第3章 程序设计的基本技术

在汇编语言程序中，最常见的形式有顺序程序、分支程序、循环程序和子程序。这几种程序的设计方法是汇编程序设计的基础。本章将结合实例详细地介绍这些程序的设计技术以及第2章尚未讲述的指令。如前所述，本书有关汇编语言程序设计的讨论只限于DOS环境下(MASM 5.0)的实地址方式，而在实地址方式下，一个逻辑段的空间最大为64KB，因此在实地址方式下最好采用16位寻址的控制转移。本章介绍的控制转移类指令，包括转移指令、子程序的调用指令和返回指令在实现转移或调用时都要修改IP或EIP，在实地址方式下的EIP就是16位的IP，所以关于这两类指令就只限于修改IP的介绍。

3.1 顺序程序设计

顺序程序是最简单的程序，它的执行顺序和程序中指令的排列顺序完全一致。下面先介绍乘除法指令及BCD运算的调整指令。

3.1.1 乘除法指令

乘除法指令应该有无符号数乘除法指令和符号数乘除法指令之分。这是因为乘除法不同于加减法，无符号数的乘法和除法指令对符号数进行乘除运算不能得到正确的结果。如用无符号数的乘法运算做FFH乘以FFH结果为FE01H。把它们看作无符号数为255×255＝65 025(FE01H＝65 025)，其结果是正确的；若把它们看作符号数(一般情况下，都将符号数看作补码数)为(－1)×(－1)＝－511(FE01H＝－511)，显然是错误的。因此符号数必须用专用的乘除法指令。

1. 乘法指令MUL和符号整数乘法指令IMUL

指令格式

```
MUL source
IMUL source
```

其中，源操作数source可以是字节、字或者双字，与其对应的目的操作数是AL、AX或EAX。源操作数只能是寄存器和存储器，不能为立即数。在乘法指令之前必须将另一个乘数送AL(字节乘)、AX(字乘)或者EAX(双字乘)。乘法指令所执行的操作是AL、AX或者EAX乘source，乘积放回到AX、DX和AX或者EDX和EAX，如图3-1所示。

如用乘法指令实现例2.4(将AX中小于255大于0的三位BCD数转换为二进制数，存入字节变量SB中)的程序段如下。

```
MOV CH,10
MOV CL,4
MOV SB,AL          ; 暂存十位和个位
MOV AL,AH
MUL CH             ; 百位×10
MOV AH,SB
SHR AH,CL          ; 取十位
ADD AL,AH          ; 百位×10+十位
MUL CH             ; (百位×10+十位)×10
AND SB,0FH         ; 取个位
ADD SB,AL          ; (百位×10+十位)×10+个位
```

图 3-1　乘法指令的操作

乘法指令对除 CF 和 OF 以外的状态标志位无定义(注意：无定义和不影响不同,无定义是指指令执行后这些状态标志位的状态不确定,而不影响则是指该指令的执行不影响状态标志位,因而状态标志应保持原状态不变)。对于 MUL 指令,如果乘积的高一半为 0(即字节操作的 AH=0、字操作时 DX=0 或双字操作时 EDX=0),则 CF 和 OF 均为 0；否则 CF 和 OF 均为 1。对于 IMUL 指令,如果乘积的高一半是低一半的符号扩展,则 CF 和 OF 均为 0,否则 CF 和 OF 均为 1。

除了 8086 微处理器外,符号整数乘法指令 IMUL 还有双操作数指令和三操作数指令,其格式及其功能如下。

```
IMUL REG, source              ; REG ← REG × source
IMUL REG, source, imm         ; REG ← source × imm
```

双操作数乘法指令的意义是用源操作数乘目的操作数,乘积存入目的操作数。目的操作数只能是 16 位和 32 位的寄存器,源操作数可以是寄存器和存储器,但其类型要与目的操作数一致。若目的操作数是 16 位的寄存器,则源操作数还可以是立即数。

三操作数乘法指令的意义是用源操作数乘立即数,乘积存入目的操作数。目的操作数只能是 16 位和 32 位的寄存器,源操作数可以是寄存器和存储器,但其类型要与目的操作数一致。

2. 除法指令 DIV 和符号整数除法指令 IDIV

指令格式

```
DIV source
IDIV source
```

其中,源操作数 source 可以是字节、字或者双字,可为寄存器或存储器操作数,不能为立即

数。目的操作数是AX、DX和AX或者EDX和EAX。

除法指令所执行的操作是用指令中指定的源操作数source除AX中的16位二进制数或DX和AX中的32位二进制数或者EDX和EAX中的64位二进制数,被除数是AX还是DX和AX或者EDX和EAX,由源操作数是字节还是字或者双字确定。商放入AL、AX或者EAX,余数放入AH、DX或者EDX,如图3-2所示。

