节 Byte(8 位)，一个字 Word(16 位)，还可以是一个双字 Double Word(32 位)等。有三个问题需要说明：第一 DST 必须是具有存储数据能力的寄存器、存储器，而 SRC 可以是寄存器、存储器和立即数；第二 DST 和 SRC 必须保持数据类型一致，即都是字节

型、字型和双字型，但不能同时是内存操作数；第三 DST 不能是立即数，因为立即数不具有存储数据的能力。此外，DST 也不能是代码段寄存器 CS，代码段寄存器是不允许由程序赋值的。注意到上述三点后可以写出很多数据传送的例子，为了方便，这里给出数据传送示意图作为向导，如图 3-3 所示。

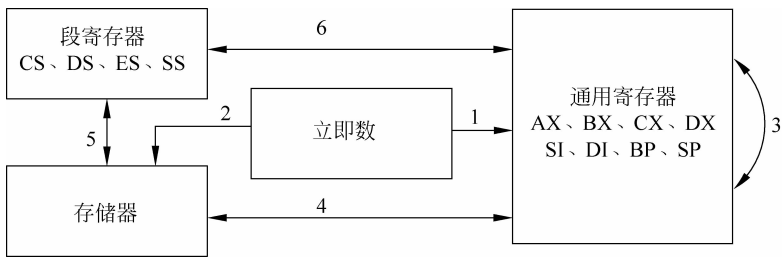

图 3-3 MOV 指令数据传送导向示意图

根据这个示意图和前面提及的三点说明，可以写出很多数据传送的指令的例子。例如：

```
(1) MOV  AH,2
    MOV  CX,128
(2) MOV  VARB, - 2
    MOV  VARW,30000
    MOV  BYTE  PTR [1200H],120
(3) MOV  AH,BL
    MOV  SP,BP
    MOV  DL, [SI]
(4) MOV  VARB,CL
    MOV  [DI],BX
    MOV  BUFFER,DX
    MOV  SI,VARW
(5) MOV  DS,VARW
    MOV  BUFFER,CS
(6) MOV  DS,AX
    MOV  BX,SS
```

2. 数据交换指令

其格式如下：

```
XCHG  OPRD1, OPRD2
```

功能：操作数 OPRD1 和操作数 OPRD2 里的数据互换。

例如：

```
XCHG  AX,BX
```

若(AX)＝12，(BX)＝100，执行指令后(AX)＝100，(BX)＝12。

两个操作数不能有立即数，也不能同是内存变量，且长度应该一致。可以是寄存器对寄存器，寄存器对存储器，如图 3-4 所示。

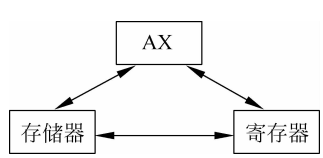

图 3-4 数据交换指令传送示意图

3. 堆栈操作指令

1）入栈指令 PUSH

其格式如下：

```
PUSH   SRC
```

功能：将源操作数 SRC 压入堆栈指针 SP 所指的存储区，堆栈指针内容减 2 或减 4。

源操作数可以是 16 位或 32 位。入栈时若源操作数是 16 位则堆栈指针减 2，是 32 位时堆栈指针减 4。

设

```
(SP) = 1200H
(AX) = 1234H
PUSH   AX
```

后

```
(SP) = 11FEH
```

数据入栈示意图如图 3-5 所示。

2）出栈指令 POP

其格式如下：

```
POP   DST
```

功能：将堆栈指针 SP 所指存储单元的内容弹到 DST，堆栈指针内容加 2 或加 4。

目的操作数可以是 16 位或 32 位。出栈时若源操作数是 16 位则堆栈指针加 2，是 32 位时堆栈指针加 4。

设

```
(SP) = 11FEH
POP   BX
```

后

```
(SP) = 1200H
(BX) = 1234H
```

数据出栈示意图如图 3-6 所示。

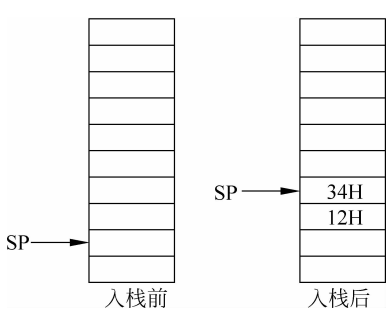

图 3-5　数据入栈

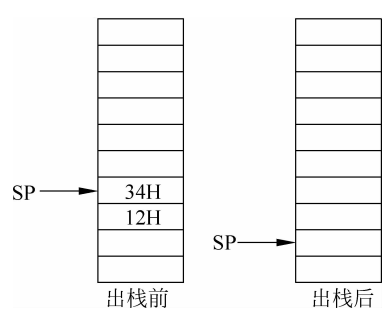

图 3-6　数据出栈

4. 知识扩展内容

前面介绍了基本的传送指令，此外还有一些指令也属于数据传送类指令，它们是 LEA、LDS、LES、IN、OUT 、XLAT、LAHF、SAHF 等。这里仅就格式作简单介绍，学生可以自行练习验证指令的功能。

(1) LEA 有效地址传送指令。

其格式如下：

```
LEA  REG,SRC
```

功能：REG ← SRC 将 SRC 的有效地址传送给寄存器 REG。该指令与“MOV REG, OFFSET SRC”的功能一样，都是将 SRC 的有效地址即偏移量送给 REG。只不过 OFFSET 伪操作符要求后面必须是个变量，不能是表达式，而 LEA 中 SRC 既可以是变量，也可以是表达式，如[BX+10]。若 BUFFER 是个内存变量，则有：

```
MOV  SI,OFFSET  BUFFER
LEA   SI,BUFFER
```

两者功能一样，均为把 BUFFER 的偏移量即有效地址送给 SI。后者书写量少，但汇编后的机器码比前者多一个字节。

(2) LDS 32 位地址传送指令，将地址指针送 REG 和 DS。

其格式如下：

```
LDS  REG,双字存储单元
```

功能：将双字存储单元的低字送寄存器 REG，高字送 DS 寄存器。

假设(DS)=2000H ,(BX)=26H,(20026H)=2300H,(20028H)=0F123H。

执行指令 LDS SI,[BX]前，有效地址 EA=(BX)=26H。

物理地址=(DS)×10H+EA=20000H+26H=20026H。

执行指令 LDS SI,[BX]后，将 20026H 单元的内容 2300H 送给 SI，将 20028H 单元的内容 0F123H 送给 DS，即(DS)=0F123H,(SI)=2300H。

(3) LES 32 位地址传送指令，将地址指针送 REG 和 ES。

其格式如下：

```
LES  REG, 双字存储单元
```

功能：将双字存储单元的低字送寄存器 REG，高字送 ES 寄存器。

假设(DS)=2000H ,(BX)=26H,(20026H)=2300H,(20028H)=0F123H。

执行指令 LES SI ,[BX]前，有效地址 EA=(BX)=26H。

物理地址=(ES)×10H+EA=20000H+26H=20026H。

执行指令 LES SI,[BX]后，将 20026H 单元的内容 2300H 送给 SI，将 20028H 单元的内容 0F123H 送给 ES，即(ES)=0F123H,(SI)=2300H。

(4) 查表转换指令 XLAT。

查表转换指令 XLAT 为处理数组、表和堆栈中的数据提供了方便。

其格式如下：

```
XLAT
```

功能：(AL)←((BX)+(AL))。

说明：BX 寄存器里存放表或数组的首地址，AL 寄存器中存放待查内容在表或数组中的位移量，将 BX 寄存器的首地址加上 AL 的位移量所得的地址中的内容送到 AL 寄存器中去。很多查表、加密解密等操作往往要用到 XLAT 指令。例如，(BX)＝1000H，(AL)＝03H，执行 XLAT 的过程如图 3-7 所示。

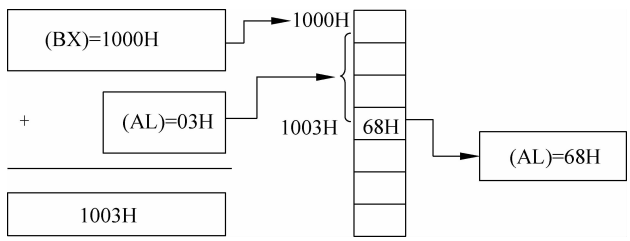

图 3-7　XLAT 操作示意

(5) 标志位传送指令 LAHF、SAHF。

其格式如下：

```
LAHF
SAHF
```

功能：LAHF 将寄存器 FLAG 的低 8 位送入 AH 寄存器，即把标志寄存器的符号标志位 SF、零标志位 ZF、辅助进位标志位 AF、奇偶标志位 PF 和进位标志位 CF 按位送到 AH 寄存器；而 SAHF 的功能与 LAHF 相反，是 AH 的内容送入标志寄存器 FLAG 的低 8 位。这样提供了对标志位的操作方法。请注意，这只是对标志寄存器的低 8 位进行操作，对高 8 位进行操作可以考虑利用堆栈操作来实现。具体方法请学生自己考虑。

(6) 输入输出指令 IN、OUT 在输入输出章节介绍。

上机题目：将数据区 AREA1 中的 26 个小写字母搬到 AREA2 数据区中去，并将其变成大写字母，同时显示。

提示：小写字母的 ASCII 码值比大写字母大 20H。要求先画出流程图，然后根据流程图编写程序。

3.1.2　算术运算类指令

当初设计计算机的目的就是要进行数据的运算处理，参加运算的数可以是二进制数、十进制数；可以是字节型(8 位)的、字型(16 位) 的和双字型 (32 位) 等有符号数和无符号数，这样就需要各种运算命令，基本围绕加、减、乘、除四种运算。

- 加法运算指令；
- 减法运算指令；
- 乘法运算指令；

- 除法运算指令；
- 调整类指令。

1. 加法指令

80x86 指令系统中常用的加法指令如下：

ADD(Add)表示加法指令。

ADC(Add With Carry)表示带进位加法指令。

INC (Increment)表示加 1 指令。

(1) ADD 加法指令。

其格式如下：

```
ADD  DST,SRC
```

功能：(DST) ←(DST)＋(SRC)。

说明：将 DST 的内容与 SRC 的内容相加，结果回送给 DST。DST 可以是存储器、寄存器，SRC 可以是存储器、寄存器和立即数。

例如：

```
ADD  AL,23                    ; (AL) = (AL) + 23
ADD  AX,BX                    ; (AX) = (AX) + (BX)
ADD  CX,COUNT                 ; (CX) = (CX) + COUNT
ADD  BYTE  PTR  [SI],2        ; (16×(DS) + (SI)) = (16×(DS) + (SI)) + 2
```

(2) ADC 带进位加法指令。

其格式如下：

```
ADC  DST,SRC
```

功能：(DST) ←(DST)＋(SRC)＋CF。

说明：将 DST 的内容、SRC 的内容及进位位 CF 内容相加，结果回送给 DST。DST 和 SRC 的规定与 ADD 指令相同。带进位的加法指令一般用在双精度数的运算。主要是当低字(或字节)相加产生进位时，高字(或字节)相加必须考虑这个进位。

例如：

```
ADC  BH,3                     ; (BH) = (BH) + 3 + CF
ADC  AX,BX                    ; (AX) = (AX) + (BX) + CF
ADC  CX,BUFFER                ; (CX) = (CX) + (16×(DS) + (BUFFER)) + CF
ADC  BYTE  PTR  [SI],2        ; (16×(DS) + (SI)) = (16×(DS) + (SI)) + 2 + CF
```

(3) INC 加 1 指令。

其格式如下：

```
INC  OPRD
```

功能：OPRD← (OPRD)＋1。

说明：将 OPRD 中的内容加 1 结果回送 OPRD。OPRD 应该是具有存储能力的寄存器或存储单元。

例如：

```
INC  SI                          ; (SI) = (SI) + 1
INC  AL                          ; (AL) = (AL) + 1
INC  BYTE  PTR [BX]              ; (16 × (DS) + (BX)) = (16 × (DS) + (BX)) + 1
INC  WORD PTR [DI]               ; (16 × (DS) + (DI), 16 × (DS) + (DI) + 1) = (16 × (DS) +
                                   (DI), 16 × (DS) + (DI) + 1)
```

2. 减法指令

80x86 指令系统中常用的减法指令如下：

SUB(Subtract)表示减法指令。

SBB(Subtract With Borrow)表示带借位减法指令。

DEC(Decrement)表示减 1 指令。

NEG(Negate)表示求补指令。

CMP(Compare)表示比较指令。

(1) SUB 减法指令。

其格式如下：

```
SUB  DST, SRC
```

功能：(DST)←(DST)－(SRC)。

说明：将 DST 的内容与 SRC 的内容相减，结果回送给 DST。DST 可以是存储器、寄存器，SRC 可以是存储器、寄存器和立即数。

例如：

```
SUB  CL, 3                       ; (CL) = (CL) - 3
SUB  AX, BX                      ; (AX) = (AX) - (BX)
SUB  BYTE  PTR  [SI], 26         ; (16 × (DS) + (SI)) = (16 × (DS) + (SI)) - 26
```

(2) SBB 带借位减法指令。

其格式如下：

```
SBB  DST, SRC
```

功能：(DST) ←(DST)－(SRC)－CF。

说明：将 DST 的内容、SRC 的内容及进位位 CF 内容相减，结果回送给 DST。DST 和 SRC 的规定与 SUB 指令相同。带进位的减法指令一般用在双精度数的运算。主要是当低字(或字节)相减产生借位时，高字(或字节)相减必须考虑这个借位。

例如：

```
SBB   CL, 3                      ; (CL) = (CL) - 3 - CF
SBB   AX, BX                     ; (AX) = (AX) - (BX) - CF
SBB   CX, BUFFER                 ; (CX) = (CX) - 16 × (DS) + (BUFFER) - CF
SBB   BYTE  PTR  [SI], 26        ; (16 × (DS) + (SI)) = (16 × (DS) + (SI)) - 26 - CF
```

(3) DEC 减 1 指令。

其格式如下：

```
DEC  OPRD
```

功能：OPRD←(OPRD)－1。

说明：将 OPRD 中的内容减 1 结果回送 OPRD。OPRD 应该是具有存储能力的寄存器或存储单元。

例如：

```
DEC  SI                          ; (SI) = (SI) - 1
DEC  CL                          ; (CL) = (CL) - 1
DEC  BYTE  PTR [BX]              ; (16 × (DS) + (BX)) = (16 × (DS) + (BX)) - 1
DEC  WORD PTR [DI]               ; (16 × (DS) + (DI)) = (16 × (DS) + (DI)) - 1
```

(4) NEG 求补指令。

其格式如下：

```
NEG  OPRD
```

功能：OPRD←0FFFFH－(OPRD)＋1。

说明：将 OPRD 取反末位加 1 结果回送 OPRD。

例如：

```
MOV  AX, - 1                     ; (AX) = 0FFFFH
NEG  AX                          ; (AX) = 0001H
MOV  BX,3                        ; (BX) = 0003H
NEG  BX                          ; (BX) = 0FFFDH
```

(5) CMP 比较指令。

其格式如下：

```
CMP  OPRD1,OPRD2
```

功能：(OPRD1)－(OPRD2)。

说明：执行该指令时，只对两个操作数作减法操作，结果并不回送，但影响标志位。该指令后面往往紧跟一条条件转移指令，来控制程序的分支转向。

图 3-8 所示是加减法运算的导向图，根据这个导向图可以很自如地写出各种加减法指令而不至于犯一般语法错误。

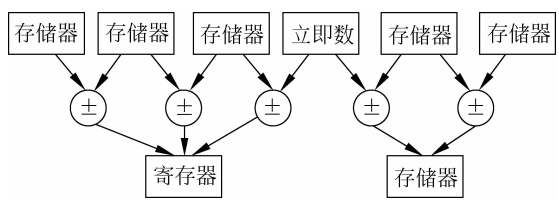

图 3-8　加减法运算导向图

例 3-3　在数据段中有三个双精度数 X、Y、Z，分别存放在 X，X＋2，Y，Y＋2，Z，Z＋2 字存储单元中。存放原则是高字在高位地址中，低字在低位地址中，试编程实现公式 W＝X＋Y＋12－Z 指定的计算。

程序代码如下：

```
DATA  SEGMENT
X  DW ?,?
Y  DW ?,?
```

```
Z   DW ?,?
W   DW ?,?,?
DATA   ENDS
CODE   SEGMENT
ASSUME   CS: CODE,DS: DATA
START:    MOV   AX, DATA
MOV   DS, AX
MOV   AX, X
MOV   DX, X + 2                    ; 取第一个双精度数
ADD   AX, Y                        ; 与第二个双精度数的低字相加
ADC   DX, Y + 2                    ; 高位字带进位加
ADD   AX, 12                       ; 低字加 12
ADC   DX, 0                        ; 高字带进位加
SUB   AX,Z                         ; 低位减 Z 的低位
SBB   DX,Z + 2                     ; 高位带借位相减
MOV   W,AX                         ; 存结果的低位
MOV   W + 2,DX                     ; 存结果的高位
MOV   AH, 4CH
INT   21H                          ; 程序结束返回
CODE   ENDS
END   START
```

例 3-4　把字节型数组 AREA1 和字节型数组 AREA2 中的数据由低位向高位相加，结果存放在 AREA2 中，两个数组均为 10 个字节。要求画出流程图。

例 3-4 的流程图如图 3-9 所示。

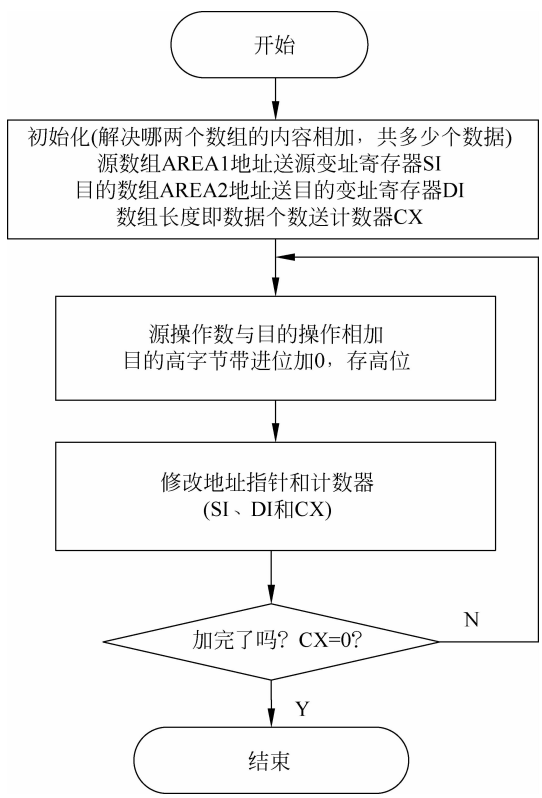

图 3-9　数据相加流程图

程序代码如下：

```
DATA  SEGMENT
AREA1  DB 10 DUP(?)
AREA2  DB 10 DUP(?)
       DB ?
DATA  ENDS
CODE   SEGMENT
     ASSUME  CS: CODE
     ASSUME  DS: DATA
START:   MOV  AX,DATA
         MOV  DS,AX
         MOV  CX,10
         LEA  SI,AREA1
         LEA  DI,AREA2
         CLC
AGAIN:   MOV  AL,[SI]
         ADC  [DI],AL
         INC  SI
         INC  DI
         DEC  CX
         JNZ  AGAIN
         ADC  [DI+1],0
         MOV  AH,4CH
         INT  21H
CODE     ENDS
         END  START
```

需要说明的是，程序内部用了一条 ADC 带进位加的指令，由于开始 CF＝0，所以第一次尽管是作带进位的加法，但实际上加的是 0。以后每次都考虑到可能有进位。最后，当循环结束时要考虑最后可能产生的进位，所以用一条“ADC ［DI＋1］，0”语句。

3. 乘法指令

80x86 指令系统的乘法指令如下：

MUL(Unsigned Multiple)表示无符号数乘法指令。

IMUL(Signed Multiple)表示带符号乘法指令。

(1) MUL 无符号数乘法指令。

其格式如下：

```
MUL  SRC
```

功能：字节操作时(AX)←(AL)＊(SRC)；字操作时(DX,AX)←(AX)＊(SRC)。

说明：目的操作数必须是累加寄存器，源操作数是除立即数以外的任一种寻址方式所得的数据。如果两数是字节型的，被乘数默认在 AL 中，结果存于 AX；如果是字型的，被乘数默认在 AX 中，结果高字存于 DX，而低字存于 AX。

(2) IMUL 带符号乘法指令。

其格式如下：

```
IMUL  SRC
```

功能：功能与说明同 MUL 一样，只是运算数是带符号数，而 MUL 是无符号数乘法。

4. 除法指令

80x86 指令系统的除法指令如下：

DIV(Unsigned Divide)表示无符号除法指令。

IDIV(Signed Divide)表示带符号除法指令。

CBW(Convert Byte to Word)表示将字节转换成字指令。

CWD(Convert Word to Double Word)表示将字转换成双字指令。

(1) DIV 无符号除法指令。

其格式如下：

```
DIV  SRC
```

功能：字节操作时，商(AL) ←(AX)/ (SRC)　余数(AH) ←(AX)/(SRC)。

目的操作数必须是累加寄存器，源操作数是除立即数以外的任一种寻址方式所得的数据。16 位被除数在 AX 中，源操作数为 8 位，结果 8 位商数在 AL 中，余数也是 8 位在 AH 中。32 位被除数在 DX，AX 中，源操作数为 16 位，结果 16 位商数在 AX 中，余数也是 16 位在 DX 中。

(2) IDIV 有符号除法指令。

其格式如下：

```
IDIV  SRC
```

功能：字节操作时，商(AL) ←(AX)/ (SRC)　余数(AH) ←(AX)/(SRC)。

执行的操作与 DIV 一样，只是参加运算的数均为带符号数。

(3) CBW 字节转换成字指令，实际上是符号扩展指令，主要是为了使运算数的位数满足运算要求。

其格式如下：

```
CBW
```

功能：将 AL 的符号位扩展到 AH，若(AL)的最高位为 0，则(AH)＝00H；若(AL)的最高位为 1，则(AH)＝0FFH。

如(AL)＝12，使用 CBW 指令后，(AH)＝0；如(AL)＝－1，使用 CBW 指令后，(AH)＝0FFH。

(4) CWD 字转换成双字指令，实际上也是符号扩展指令，主要也是为了使运算数的位数满足运算要求。

其格式如下：

```
CWD
```

功能：将 AX 的符号位扩展到 DX，若(AX)的最高位为 0，则(DX)＝00H；若(AX)的最高位为 1，则(DX)＝0FFFFH。

CBW 和 CWD 均不影响标志位。

例 3-5 四则运算综合题：数据段有 4 个带符号的字变量 NUM1，NUM2，NUM3，NUM4，外加 RESULT 双字单元存放结果。要求完成按(NUM1－(NUM2 * NUM3＋NUM4)－120)/NUM2 规定的运算，结果存于 RESULT 双字单元。这是一个典型的顺序结构，因此只需要按运算符的级别进行了算术运算即可。