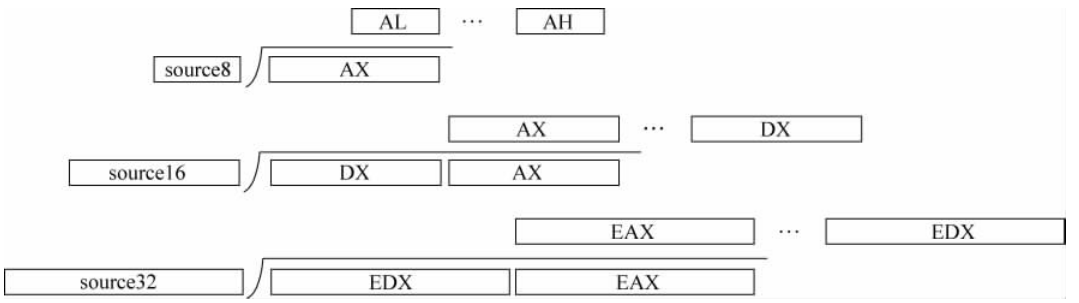

图3-2 除法指令的操作

可用除法运算将二进制数转换为BCD数。如把AL中的8位无符号二进制数转换为BCD数放入AX中的程序段如下。

```
MOV CL,10
MOV AH,0            ; 8位二进制数扩展为16位二进制数
DIV CL
MOV CH,AH           ; 暂存BCD数个位
MOV AH,0
DIV CL
MOV CL,4
SHL AH,CL           ; BCD数十位移至高4位
OR CH,AH            ; BCD数十位与个位拼合
MOV AH,0
MOV CL,10
DIV CL              ; AH中的余数为BCD数百位
MOV AL,CH           ; BCD数十位与个位送AL
```

用除10取余法将8位二进制数FFH转换为BCD数255H的二进制运算如图3-3所示。

```
      00011001          00000010          00000000
1010 / 11111111   1010 / 00011001   1010 / 00000010
       1010              1010              0000
       1011               101                10
       1010
        1111
        1010
         101
```

图3-3 8位二进制数FFH转换为BCD数255H的二进制运算

除法指令对所有的状态标志位均无定义。

3. 补码数的扩展指令

从除法指令的操作可知，要把一个 8 位二进制数除以一个 8 位二进制数，要有一个 16 位二进制数在 AX 中，只是把一个 8 位的被除数放入 AL 中是不行的，因为除法指令将把任何在 AH 中的数当作被除数的高 8 位。所以在做 8 位除以 8 位的除法之前先要把 8 位被除数扩展为 16 位，在做 16 位除以 16 位的除法之前要把 16 位被除数扩展为 32 位，在做 32 位除以 32 位的除法之前要把 32 位被除数扩展为 64 位，才能保证除法指令的正确操作。这种扩展对于无符号数除法是很容易办到的，只需将被除数的高半部清 0 即可。对符号整数除法就不能用将被除数的高半部清 0 来实现，而要通过扩展符号位来把被除数扩展。例如，把 −2 的 8 位形式 1111 1110 转换为 16 位形式 1111 1111 1111 1110，即要把高半部全部置 1（−2 的符号位）；而把 +3 的 8 位形式 0000 0011 转换成 16 位形式 0000 0000 0000 0011，却要把高半部全部置 0（+3 的符号位）。

指令格式

```
CBW(convert byte to word)
CWD/CWDE(convert word to double word)
CDQ(convert double to quad)
```

将字节扩展为字指令 CBW 所执行的操作是把 AL 的最高位扩展到 AH 的所有位。将字扩展为双字指令 CWD 把 AX 的最高位扩展到 DX 的所有位，形成 DX 和 AX 中的双字；而将字扩展为双字指令 CWDE 把 AX 的最高位扩展到 EAX 的高 16 位，形成 EAX 中的双字。将字扩展为双字指令 CWDE 与符号位扩展传送指令功能相当，指令 CWDE 就等于指令 MOVS EAX,AX。将双字扩展为 4 字指令 CDQ 把 EAX 的最高位扩展到 EDX 的所有位，形成 EDX 和 EAX 中的 4 字。在做 8 位除以 8 位，16 位除以 16 位，32 位除以 32 位的符号整数除法之前，应先扩展 AL、AX 或 EAX 中的被除数。

例如，在数据段中，有一符号字数组变量 ARRAY，第一个字是被除数，第二个字是除数，接着存放商和余数，其程序段如下。

```
MOV SI,OFFSET ARRAY
MOV AX,[SI]
CWD
IDIV WORD PTR 2[SI]
MOV 4[SI],AX
MOV 6[SI],DX
```