程序代码如下：

```
DATA   SEGMENT
NUM1    DW  ?
NUM2    DW  ?
NUM3    DW  ?
NUM4    DW  ?
DATA    ENDS
CODE    SEGMENT
          ASSUME  CS: CODE,DS: DATA
START:    MOV  AX,DATA
          MOV  DS,AX
          MOV  AX,NUM2
          IMUL NUM3                    ; 完成 NUM2 * NUM3
          MOV  CX,AX
          MOV  BX,DX
          MOV  AX,NUM4
          CWD
          ADD  CX,AX
          ADC  BX,DX                   ; 完成乘积加 NUM4
          SUB  CX,120
          SBB  BX,0                    ; 减立即数
          MOV  AX,NUM1
          CWD
          SUB  AX,CX
          SBB  DX,BX                   ; NUM1 减小括号里的结果
          IDIV NUM2                    ; 除以 NUM2
          MOV  RESULT,AX
          MOV  RESULT + 2,DX           ; 存结果
          MOV  AH,4CH
          INT  21H
CODE      ENDS
          END  START
```

5. 调整指令

80x86 指令系统的调整指令如下：

DAA(Decimal Adjust For Addition)表示加法的十进制调整指令。

DAS(Decimal Adjust For Addition)表示减法的十进制调整指令。

AAA(ASCII Adjust For Addition)表示加法的 ASCII 码调整指令。

AAS(ASCII Adjust For Subtracition)表示减法的 ASCII 码调整指令。

AAM(ASCII Adjust For Multiplicaition)表示乘法的 ASCII 码调整指令。

AAD(ASCII Adjust For Division)表示除法的 ASCII 码调整指令。

3.1.3　逻辑运算类和移位指令

1. 逻辑运算类指令

80x86 指令系统的逻辑运算类指令如下：

AND(and)表示逻辑与(逻辑乘)。

OR(or)表示逻辑或(逻辑加)。

NOT(not)表示逻辑非。

XOR(exclusive or)表示异或。

TEST(test)表示测试。

(1) AND 逻辑与。

其格式如下：

```
AND   DST,SRC
```

功能：DST←(DST)∧(SRC)。

说明：目的操作数(DST)与源操作数(SRC)内容相与，即作逻辑乘(按位乘)结果回送 DST。利用逻辑与命令可以屏蔽某些位，也可以保留某些位。如本例子中的高 6 位被保留，而最后两位被屏蔽。逻辑与操作在检测、控制等方面都有应用。

例如：

```
MOV   AL,0BFH
AND   AL,0FCH
```

指令的执行如下：

$$\begin{array}{r} 0BFH \\ \wedge\ 0FCH \end{array} \Rightarrow \begin{array}{r} 10111111B \\ \wedge\ 11111100B \\ \hline 10111100B \end{array}$$

(2) OR 逻辑或。

其格式如下：

```
OR    DST,SRC
```

功能：DST←(DST) ∨(SRC)。

说明：目的操作数(DST)与源操作数(SRC)内容相或，即作逻辑加(按位加)结果回送 DST。利用逻辑或命令可以用 1 强制某些位为 1，突出某些位。逻辑或操作在加密解密和控制等方面都有应用。

例如：

```
MOV   AL,0B0H
OR     AL,8CH
```

指令的执行如下：

$$\begin{array}{r} 0B0H \\ \vee\ \ 8CH \end{array} \Rightarrow \begin{array}{r} 10110000B \\ \vee\ 10001100B \\ \hline 10111100B \end{array}$$

(3) NOT 逻辑非。

其格式如下：

```
NOT  OPRD
```

功能：(OPRD)的内容取反回送 OPRD。

例如：

```
MOV  AL,43H
NOT  AL                              ; 01000011B→10111100B
```

说明：操作数 OPRD 内容取反后，再回送操作数 OPRD，是非常有用的逻辑操作。

(4) XOR 异或。

其格式如下：

```
XOR   DST,SRC
```

功能：DST←(DST)⊕(SRC)。

说明：异或实质上是按位加，没有进位，原则是 0 异或任何数都等于任何数，而 1 异或任何数，都等于任何数的非，即 1 异或 1 得 0，而 1 异或 0 等于 1。特别强调自身异或结果为零，这一点在动画等方面应用较多。

例如：

```
MOV  AL,0B0H
XOR  AL,8CH
```

$$\begin{array}{r} 0B0H \\ \oplus\ \ 8CH \end{array} \Rightarrow \begin{array}{r} 10110000B \\ \oplus\ 10001100B \\ \hline 00111100B \end{array}$$

(5) TEST 测试。

其格式如下：

```
TEST   OPRD1,OPRD2
```

功能：(OPRD1) ∧(OPRD2) 结果不回送，只影响标志位。

说明：利用 TEST 命令可以测试某些位是 1 还是 0，而这些位的信息往往用来代表某些设备的状态，据此可以了解外部设备的状态，以便对其有目的地发号施令，进行相应的控制操作。

例如：

```
IN     AL,61H
TEST  AL,80H
```

指令的执行如下：

$$\begin{array}{r} 61H \\ \wedge\ 80H \\ \hline \end{array} \Rightarrow \begin{array}{r} 01100001B \\ \wedge\ 10000000B \\ \hline 00000000B \end{array}$$

要测试哪些位，将测试数据的相应位设置为1，其他位设置为0。例如上例中要测试AL中的最高位，所以测试数据的最高位为1，其余各位为0，测试数据为80H。

2. 移位类指令

80x86指令系统的移位指令如下：

(1) SHL逻辑左移和SAL算术左移。

其格式如下：

```
SHL   OPRD,n
```

功能：n=1，OPRD左移一位，最高位移入CF进位标志位，末位补零；n>1，OPRD移位位数由CL内容决定(CL里存放移位位数n)，在不溢出的情况下，每左移一位，相当于被移动数乘2，左移n位相当于被移动数乘2^n，逻辑及算术左移示意图如图3-10所示。

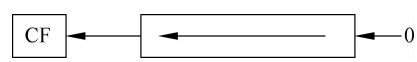

图3-10　逻辑及算术左移

需要说明的是，算术左移与逻辑左移完成相同的操作，但在DEBUG中不允许有SAL算术左移指令出现，所以一般情况下只用逻辑左移。

例如：

```
MOV   AL,34H
SHL    AL,1
```

00110100B左移一位后结果为01101000B。(CF)=0。

例如：

```
MOV   CL,2
SHL    AL,CL
```

00110100B左移两位后结果为11010000 B，(CF)=0。

(2) SHR逻辑右移。

其格式如下：

```
SHR   OPRD,n
```

功能：n=1，OPRD右移一位，最高位补零，末位移入CF进位标志位；n>1时，OPRD移位位数由CL内容决定，在不溢出的情况下，每右移一位，相当于被移动数除2，右移n位相当于被移动数除2^n，逻辑右移示意图如图3-11示。

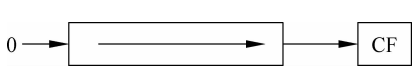

图3-11　逻辑右移

例如：

```
MOV   AL,34H
SHR    AL,1
```

00110100B 右移一位后结果为 00011010B。(CF)=0。

例如：

```
MOV  CL,2
SHR   AL,CL
```

00110100 右移两位后结果为 00001101。(CF)=0。

(3) SAR 算术右移。

其格式如下：

```
SAR  OPRD,n
```

功能：n=1,OPRD 右移一位,最高位补数的符号,末位移入 CF 进位标志位；n>1,OPRD 移位位数由 CL 内容决定,每右移一位,相当于被移动数除 2,右移 n 位相当于被移动数除 2^n,算术右移示意图如图 3-12 所示。

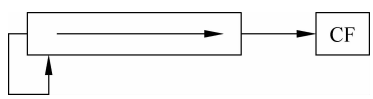

图 3-12 算术右移

例如：

```
MOV  AL,34H
SAR   AL,1
```

00110100 右移一位后结果为 00011010。(CF)=0。

例如：

```
MOV  AL,94H
SAR   AL,1
```

10010100 右移一位后结果为 11001010。(CF)=0。

算术右移的特点是保留符号位不变,原来是正数算术移位后依然是正数；原来是负数,算术移位后还是负数,亦即数的属性不变。

(4) ROL 循环左移。

其格式如下：

```
ROL  OPRD,n
```

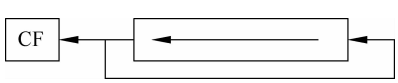

图 3-13 循环左移

功能：n=1,OPRD 左移一位,最高位移入末位和 CF 进位标志位；n>1,OPRD 左循环移位位数由 CL 内容决定。左循环移位 ROL 亦称小循环,循环左移示意图如图 3-13 所示。

例如：

```
MOV  AL,83H
ROL   AL,1
```

移位前：10000011。移位后：00000111。(CF)=1。

又如：

```
MOV  AL,83H
MOV  CL,3
```

```
ROL   AL,CL
```

移位前：10000011。移位后：00011100。(CF)＝0。

(5) ROR 循环右移。

其格式如下：

```
ROR  OPRD,n
```

功能：n＝1，右移一位，末位移入最高位和 CF 进位标志位；n＞1，右循环位数由 CL 内容决定。右循环移位 ROR 亦称小循环，循环右移示意图如图 3-14 所示。

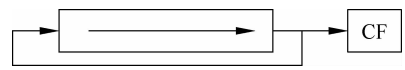

图 3-14　循环右移

例如：

```
MOV  AL,83H
ROR   AL,1
```

移位前：10000011。移位后：11000001。(CF)＝1。

又如：

```
MOV  AL,83H
MOV  CL,3
ROL   AL,CL
```

移位前：10000011。移位后：01110000。(CF)＝0。

(6) RCL 带进位的循环左移。

其格式如下：

```
RCL  OPRD,n
```

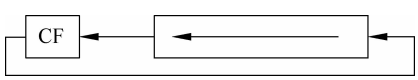

图 3-15　带进位的循环左移

功能：n＝1，左循环移一位，最高位进 CF，CF 进末位；n＞1，左循环移位次数由 CL 内容决定，带进位的循环左移示意图如图 3-15 所示。

例如：

```
MOV  AL,83H
RCL   AL,1
```

移位前：10000011。移位后：00000110。(CF)＝1。

```
MOV  CL,2
RCL   AL,CL
```

移位前：00000110。移位后：00011010。(CF)＝0。

(7) RCR 带进位的循环右移。

其格式如下：

```
RCR  OPRD,n
```

功能：n＝1，右循环移一位，CF 进最高位，末位进 CF；n＞1，右循环移位次数由 CL 内

容决定,带进位的循环右移示意图如图 3-16 所示。

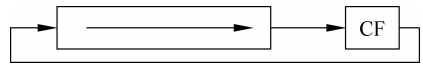

图 3-16 带进位的循环右移

例如:

```
MOV  AL,83H
RCR   AL,1
```

移位前:10000011。移位后:01000001。(CF)=1。

```
MOV  CL,2
RCR   AL,CL
```

移位前:01000001。移位后:11010000。(CF)=0。

3.1.4 字符串操作类指令

1. MOVS 串传送

其格式如下:

```
MOVS   DST,SRC
MOVSB
MOVSW
```

功能:

ES:[DI]←(DS:[SI])。

SI←(SI)±1/±2。

DI←(DI)±1/±2。

说明:SI 开始时指向源串首址,DI 开始时指向目的串首址,串操作指令 MOVS 把(SI)所指单元的数据传送到(DI)所指的单元中去,可以是一个字节,也可以是一个字;同时根据方向标志和数据格式(字或字节)来修改 SI 的 DI 指针值,指向下一个待传送数据。若是字节传送 SI 和 DI 的内容±1,若是字传送 SI 和 DI 的内容±2,其中当 DF=0 时作加运算,当 DF=1 时作减运算。

当该指令与 REP 联用时,可以将数据段中的整串数据连续地传送到辅加段目的串中去,传送的个数由 CX 寄存器的值控制。

例 3-6 试将数据段中 STR1 串的字符搬到辅加段的 STR2 串变量里去。

程序代码如下:

```
DATA  SEGMENT
    Str1  db 'This  is  a  personal  computer!'
    Leth  equ  $ - str1
DATA  ENDS
EXTRA  SEGMENT
    Str2  db  leth dup(' ')
EXTRA  ENDS
CODE  SEGMENT
      ASSUME  CS: CODE,DS: DATA,ES: EXTRA
    START:  MOV  AX,DATA
            MOV  DS,AX
```

```
            MOV  AX ,EXTRA
            MOV  ES,AX
            MOV  CX,leth
            MOV  SI,OFFSET  str1
            MOV  DI,OFFSET  str2
            CLD
            REP  MOVSB
            MOV  AH,4CH
            INT  21H
CODE  ENDS
      END  START
```

2. CMPS 串比较

格式如下：

```
CMPS  DST,SRC
CMPSB
CMPSW
```

功能：(SI)－(DI)，即将 SI 指向的数据段中的一个字节或一个字，与 DI 所指向的附加段中的一个字节或一个字作减操作，但结果不回送，只根据比较的结果改变标志位。

说明：

(1) SI 指向源串的首地址，DI 指向目的串的首地址，CX 寄存器指出比较的次数。

(2) 与 REPZ/REPE 配合使用时，表明结果为 0 或比较结果相同时继续，直到 CX 内容为零。换句话说比较相同则继续，比较不同或 CX 内容为零时结束。

(3) 与 REPNZ/REPNE 配合使用时，表明结果不为 0 或比较结果不相同时继续，直到 CX 内容为零。换句话说比较不同则继续，比较相同或 CX 内容为零时结束。

例 3-7　比较两个字符串是否相同，相同时输出 Y，不同时输出 N。

程序代码如下：

```
DATA  SEGMENT
    Str1  db 'This  is  a  personal  computer!'
    Leth  equ  $ - str1
    Str2  db'This  is  not  a  personal  computer!'
DATA  ENDS
CODE  SEGMENT
      ASSUME  CS: CODE,DS: DATA,ES: DATA
    START:  MOV  AX,DATA
            MOV  DS,AX
            MOV  ES,AX
            MOV  CX,LETH
            MOV  SI,OFFSET  str1
            MOV  DI,OFFSET  str2
            CLD
            REPE CMPSB
            JZ   EQUAL
            MOV  DL,'N'
```

```
            JMP  PRINT
    EQUAL:  MOV  DL,'y'
    PRINT:  MOV  AH,2
            INT  21H
            MOV  AH,4CH
INT   21H
CODE  ENDS
      END START
```

3. SCAS 串扫描、串搜索

其格式如下：

```
SCAS  DST
SCASB
SCASW
```

功能：(AL)－(ES：(DI))　字节操作，(DI) ±1 回送 DI。

(AX)－(ES：(DI))　字操作，(DI) ±2 回送 DI。

功能就是在目的串里找与(AL)/(AX)内容相同/不同的字节或字。

说明：目的串在附加段里，偏移地址由 DI 内容指出，搜索次数由 CX 内容决定。

例 3-8　在数组 Array 中查找第一个－1，找到后将存放－1 的地址保存到内存变量 Address 中，并正常退出；若没找到，则输出“Not　found.”结束。

程序代码如下：

```
DATA  SEGMENT
    array  db  1,2,3,4,5,-1,8,12,-2,-23
    count    equ  $ -array
    address  dw  ?
    message  db  'Not  found. $ '
DATA  ENDS
CODE  SEGMENT
      ASSUME  CS: CODE,ES: DATA
    START:  MOV  AX, DATA
            MOV  ES,AX
            MOV  DI,OFFSET  array
            MOV  AL, -1
            MOV  CL,count
            CLD
            REPNZ  SCANSB
            JNE  NOTFOUND
            SUB  DI,1
                 ; 当搜索成功时,地址指针已经指向下一个字单元,故(DI) -1
            MOV  ADDRESS,DI
            JMP  EXIT
    NOTFOUND:
            MOV  DX,OFFSET  message
            MOV  AH,9
            INT  21H
```

```
    EXIT:   MOV  AH,4CH
            INT  21H
CODE  ENDS
      END  START
```

4. STOS 存入串指令

其格式如下：

```
STOS  DST
STOSB
STOSW
```

功能：将 AL 内容存入(ES：(DI))所指字节单元，(DI) ±1 回送 DI。

将 AX 内容存入(ES：(DI)) 所指字单元，(DI) ±2 回送 DI。

例 3-9　将数组 array 清零后结束。

程序代码如下：

```
DATA  SEGMENT
    array  db  1,2,3,4,5,-1,8,12,-2,-23
    count   equ  $ -array
DATA  ENDS
CODE  SEGMENT
      ASSUME  CS: CODE,ES: DATA
    START:  MOV  AX, DATA
            MOV  ES,AX
            MOV  DI,OFFSET  array
            MOV  AL,0
            MOV  CL,count
            CLD
            REP  STOSB
            MOV  AH,4CH
            INT  21H
CODE  ENDS
      END  START
```

5. LODS 装入串指令

其格式如下：

```
LODS  SRC
LODSB
LODSW
```

功能：将(DS：(SI))所指字节单元内容存入 AL，(SI) ±1 回送 SI。

将(DS：(SI)) 所指字单元内容存入 AX，(SI) ±2 回送 SI。

功能：该指令将数据段中源变址寄存器 SI 所指单元的内容送到 AL 或 AX 中，并根据方向标志修改 SI 的内容，字节操作时(SI) ±1 回送 SI，字操作时(SI) ±2 回送 SI。通常情况下该指令一般不与重复前缀配合使用，因为若中间没有其他操作，重复的结果只能是取最

后的数据送 AL 或 AX。但该指令可以用于对数组或串中的数据作单个测试比较。

REP(Repeat)表示重复。

REPE/REPZ (Repeat While Equal/Zero)表示相等/为零则重复。

REPNE/REPNZ (Repeat While Not Equal/Not Zero)表示不相等/不为零则重复。

LOOP(Loop)表示循环。

LOOPZ/LOOPE(Loop Whole Zero,Or Equal)表示当为零/相等时循环。

LOOPNZ/LOOPNE(Loop While Not Zero,Or Not Equal)表示当不为零/不相等时循环。

3.1.5 控制转移类指令

JZ/JE(Jump If Zero Or Eual)表示结果为零/相等则转移。

JNZ/JNE(Jump If Not Zero Or Not Eual)表示结果不为零(或不相等)则转移。

JS/JNS(Jump If Sign Or Not Sign)表示结果为负则转移/结果为正则转移。

JO/JNO(Jump If Overflow Or Not Overflow)表示溢出则转移/不溢出则转移。

JP/JPE(Jump If Parity Or Parity Even)表示奇偶位为 1 则转移。

JNP(Jump If Not Parity Or Parity Odd)表示奇偶位为零则转移。

JB/JNAE,JC(Jump If Below,Or Not Above Or Equal,Or Carry)表示低于,或不高于,或等于,或进位为 1 则转移。

JNB/JAE,JNC(Jump If Not Below,Or Above Or Equal,Or Not Carry)表示不低于,或高于等于,或进位为零则转移。

JBE/JNA(Jump If Below Or Equal,Or Not Above) 表示低于或等于,或不高于则转移。

JNBE/JA(Jump If Not Below Or Equal,Or Jump Above)表示不低于或等于,或高于则转移。

JL/JNGE(Jump If Less,Or Not Greater,Or Equal)表示小于或不大于或等于则转移。

JNL/JGE(Jump If Not Less,Or Greater,Or Equal)表示小于或等于,或大于等于则转移。

JLE/JNG(Jump If Less Or Equal,Or Not Greater)表示小于或等于,或不大于则转移。

JNLE/JG(Jump If Not Less Or Equal,Or Greater)表示不小于或等于,或者大于则转移。

JCXZ(Jump If CX Register Is Zero)表示当 CX 寄存器的内容为零则转移。

3.1.6 调整类指令

前面介绍的算术运算指令都是针对二进制而言的,但在实际工作中人们还是常常使用十进制。为了方便十进制运算,计算机提供了一组十进制调整指令,在二进制运算的基础上,给予十进制调整,得到十进制的结果。这里的十进制就是 BCD(Binary Coded Decimal)码,即二—十进制数。它是用 4 位二进制数表示一位十进制数。因为 4 位二进制有 16 种组合,取最自然的一组 8421 码,其对应关系如表 3-1 所示。

表 3-1　BCD 码与相应十进制数对照表

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

BCD 码又有非压缩的和压缩 BCD 之分。其中非压缩的 BCD 码就是其 ASCII 码，8 位二进制，高 4 位为 0011，没有数值意义。压缩的 BCD 码是 4 位二进制数。例如，2041D(Decimal)表示十进制数，用 BCD 码表示则为 0010 0000 0100 0001。由于 BCD 码有两种格式，所以调整指令也有两种。

1. 压缩 BCD 码调整指令

DAA(Decimal Adjust For Aiddtion)表示加法的十进制调整指令。

DAS(Decimal Adjust For Subtraction)表示减法的十进制调整指令。

由于压缩的 BCD 码是一个字节有两位十进制数，单纯用算术运算指令计算所得的数据必须经过调整才能正确。对于加法运算，当两个 BCD 码相加的结果在 1010～1111 之间或者两数相加向高位有进位时，可加 6 调整成正确值。

例如：

```
    8              1000
  + 7      ⇨     + 0111
   15             1111
                + 0110
                10101
```

从上式可以看出，BCD 码第一次的结果 1111 不是 8421 的 BCD 码，加 6 调整后为 0001 0101 才是正确的 BCD 码结果。

再如：

```
    1                1
    9             1001
  + 9      ⇨     + 1001
   19             0011
                10110
                11001
```