一般情况下，都将符号数看作补码数，扩展指令和符号整数除法指令仅对补码数适用。若特别指出该符号数为原码数，则其扩展和除法运算都要另编程序段实现。

3.1.2 BCD 数调整指令

第 2.3 节介绍的加减指令和本节介绍的乘除指令都是对二进制数进行操作。二进制数算术运算指令对 BCD 数进行运算，会得到一个非 BCD 数或不正确的 BCD 数。例如：

```
0000 0011B + 0000 1001B = 0000 1100B
0000 1001B + 0000 0111B = 0001 0000B
```

第一个结果是非 BCD 数；第二个结果是不正确的 BCD 数。其原因是 BCD 数向高位的

进位是逢10进1,而4位二进制数向高位进位是逢16进1,中间相差6。若再加上6,就可以得到正确的BCD数:

```
0000 1100B + 0000 0110B = 0001 0010B
0001 0000B + 0000 0110B = 0001 0110B
```

8086/8088对BCD数使用二进制数算术运算指令进行运算,然后执行一条能把结果转换成正确的BCD数的专用调整指令来处理BCD数的结果。

1. BCD数加法调整指令DAA和AAA

指令格式

```
DAA
AAA
```

DAA指令的意义是将AL中的数当作两个压缩BCD数相加之和来进行调整,得到两位压缩BCD数。具体操作是,若(AL & 0FH)>9或AF=1,则AL加上6;若(AL &0F0H)>90H或CF=1,则AL加60H。例如:

```
MOV AX,3456H
ADD AL,AH          ; AL = 8AH
DAA                ; AL = 90H
```

例3.1 已知字变量W1和W2分别存放着两个压缩BCD数,编写求两数之和,并将其和送到SUM字节变量中的程序。

此例应注意以下两个问题。

(1) 定义字变量W1和W2的4位数应为BCD数,其后要加H,只有这样定义装入内存中的数据才是4位BCD数。

(2) BCD数的加减运算只能做字节运算,不能做字运算。这是因为加减指令把操作数都当作二进制数进行运算,运算之后再用调整指令进行调整,而调整指令只对AL作为目的操作数的加减运算进行调整。

程序如下。

```
stack      segment stack 'stack'
           dw 32 dup(0)
stack      ends
data        segment
W1         DW  8931H
W2         DW  5678H
SUM        DB 3 DUP(0)
data       ends
code       segment
begin      proc far
           assume  ss: stack,cs: code,ds: data
           push ds
           sub ax,ax
           push ax
           mov ax,data
           mov ds,ax
           MOV AL, BYTE PTR W1     ; AL = 31H
```

```
        ADD AL, BYTE PTR W2     ; AL = A9H, CF = 0, AF = 0
        DAA                     ; AL = 09H, CF = 1
        MOV SUM,AL
        MOV AL, BYTE PTR W1 + 1 ; AL = 89H
        ADC AL, BYTE PTR W2 + 1 ; AL = E0H, CF = 0, AF = 1
        DAA                     ; AL = 46H, CF = 1
        MOV SUM + 1,AL
        MOV SUM + 2, 0          ; 处理向万位的进位
        RCL SUM + 2, 1          ; 也可用指令 ADC SUM + 2, 0
        ret
begin   endp
code    ends
        end  begin
```

AAA 指令的意义是将 AL 中的数当作两个非压缩 BCD 数相加之和进行调整，得到正确的非压缩 BCD 数送 AX。具体操作是，若(AL&0FH) >9 或 AF=1，则(AL+6)& 0FH 送 AL，AH 加 1 且 CF 置 1；否则 AL & 0FH 送 AL，AH 不变且 CF 保持 0 不变。应特别注意，AAA 指令执行前 AH 的值。例如：

```
MOV AX,0806H
ADD AL,AH           ; AX = 080EH
MOV AH,0
AAA                 ; AX = 0104H
```

又如，若要将两个 BCD 数的 ASCII 码相加，得到和的 ASCII 码，可以直接用 ASCII 码相加，加后再调整。

```
MOV AL,35H          ; '5'
ADD AL,39H          ; '9',AL = 6EH
MOV AH,0
AAA                 ; AX = 0104H
OR AX,3030H         ; AX = 3134H 即'14'
```

例 3.2 已知字变量 W1 和 W2 分别存放着两个非压缩 BCD 数，编写求两数之和，并将其和送到 SUM 字节变量中的程序。

定义字变量 W1 和 W2 的数应为两位非压缩 BCD 数，其后要加 H。程序如下。

```
stack   segment stack 'stack'
        dw 32 dup(0)
stack   ends
data    segment
W1      DW  0809H
W2      DW  0607H
SUM     DB 3 DUP(0)
data    ends
code    segment
begin   proc  far
        assume  ss: stack,cs: code,ds: data
        push ds
        sub ax,ax
        push ax
```

```
        mov ax,data
        mov ds,ax
        MOV AX,W1                ; AX = 0809H
        ADD AL,BYTE PTR W2       ; AL = 10H, AF = 1
        AAA                      ; AX = 0906H
        MOV SUM,AL
        MOV AL,AH
        ADD AL,BYTE PTR W2 + 1   ; AL = 0FH, AF = 0
        MOV AH,0
        AAA                      ; AL = 05H, AH = 01H
        MOV WORD PTR SUM + 1,AX
        ret
begin   endp
code    ends
        end  begin
```