这种情况是低位向上位有进位，第一次加得结果向高位有进位，根据调整原则加上 6 并保留进位值可得到正确结果 00011001 即十进制的 19。

(1) 加法调指令 DAA。

其格式如下：

```
DAA
```

说明：在调整之前，必须执行 ADD 或 ADC 指令，且必须是对两个压缩的 BCD 码相加，结果存到 AL 寄存器中。

调整方法如下：

如果 AF 标志位(辅助进位位)为 1,或 AL 寄存器的低 4 位在 1010～1111 之间,则 AL 内容加 06H 调整,并将 AF 置 1。

如果 CF 标志位(进位位)为 1,或 AL 寄存器的高 4 位在 1010～1111 之间,则 AL 内容加 60H 调整,并将 CF 置 1。

例 3-10 设执行指令前(AL)＝28H,(BL)＝68H。

执行指令

```
ADD  AL,BL 后(AL) = 90H  CF = 0  AF = 1
```

用调整指令

```
DAA
(AL) = 96H
```

结果正确。

$$\begin{array}{r} 00101000 \\ +\ 01101000 \\ \hline 10010000 \\ +\ 00000110 \\ \hline 10010110 \end{array}$$

(2) 减法调指令 DAS。

其格式如下：

```
DAS
```

说明：在调整之前,必须执行 SUB 或 SBB 指令,且必须是对两个压缩的 BCD 码相减,结果存到 AL 寄存器中。

调整方法如下：

如果 AF 标志位(辅助进位位)为 1,或 AL 寄存器的低 4 位在 1010～1111 之间,则 AL 内容减 06H 调整,并将 AF 置 1。

如果 CF 标志位(进位位)为 1,或者 AL 寄存器的高 4 位在 1010～1111 之间,则 AL 内容减 60H 调整,并将 CF 置 1。

例 3-11 设执行指令前(AL)＝86H,(AH)＝07H。执行指令 SUB AL,BL 后(AL)＝7FH,CF＝0,AF＝1,用调整指令 DAS(AL)＝79H,结果正确。

$$\begin{array}{r} 10000110 \\ -\ 00000111 \\ \hline 01111111 \\ -\ 00000110 \\ \hline 01111001 \end{array}$$

2. 非压缩的 BCD 码调整指令

AAA(ASCII Adjust For Addition)表示加法的 ASCII 码调整指令。

AAS(ASCII Adjust For Subtraction)表示减法的 ASCII 码调整指令。

AAM(ASCII Adjust For Multiplication)表示乘法的 ASCII 码调整指令。

AAD(ASCII Adjust For Division)表示除法的 ASCII 码调整指令。

这是一组 ASCII 码的调整指令,也适用于非压缩的 BCD 码调整。

例 3-12　设执行指令前(AX)=0535H,(BL)=39H,实际上 AL 和 BL 里的内容分别为 ASCII 码 5 和 9。

执行指令　ADD AL,BL 后(AL)=6EH,AF=0。

用调整指令 AAA(AX)=0604H　AF=1　CF=1,结果正确。

$$\begin{array}{r} 00110101 \\ +\ 00111001 \\ \hline 01101110 \\ +\ 00000110 \\ \hline 01100100 \end{array}$$

(1) AAA 加法的 ASCII 码调整指令。

其格式如下:

```
AAA
```

说明:这条指令执行前必须执行加法指令 ADD 或带进位的加法指令 ADC,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 寄存器中。

执行的操作如下:

把 AL 中的和调整成非压缩的 BCD 格式后送 AL 寄存器;而把调整时可能产生的进位加到 AH 寄存器中。

AAA 调整步骤如下:

① 如果 AL 寄存器中低 4 位在 0～9 之间,且 AF 为 0,不调整,直接进入③。

② 如果 AL 寄存器中低 4 位在 A～F 之间或 AF 为 1,则 AL 内容加 6,AH 寄存器内容加 1,并将 AF 置 1。

③ 清除 AL 寄存器的高 4 位。

④ AF 的值送 CF。

(2) AAS 减法的 ASCII 码调整指令。

其格式如下:

```
AAS
```

说明:这条指令执行前必须执行减法指令 SUB 或带借位的减法指令 SBB,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 寄存器中。

执行的操作如下:

把 AL 中的差调整成非压缩的 BCD 格式后送 AL 寄存器;而把调整时可能产生的进位从 AH 寄存器中减去。

AAS 调整步骤如下:

① 如果 AL 寄存器中低 4 位在 0～9 之间,且 AF 为 0,不调整,直接进入③。

② 如果 AL 寄存器中低 4 位在 A～F 之间或 AF 为 1,则 AL 内容减 6,AH 寄存器内容减 1,并将 AF 置 1。

③ 清除 AL 寄存器的高 4 位。

④ AF 的值送 CF。

例 3-13 计算 25+48−19。

程序代码如下：

```
DATA  SEGMENT
    U1   DB   05H,02H
    U2   DB   08H,04H
    U3   DB   09H,01H
DATA  ENDS
CODE   SEGMENT
       ASSUME  CS: CODE,DS: DATA
    START:  MOV  AX,DATA
            MOV  DS,AX                  ; 数据段赋值
            MOV  AX,0                   ; 清零
            MOV  AL,U1
            ADD  AL,U2                  ; 低位相加(AL) = 0DH,13
            AAA                         ; 调整(AL) = 03H CF = AF = 1
            MOV  DL,AL                  ; 低位和存 DL (DL) = 3
            MOV  AL,U1 + 1
            ADC  AL,U2 + 1              ; 高位带进位加(AL) = 07H
            AAA                         ; 调整(AL) = 07H CF = AF = 1
            XCHG AL,DL                  ; AL 和 DL 交换数据
            SUB  AL,U3                  ; 减低位(AL) = FAH
            AAS                         ; 调整(AL) = 4
            XCHG AL,DL                  ; AL 和 DL 交换数据
            SBB  AL,U3 + 1              ; 作带借位减法(AL) = 5
            AAS                         ; 调整(AL) = 05H CF = AF = 0
            MOV  DH,L                   ; 结果(DX) = 0504H
            MOV  AH,4CH
            INT  21H                    ; 结束退出
CODE   ENDS
            END  START                  ; 结果为 54 正确
```

(3) AAM 乘法的 ASCII 调整指令。

其格式如下：

```
AAM
```

说明：这条指令执行前必须执行 MUL 指令把两个非压缩的 BCD 码相乘(此时要求其高 4 位为零),结果存放在 AL 寄存器中。调整方法是把 AL 寄存器中的内容除以 0AH,商在 AH 中,余数在 AL 中。

执行的操作如下：

例 3-14 设执行指令前(AL)=07H,(BL)=09H。

执行如下指令：

```
MUL  BL                                 ; (AL) = 3FH
AAM                                     ; (AH) = 06H  (AL) = 03H
```

(4) AAD 除法的 ASCII 码调整指令。

其格式如下：

```
AAD
```

说明：前面介绍的 AAA、AAS、AAM 调整指令都是作相应的加减乘运算后再作的调整。除法与之不同，除法调整指令是针对下列情况设立的。

如果被除数是存在 AX 寄存器的非压缩 BCD 码，AH 中存放十位数，AL 中存放个位数，而且要求 AH 和 AL 的高 4 位都是 0。除数是一位非压缩的 BCD 码，高位也是 0。在把两个数用 DIV 作除之前，必须先用 AAD 指令把 AX 中的被除数调整成二进制数，并存放在 AL 寄存器中。因很少用这条指令，故只作个简单介绍。

3.1.7　处理器控制类指令

在处理器控制类指令中，主要介绍标志位处理指令，这些指令只影响本指令指定的标志，而不影响其他标志。主要有 7 条指令，它们是进位标志处理指令 CLC、CMC 和 STC，方向标志处理指令 CLD 和 STD，中断标志处理指令 CLI 和 STI。

1. CLC 清除进位位/使进位位为 0

其格式如下：

```
CLC
```

功能：使 CF＝0。

2. CMC 进位标志取反指令

其格式如下：

```
CMC
```

功能：使 CF 取反回送 CF。

3. STC　置进位标志指令

其格式如下：

```
STC
```

功能：使 CF＝1。

4. CLD 清方向标志指令

其格式如下：

```
CLD
```

功能：使 DF＝0。

其主要作用是作串/数组处理时，使地址指针按增加方向变化。

5. STD 置方向标志指令

其格式如下：

```
STD
```

功能：使 DF＝1。
其主要作用是作串/数组处理时，使地址指针按减少方向变化。

6. CLI 清中断指令

其格式如下：

```
CLI
```

功能：使 IF＝0。
其作用是关掉中断允许触发器，使 CPU 不接受外部设备中断。

7. STI 开中断指令

其格式如下：

```
STI
```

功能：使 IF＝1。
其作用是开中断允许触发器，使 CPU 接受外部设备中断。

3.2 寻 址 方 式

由于数据和指令在计算机中都是按地址存储的，所谓寻址就是对指令和对数据进行寻找地址。寻址方式就是寻找地址的计算方法。寻址分为指令寻址和数据寻址。

3.2.1 数据寻址

对数据寻址有很多种方式，概括起来大致可分为 7 种：立即寻址、寄存器寻址、直接寻址、寄存器间接寻址、寄存器相对间址寻址、基址加变址寻址、相对基址加变址寻址。其中，立即寻址、寄存器寻址不访问存储器，故速度最快。其余 5 种寻址方式都需要访问存储器，速度稍慢。而且既然要访问存储器就要涉及数据段、辅加段甚至其他段，因此除默认状态外，应该考虑对段的说明，即指令中有效地址 EA(Effective Address)与哪个段寄存器发生关系。需要声明一点，寻址方式既可以对目的操作数 DST，也可以对源操作数 SRC，但这里我们所谈的操作数寻址方式是指源操作数的寻址方式。

1. 立即寻址方式

其格式如下：

```
OPRD  DST,imm
```

功能：　DST ←（DST）　OPRD imm。

说明：目的操作数 DST 与立即数 imm 完成操作码 OPRD 所规定的操作，结果回送给目的操作数 DST。这里的目的操作数 DST 必须是具有存储能力的寄存器或存储器。

例如，MOV　AH,30H 即是将立即数 30H 送给 AH 寄存器，它的机器码是 B430H，操作数就在指令本身，不需要访问存储器，故称为立即寻址。

又如，ADD　BX,89H 其机器码为 81C38900H 等。执行结果如图 3-17 所示。

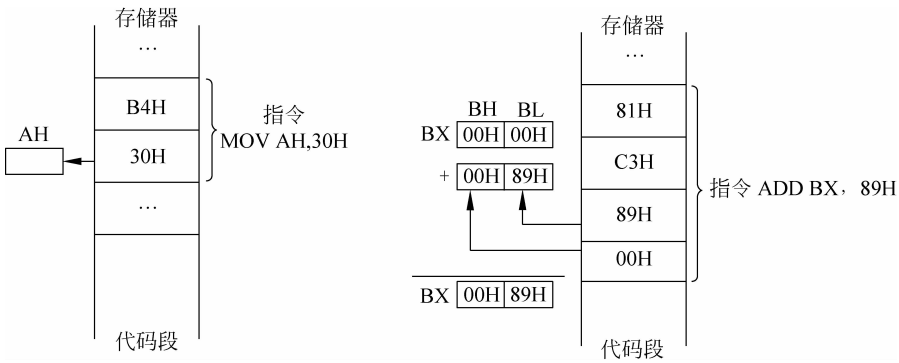

图 3-17　立即寻址方式

立即寻址方式主要用来为寄存器或存储器的单元赋初值，如系统功能调用时的功能号就是利用立即方式给出的。例如，MOV　AH,2 就是 2 号系统功能调用的语句。很多循环初值等都用立即方式赋值，这是最快捷的寻址方式。

2. 寄存器寻址方式

其格式如下：

```
OPRD　DST,SRC
```

功能：DST ←（DST）　OPRD　（SRC）。

说明：源操作数如果是寄存器，则为寄存器寻址。其实就是指操作数在寄存器中，要让这个数与目的操作数 DST 中的数据完成操作码 OPRD 所规定的操作，结果回送到目的操作数 DST 中。操作中只需要注意数据类型一致就可以了，即 DST 和 SRC 应同时为 8 位、16 位或 32 位等。

例如：

```
MOV　AX,BX
```

若执行前 AX 内容为 0，而 BX 的内容为 1234H，执行上述指令后，AX 的内容变为 1234H。若在前条指令的基础上，再执行 ADD　AL,BH，则 AL 的内容为 68H。

DST 也可以是存储单元，如 MOV　[2800H],AX 则是把 AX 寄存器的数据送到数据段 2800H 字单元中。在存储器寻址方式中，如操作数在存储器中，必须通过访问存储器才能找到操作数，需要计算有效地址 EA。所谓的有效地址 EA 就是相对于段基值的偏移量。最简单的存储器寻址方式就是直接给出有效地址 EA。

3. 直接寻址方式

其格式如下：

```
OPRD  REG,mm
```

功能：REG←（REG） OPRD mm。

说明：目的操作数 REG 寄存器中的数据与存储器 mm 单元中的数据完成操作码 OPRD 所规定的操作，结果回送给目的操作数 REG。源操作数的有效地址 EA 直接写在指令中，它就是个地址变量，可以是变量名字的形式，或是用方括号括起来的数字。但 REG 与 mm 必须注意数据类型保持一致。

操作数的物理地址＝(DS)＊10H＋EA

例如：

```
MOV AX,DS:[2800H]
```

设(DS)＝3000H，EA＝2800H，[32800H]＝56H，[32801H]＝88H，执行该条指令的结果就是将数据段 2800H 字单元的内容 8856H 送到 AX 寄存器中去。这里的 DS 可以省略，写成 MOV AX，[2800H]，系统默认是从数据段的 2800H 单元取出数据送到 AX 寄存器中去。但切不可将上句指令与 MOV AX，2800H 弄混淆，前者是从存储单元[32800H]中取数；后者是立即寻址，是把立即数 2800H 送到 AX 寄存器中。前者的操作码为 A10028H，执行的结果将 32800H 字单元的内容 8856H 送给 AX 寄存器，如图 3-18 所示；后者的机器码为 B80028H，执行的结果是把立即数 2800H 送给 AX 寄存器。

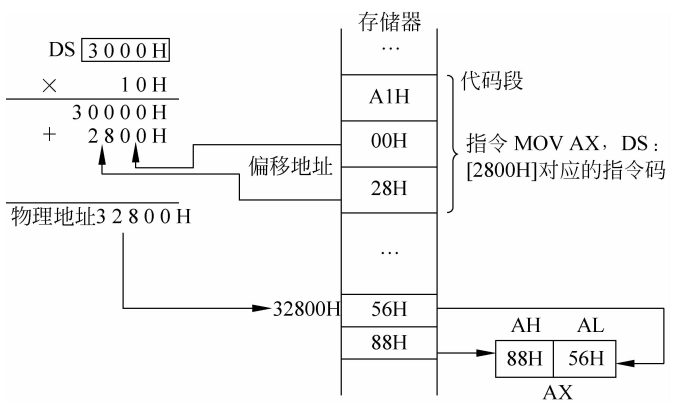

图 3-18 直接寻址方式

直接寻址方式的有效地址 EA 还可以在其他段中（如在辅加段 ES、代码 CS 或堆栈段 SS 中），这时一定要标明段前缀，即说明是段超越的直接寻址。其实段超越就是超越段。

例如，MOV AX，ES：[2800H] 即属于段超越的直接存取方式。

很多时候有效地址 EA 以变量名的形式提供，如 BUF 是一个字数据变量，MOV AX，BUF 执行的就是直接寻址方式，这里的 BUF 就是直接地址。若 BUF 中存的内容为 5678H，执行指令后，直接把 BUF 里的数据 5678H 送给 AX 寄存器。

直接寻址无疑是访问存储器寻址中最直接、最明确的寻址方式，对于单个数据很实用。

但是当遇到数组、字符串及表类操作时就不方便了。如例 3-1 的数据搬家，如果采取直接寻址方式，逐次从数组 AREA1 中取一个数据，送到数组 AREA2，就会出现一个很长的顺序结构的程序：

```
MOV   AL,AREA1
MOV   AREA2,AL
MOV   AL,AREA1 + 1
MOV   AREA2 + 1,AL
⋮
```

这是最简单的顺序结构，虽然直观明确，但当数据多时会使程序很冗长，占存储空间大。如果采取其他寻址方式，如寄存器间接寻址，则可大简化程序设计量，使程序简捷占存储空间小。如例 3-1 所示。所以引入了寄存器间接寻址等多种方式。

4. 寄存器间接寻址

其格式如下：

```
OPRD   DST,[BX/BP/SI/DI]
```

功能：根据指令给出的间址寄存器[BX/BP/SI/DI]计算出有效地址 EA，然后将 EA 所指单元里的内容与目的操作数 DST 的内容作操作码 OPRD 规定的操作，结果回送到目的操作数 DST。

说明：目的操作数 DST 只能是寄存器。

在有些书中把含有[BX]、[BP]的形式称为基址寻址，而把含有[SI]、[DI]的形式称为变址寻址，实际上由于有效地址 EA 是从寄存器间接给出的，所以可统一称其为间接寻址；在实方式中只允许 BX、BP、SI、DI 四个寄存器作间址寄存器。

数据段操作数物理地址＝(DS) * 10H＋(BX/SI/DI)

堆栈段操作数物理地址＝(SS) * 10H＋(BP)

例如：

```
MOV   AX,[BX]
```

设(DS)＝3000H，(BX)＝2800H，[32800H]＝56H，[32801H]＝88H，执行该条指令的结果就是将数据段 2800H 字单元的内容 8856H 送到 AX 寄存器中去，如图 3-19 所示。这里的 DS 是省略的，实际上可写成 MOV AX,DS:[BX]。

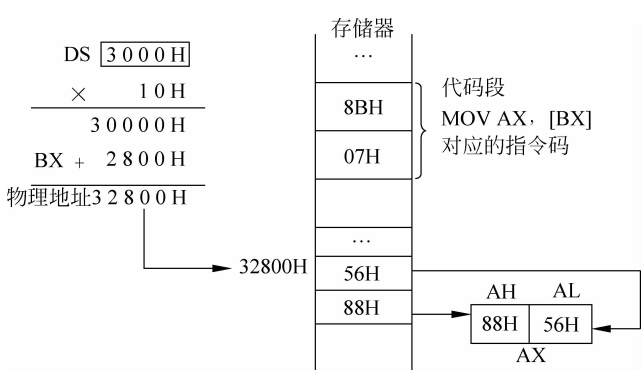

图 3-19　寄存器间接寻址方式

在例 3-1 中利用寄存器间接寻址方式解决了直接寻址方式的不便，但如果遇到下面的问题，单纯的寄存器间接寻址也显得不方便。

例 3-15 在数据段中有三个数组 X、Y、Z，现要求计算数组 X 与数组 Y 对位加，结果回送给数组 Z。这时用寄存器相对寻址就非常容易。

流程图如图 3-20 所示。

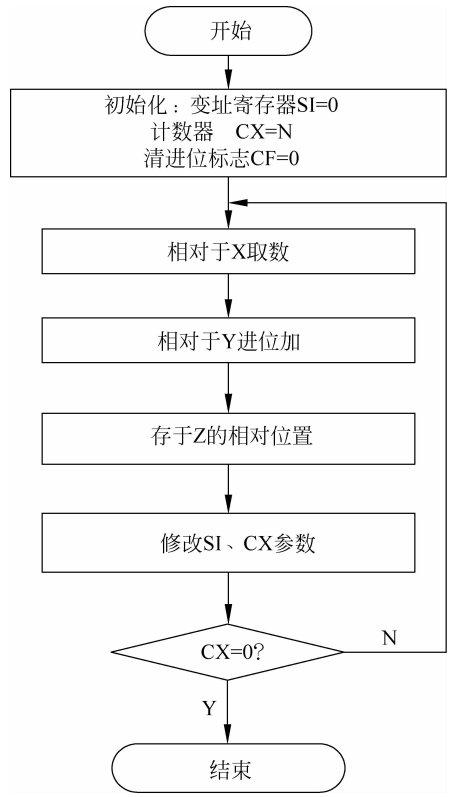

图 3-20 寄存器相对寻址实例流程图

程序代码如下：

```
DATA   SEGMENT
X   DB 11,22,33,44,55,66,77,88,99
N   equ   $ - X
Y   DB 1,2,3,4,5,6,7,8,9
Z   DB   N + 1   dup(0)
DATA   ENDS
CODE   SEGMENT
ASSUME   CS: CODE,DS: DATA
START: MOV   AX,DATA
       MOV   DS,AX
       MOV   SI,0
       MOV   CX,N
       CLC
AGAIN: MOV   AL, X[SI]
       ADC   AL,Y[SI]
       MOV   Z[SI],AL
       INC   SI
       DEC   CX
       JNZ   AGAIN
       ADC   Z[SI + N],0
       MOV   AX,4C00H
       INT   21H
CODE   ENDS
       END   START
```

这就引入了寄存器的相对寻址。

5. 寄存器相对寻址方式

其格式如下：

```
OPRD   DST,偏移量[BX/BP/SI/DI]
```

功能：DST←(DST)　OPRD（[BX/BP/SI/DI＋偏移量]）。

说明：根据指令中的偏移量与间址寄存器[BX/BP/SI/DI]计算出有效地址 EA，然后将 EA 所指单元里的内容与目的操作数 DST 的内容作操作码 OPRD 规定的操作，结果回送到目的操作数 DST。

① 目的操作数 DST 只能是寄存器。

② 偏移量是个 8 位或 16 位的数。

在实方式中只允许 BX、BP、SI、DI 四个寄存器作间址寄存器。

数据段操作数物理地址＝(DS)＊10H＋(BX/SI/DI)＋ 偏移量

堆栈段操作数物理地址＝(SS)＊10H＋(BP)＋ 偏移量

例如：

```
MOV  AX,COUNT[BP]
```

设(SS)＝4000H,(BP)＝2800H,[42900H]＝56H,[42901H]＝88H,COUNT＝100H 执行该指令的结果就是将堆栈段 2900H 字单元的内容 8856H 送到 AX 寄存器中去,如图 3-21 所示。该指令也可以写成 MOV AX,[BP ＋COUNT]。

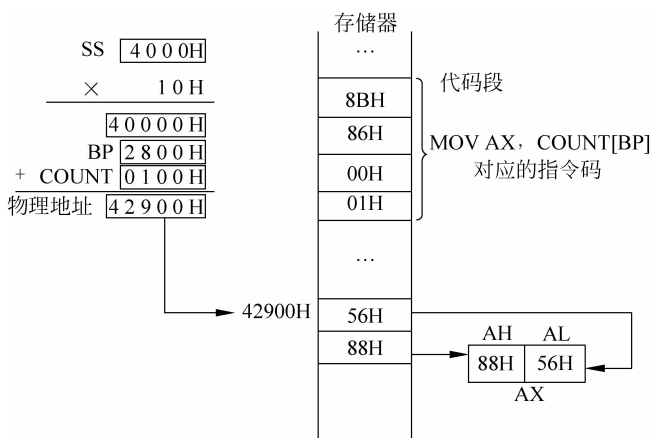

图 3-21　寄存器相对寻址方式

注意：寄存器间接寻址和寄存器相对寻址实质是一回事,可以把寄存器间接寻址看成是偏移量为 0 的寄存器相对寻址。

6. 基址加变址寻址

其格式如下：

```
OPRD  DST,[BX/BP][SI/DI]
```

功能：DST←(DST)　OPRD　([BX/BP][SI/DI])。

根据指令给出的基址寄存器[BX/BP]与变址寄存器[SI/DI]计算出有效地址 EA,然后将 EA 所指单元里的内容与目的操作数 DST 的内容作操作码 OPRD 规定的操作,结果回送到目的操作数 DST。

说明：

① 目的操作数 DST 只能是寄存器。

② BX、BP 为基址寄存器,SI、DI 为变址寄存器且两者不能联用。

操作数物理地址＝(DS)＊10H＋(BX/BP)＋(SI/DI)

例如：

```
MOV  AX,[BP][SI]
```

设(SS)＝4000H,(BP)＝2800H,(SI)＝100H,[42900H]＝56H,[42901H]＝88H,执行该指令就是将堆栈段 2900H 字单元的内容 8856H 送到 AX 寄存器中去,如图 3-22 所示。该指令也可以写成 MOV AX,[BP＋SI]。

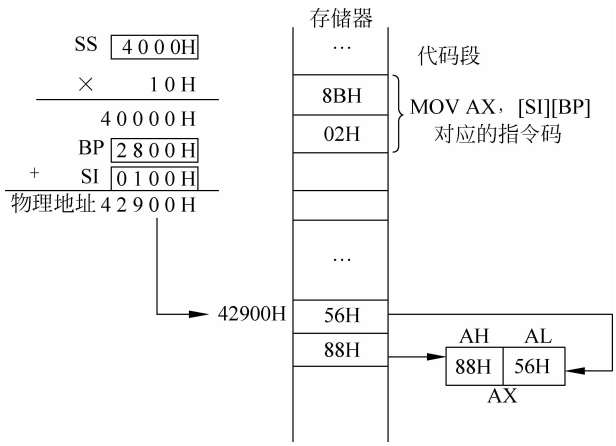

图 3-22 基址加变址寻址方式

如果在例 3-15 中把数组 X 的地址作为基准地址,用基址寄存器 BX 的内容表示,那么数组 Y 的地址变为[BX＋N],数组 Z 的地址为[BX＋2N],这样由寄存器相对寻址所表示的问题就可以用基址加变址和相对基址加变址来实现了。基本代码如下:

寄存器相对寻址方式:

```
AGAIN: MOV  AL, X[SI]
       ADC  AL,Y[SI]
       MOV  Z[SI],AL
       INC  SI
       DEC  CX
       JNZ  AGAIN
```

基址加变址和相对基址加变址:

```
AGAIN: MOC  AL, [BX][SI]
       ADC  AL,N [BX][SI]
       MOV  2 * N[BX][SI],AL
       INC  SI
       DEC  CX
       JNZ  AGAIN
```

7. 相对址加变址

其格式如下:

```
OPRD  DST,偏移量[BX/BP][SI/DI]
```

功能:DST←(DST)　OPRD　(偏移量[BX/BP][SI/DI])。

根据指令给出的基址寄存器[BX/BP]、变址寄存器[SI/DI]及偏移量计算出有效地址EA，然后将EA所指单元里的内容与目的操作数DST的内容作操作码OPRD规定的操作，结果回送到目的操作数DST。

说明：

① 目的操作数DST只能是寄存器。

② 偏移量是个8位或16位的数。

③ BX、BP为基址寄存器，SI、DI为变址寄存器且两者不能联用。

操作数物理地址＝(DS)＊10H＋(BX/BP)＋(SI/DI)＋ 偏移量

例如：

```
MOV  AX,COUNT[BP][SI]
```

设(SS)＝4000H，(BP)＝2600H，(SI)＝200H，[42900H]＝56H，[42901H]＝88H，COUNT＝100H，执行该指令就是将堆栈段2900H字单元的内容8856H送到AX寄存器中去，如图3-23所示。

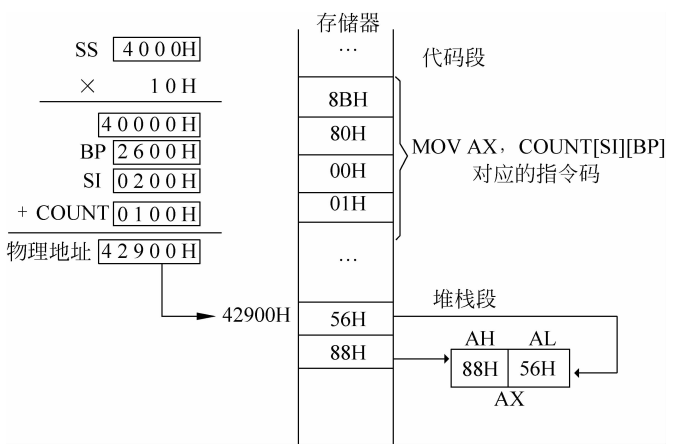

图3-23　相对基址加变址寻址方式

该指令也可以写成MOV AX,COUNT [BP＋SI]或MOV AX,[BP＋SI＋COUNT]。

3.2.2　指令寻址

程序也是按地址存放的，所以取指令也有寻址问题。实际上计算机执行指令时，是根据代码段寄存器CS和指令指针寄存器IP里内容所指的地址寻址的。程序开始运行时系统把程序的首地址送给CS:IP，以后基本上是顺序存取，也就是指令指针内容顺次加1，找到下条指令地址，取指令、分析指令和执行指令，只有遇到分支时，才根据条件发生转移。而转移也只不过是把新的方向地址送CS:IP，以后又顺序地取指令、分析指令和执行指令。程序在执行时大体有三种分支，一是无条件转移，用JMP指令实现；二是条件转移，用JCC指令实现，其中CC是执行上一条指令产生的条件或者说是机器的状态；三是调用子程序，用CALL和RET指令实现。

1. 无条件转移

其格式如下：

```
JMP  标号
JMP  SHORT  标号
JMP  NEAR  PTR 标号
JMP  FAR  PTR  标号
```

功能：将标号地址送 CS:IP，使程序从标号地址继续执行。

说明：无条件转移有两种形式，即段内转移和段间转移。如果是段内直接转移只修改 IP；而段间直接转移需要送 32 位地址，修改 CS:IP 值。通常情况下都是段内直接转移，为简便起见只写 JMP 标号形式，不需要 SHORT 和 NEAR 属性说明。

对于段内可以细分为段内直接短转移、段内直接近转移和段内间接转移。

(1) 段内转移。

① 段内直接短转移。

格式如下：

```
JMP  SHORT  标号
```

功能：(IP)←(IP)＋8 位标号位移量。

说明：这个位移量范围为＋127～－128，256 字节的范围。

例如：

```
        ⋮
JMP  SHORT  NEXT
        ⋮
NEXT:
```

② 段内直接近转移。

格式如下：

```
JMP  NEAR  PTR  标号
```

功能：(IP)←(IP)＋16 位标号位移量。

说明：这个位移量范围为＋32767～－32768，65536 字节的范围。

例如：

```
        ⋮
        JMP  NEAR  PTR  NEXT
        ⋮
NEXT:
```

③ 段内间接转移。

格式如下：

```
JMP  WORD  PTR  标号
```

功能：(IP)←(EA)。

说明：有效地址 EA 的值由标号的寻址方式决定。如果用 16 位寄存器，则把寄存器的内容送 IP，如是一个字存储单元，则把存储单元的内容送 IP。

(2) 段间转移。

① 段间直接远转移。

格式如下：

```
JMP  FAR  PTR  标号
```

功能：IP←标号所在段的段内偏移量，CS←标号所在段的段地址。

例如：

```
CODE1  SEGMENT
         ⋮
        JMP  FAR  PTR  NEXTPROGRAMM
         ⋮
CODE1  ENDS
CODE2  SEGMENT
         ⋮
    NEXTPROGRAMM:
         ⋮
CODE2  ENDS
```

这里 CODE1 和 CODE2 是两个独立的段，CODE1 中的无条件转移指令直接转移到 CODE2，故是段间直接转移，是远转移。

② 段间间接转移。

格式如下：

```
JMP  DWORD  PTR  标号
```

功能：(IP)←(EA)，(CS)←(EA+2)。

说明：有效地址 EA 的值由标号的寻址方式决定，可以使用除立即数和寄存器方式以外的任何存储器寻址方式，即根据寻址方式求出 EA 后，把指定存储单元的字内容送 IP，同时把下一个存储字的内容送 CS。这样就实现了段间转移。

2. 条件转移

无条件转移是人为的强制性的分支，而条件转移虽然也是人为的，但却是根据指令执行的具体情况随机变化的，也就是说是有条件的。这个条件就是条件转移指令前面的指令所改变的机器状态，这个状态在标志位里反映出来。条件转移或者说条件分支可分为四种情况：根据条件标志转移指令，用于无符号数的转移指令，用于有符号数的转移指令，根据 CX (或 ECX)内容转移指令。

在程序的执行过程中，如果指令的运行改变了标志寄存器的内容，可以根据单个标志改变程序运行走向。

格式如下：

JCC 标号

功能：根据条件控制程序转向标号所指的指令继续运行程序。

根据单个标志的条件转移指令如表 3-2 所示。

表 3-2 根据单个标志的条件转移指令

助记符	功能	测试条件
JZ/JE	结果为零或相等时转移 Jump If Zero Or Equal	ZF＝1
JNZ/JNE	结果不为零或不相等时转移 Jump If Not Zero Or Not Equal	ZF＝0
JC	有进位或借位时转移 Jump If Carry	CF＝1
JNC	没有进位或借位时转移 Jump If Not Carry	CF＝0
JS	结果为负时转移 Jump If Sign	SF＝1
JNS	结果不为负时转移 Jump If Not Sign	SF＝0
JO	结果有溢出时转移 Jump If Overflow	OF＝1
JNO	结果无溢出时转移 Jump If Not Overflow	OF＝0
JP/JPE	结果数含偶数个 1 时转移 Jump If Parity,Or Parity Even	PF＝1
JNP/JPO	结果数含奇数个 1 时转移 Jump If Parity,Or Parity Odd	PF＝0

无符号数的转移指令如表 3-3 所示。

表 3-3 用于无符号数的条件转移指令(设 A、B 为两个无符号数)

助记符	功能	测试条件
JA/JNBE	高于/不低于且不等于(即 A＞B)时转移	CF＝0 且 ZF＝0
	Jump If Above Or Not Below Or Not Equal	CF ∨ ZF＝0
JAE/JNB/JNC	高于等于/不低于(即 A＞＝B)时转移	CF＝0 或 ZF＝1
	Jump If Above Or Equal Or Not Below	CF ∨ ZF＝1
JB/JNAE/JC	低于/不高于且不等于(即 A＜B)时转移	CF＝1 且 ZF＝0
	Jump If Below Or Not Above Or Not Equal Or If Carry	CF ∨ ZF＝1
JBE/JNA	低于等于/不高于(即 A＝＜B)时转移	CF＝1 或 ZF＝1
	Jump If Below Or Equal Or Not Above	CF ∨ ZF＝1

用于有符号数的转移指令如表 3-4 所示。

表 3-4　用于有符号数的条件转移指令(设 A、B 为两个有符号数)

助记符	功　能	测试条件
JG/JNLE	大于/不小于且不等于(即 A>B)时转移 Jump If Greater Or Not Less Or Not Equal	SF=OF 且 ZF=0
JGE/JNL	大于等于/不小于(即 A>=B)时转移 Jump If Greater Or Equal Or Not Less	SF=OF 且 ZF=1
JL/JNGE	小于/不大于且不等于(即 A<B)时转移 Jump If Less Or Not Greater Or Equal	SF≠OF 且 ZF=0
JLE/JNG	小于等于/不大于(即 A=<B)时转移 Jump If Less Or Not Greater	SF≠OF 且 ZF=1

根据 CX(或 ECX)内容转移的指令如表 3-5 所示。

表 3-5　根据 CX(或 ECX)内容转移的条件转移指令

助记符	功　能	目的操作数
JCXZ	CX 寄存器的内容为 0 时转移	
JECXZ	ECX 寄存器的内容为 0 时转移	只能为短标号

条件转移指令通常情况下常与 CMP、TEST、DEC、INC 等指令连用。也就是说通过比较、通过测试、通过修改某些参数来判断程序是否应该转移。

例如：

```
        CMP   AL,12
        JE    EQUAL
        ⋮
EQUAL:
        ⋮
```

或

```
        TEST CL,10
        JNE  LOP
        ⋮
LOP:
        ⋮
```

或

```
        ⋮
AGAIN:
        ⋮
        DEC  CX
        JNZ  AGAIN
        ⋮
```

3. 子程序调用与返回

改变程序顺序的另一种方式是子程序调用。程序在执行过程中，如果遇到了子程序调

用语句 CALL <子程序>,则无条件地放弃下条指令语句,而转向所指子程序的执行工作,如图 3-24 所示。

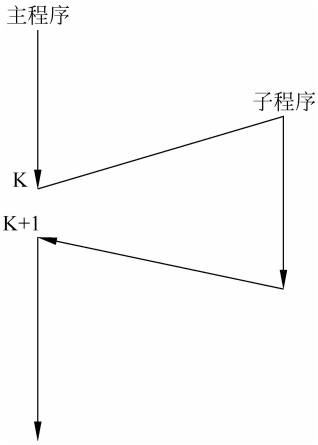

图 3-24 子程序调用

图中主程序正常执行到第 K 条指令时,接下来本应执行第 K+1 条指令,但由于第 K 条指令是子程序调用命令,所以改变了程序执行的顺序,把子程序的首地址送给了程序计数器 PC。当子程序执行完毕,又由 RET 指令返回到主程序的断点 K+1 处继续原程序的工作。关于子程序的内容准备在子程序章专门讨论。

习 题 3

3.1 什么是指令?什么是指令系统?简述 8088/8086 的指令系统中的几类指令。

3.2 什么是寻址?什么是寻址方式?举例说明 8086/8088 微处理器的数据寻址有哪几种方式及其基本特征是什么?

3.3 根据数据传送导向图,每种情况写出 2 条指令来。

3.4 假定(DI)=1000H,(SI)=007FH,(BX)=0040H,(BP)=0016H,变量 TABLE 的偏移地址为 0100H。试指出下列指令的源操作数字段的寻址方式,它的有效地址(EA)和物理地址(PA)分别是多少?

(1) MOV AX,[1234H] (2) MOV AX,TABLE

(3) MOV AX,[BX+100H] (4) MOV AX,TABLE[BP][SI]

3.5 若 BUFFER 为数据段单元的符号名,其中存放的内容为 1234H,试问下列两条指令有什么区别?执行完指令后,AX 寄存器的内容是什么?

```
MOV  AX,BUFFER
LEA  AX,BUFFER
```

3.6 指出下列指令的错误。

(1) MOV CL,DX (2) SHL AX,5

(3) MOV DS,1234H (4) MOV ES,DS

(5) SUB　AL,1200H　　(6) MOV　CS,AX

(7) MOV　AX,BX+DI　　(8) SUB　AX,[BX+CX]

(9) CMP　123,[SI]　　(10) ADD　AX,CL

3.7　指出下列指令的错误。

(1) POP　　CS　　(2) SUB　[SI],[DI]

(3) PUSH　AH　　(4) ADC　AX,DS

(5) XCHG　[SI],278H　　(6) DIV　12

(7) IN　AL,378H　　(8) OUT　DX,AL

(9) PUSH　CH　　(10) MUL　AL,CL

3.8　请分别用一条汇编语言指令完成如下功能。

(1) 把 BX 寄存器和 DX 寄存器的内容相加,结果存入 DX 寄存器。

(2) 用寄存器 BX 和 SI 的基址变址寻址方式把存储器的一个字节与 AL 寄存器的内容相加,并把结果送到 AL 中。

(3) 用 BX 和位移量 0B2H 的寄存器相对寻址方式把存储器中的一个字和 CX 寄存器的内容相加,并把结果送回存储器中。

(4) 把数 0A0H 与 AL 寄存器的内容相加,并把结果送回 AL 中。

3.9　求出以下十六进制数与 0B800H 之和,并根据结果设置标志位 SF、ZF、CF 和 OF 的值。

(1) 1234H　　(2) 5678H　　(3) 0AF50H　　(4) 9B7EH

3.10　执行指令 ADD　AL,72H 前,(AL)=8EH,标志寄存器的状态标志 OF、SF、ZF、AF、PF 和 CF 全为 0,指出该指令执行后标志寄存器的值。

3.11　为下列程序段每条指令加注释,标明每条指令执行后各目的寄存器的内容。

```
MOV  AX,1234H
MOV  CL,4
ROL   AX,CL
DEC   AX
MOV  CX,4
MUL  CX
```

3.12　编程序段,将 AL、BL、CL、DL 相加,结果存在 DX 中。

3.13　编程序段,从 AX 中减去 DI,SI 和 BP 中的数据,结果送 BX。

3.14　假设(BX)=0E3H,变量 VALUE 中存放的内容为 79H,确定下列各指令单独执行后的结果。

(1) OR　BX,VALUE

(2) AND　BX,VALUE

(3) XOR　BX,0FFH

(4) AND　BX,01H

(5) TEST　BX,05H

(6) XOR　BX,VALUE

3.15　已知数据段 500H～600H 处存放了一个字符串,说明下列程序段执行后的

结果。

```
MOV  SI,600H
MOV  DI,601H
MOV  AX,DS
MOV  ES,AX
MOV  CX,256
STD
REP  MOVSB
```

3.16 为每条指令加注释,并说明程序段的功能。

```
CLD
MOV  AX,0FEFH
MOV  CX, 5
MOV  BX, 3000H
MOV  ES, BX
MOV  DI,2000H
REP   STOSW
```

3.17 写出下列逻辑地址表示的物理地址是多少?

```
4017: 000AH,
4015: 002AH
4010: 007AH
```

3.18 试根据以下要求写出相应的汇编指令序列。

(1) 求 5678H－1234H 的差,要求结果存放在 DX 寄存器中。

(2) 求 2345＋34 的和,要求结果存放在 CX 寄存器中。

(3) 把地址为 2000：8F3EH 的内存单元字节值赋给 CL。

(4) 把地址为 2000：8F3EH 的内存单元字值赋给 DI。

3.19 完成下列选择题

(1)下面哪个寄存器能做地址寄存器?

① DX　② BX　③ AX　④ CX

(2) 已知 X＝11001010,Y＝01011101,下面哪种操作使其结果为 11011111?

① X OR Y　② NOT X　③ X AND Y　④ X XOR Y

(3) 哪种操作是将 BX 的内容清零?

① OR BX,BX　② MOV BX,'0'

③ AND BX,BX　④ SUB BX,BX

(4) 交换指令 XCHG 的正确操作是哪种?

① XCHG SI,DI　② XCHG SI,0A68H

③ XCHG 1234H,[SI]　④ XCHG [SI],[DI]

(5) 将字变量 AREA 的偏移地址送入寄存器 SI 的错误指令是哪种?

① LEA SI,AREA　② MOV SI,AREA

③ MOV SI,OFFSET AREA

(6) 将立即数 0B800H 送数据段寄存器 ES 的指令组如下，试问哪组不对？

① MOV　AX,0B800H
　MOV　ES,AX

② MOV　BX,0B800H
　MOV　ES,BX

③ MOV　DX,0B800H
　MOV　ES, DX

④ MOV ES,0B800H

(7) 伪指令 VAR　DW　3596 中变量 VAR 的类型是哪种？

① 字节型　② 字型　③ 字符型　④ 双字型

(8) 下面哪个寄存器不可以作指针寄存器？

① DX　② BX　③ AX　④ SI

3.20　设 AL＝89H，则执行指令 CBW 后，寄存器 AX 的值为多少？

3.21　设有一段内存如图 3-25 所示，请在以下每条指令的右边写出执行指令后的目标操作数的值。

```
MOV  AX,1000H
MOV  DS,AX
MOV  AL,DS:[20A0H]
MOV  AX,DS:[20A0H]
MOV  DX,[20A1H]
MOV  CX,20A1H
MOV  AX,CX
MOV  DH,AH
```

1000H：20A0H	12H
1000H：20A1H	34H
1000H：20A2H	56H
1000H：20A3H	78H
1000H：20A4H	9AH
1000H：20A5H	0BH

图 3-25　3.21 题的存储区情况

3.22　试根据以下要求写出相应的汇编指令序列。

(1) 求 5678H－1234H 的差，要求结果存放在 DX 寄存器中。

(2) 求 2345＋34 的和，要求结果存放在 CX 寄存器中。

(3) 把地址为 2000：8F3EH 的内存单元字节值赋给 CL。

(4) 把地址为 2000：8F3EH 的内存单元字值赋给 DI。

(5) 写出字节变量 NUM1 和 NUM2 的内容互换的指令组。

3.23　写出执行下列指令序列后 BX 寄存器的内容。执行前(BX)＝6B16H。

```
MOV  CL,3
SHR   BX,CL
```

3.24　字符串 STRING 保存有 100 个字节的 ASCII 字符，试编写一个程序统计该字符串中空格的个数，空格的 ASCII 码值为 20H。

3.25　已知内存首地址为 ARRAY 的字节数组中，存放的数据有正有负。试编写一段程序，将数组中的数据按正负属性分开，其中正数放在以 PLUS 为首地址的区域中，负数放在以 MINUS 为首地址的区域中。