由调整指令所执行的具体操作可以看到,对结果进行调整时要用到进位标志和辅助进位标志,所以调整指令应紧跟在BCD数作为加数的加法指令之后。所谓"紧跟"是指在调整指令与加法指令之间不得有改变标志位的指令。

2. BCD数减法调整指令DAS和AAS

指令格式

```
DAS
AAS
```

DAS指令的功能是将AL中的数当作两个压缩BCD数相减之差来进行调整,得到正确的压缩BCD数。具体操作是:若(AL & 0FH)>9或AF=1,则AL减6,(AL &0F0H)>90H或CF=1,则AL减60H。例如:

```
MOV AX, 5634H
SUB AL,AH          ; AL = DEH,有借位
DAS                ; AL = 78H,保持借位即134 - 56
```

AAS指令的功能是将AL中的数当作两个非压缩BCD数相减之差进行调整得到正确的非压缩BCD数。具体操作是:若(AL & 0FH)>9或AF=1,则(AL-6) & 0FH送AL,AH减1;否则AL & 0FH送AL,AH不变。应特别注意,AAS指令执行前AH的值。例如:

```
MOV AX,0806H
SUB AL,07H         ; AX = 08FFH
AAS                ; AX = 0709H
```

3. 非压缩BCD数乘除法调整指令AAM和AAD

压缩BCD数对乘除法的结果不能进行调整,故只有非压缩BCD数乘除法调整指令。

指令格式

```
AAM
AAD
```

AAM指令的功能是将AL中的小于64H的二进制数进行调整,在AX中得到正确的

非压缩 BCD 数。具体操作是 AL/0AH 送 AH，AL MOD 0AH 送 AL。例如：

```
MOV AL,63H
AAM                ; AX = 0909H
```

例 3.3 字变量 W 和字节变量 B 分别存放着两个非压缩 BCD 数，编写求两数之积，并将它存储到 JJ 字节变量中的程序。

定义字变量 W 的数应为两位非压缩 BCD 数，其后要加 H。

由于是 BCD 数的乘法，所以只能用 AL 作被乘数，因此要做两次乘法。先将第一次乘法的部分积 0603H 存入 JJ＋1 和 JJ 两个单元(JJ＋1 存高 8 位 06H，JJ 存低 8 位 03H)，然后将两次乘法的部分积相加。第二次乘法的部分积 0207H(在 AX 中)与第一次乘法的部分积相加，是第二次乘法部分积的低 8 位与第一次乘法的部分积的高 8 位相加，相加的进位加入第二次部分积的高 8 位中。由于这个加法也是非压缩 BCD 数的加法，故加后也要调整，调整后若产生进位，该进位直接加入 AH，由于此时 AH 的内容正是第二次乘法部分积的高 8 位，所以加法调整指令正好调整到位。程序如下。

```
stack     segment stack 'stack'
          dw 32 dup(0)
stack     ends
data      segment
W         DW 0307H
B         DB 9
JJ        DB 3 DUP(0)
data      ends
code      segment
begin     proc far
          assume ss: stack,cs: code,ds: data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
          MOV AL, BYTE PTR W       ; AL = 07H
          MUL B                    ; AX = 003FH
          AAM                      ; AX = 0603H
          MOV WORD PTR JJ, AX
          MOV AL, BYTE PTR W + 1   ; AL = 03H
          MUL B                    ; AX = 001BH
          AAM                      ; AX = 0207H
          ADD AL, JJ + 1           ; 07H + 06H = 0DH,即 AL = 0DH
          AAA                      ; 调整后的进位,直接加入 AH! AX = 0303H
          MOV WORD PTR JJ + 1,AX
          ret
begin     endp
code      ends
          end begin
```

AAD 指令的功能是将 AX 中的两位非压缩 BCD 数变换为二进制数。在做两位非压缩 BCD 数除以一位非压缩 BCD 数时，先将 AX 中的被除数调整为二进制数，然后用二进制除

法指令 DIV 相除,保存 AH 中的余数后,再用 AAM 指令把商变回为非压缩 BCD 数。例如:

```
MOV AX,0906H
MOV DL,06H
AAD          ; AX = 0060H
DIV DL       ; AL = 10H、AH = 0
MOV DL,AH    ; 存余数
AAM          ; AX = 0106H
```

应注意的是,除法的调整不同于加法、减法和乘法,它们的调整是在相应运算操作之后进行,而除法的调整在除法操作之前进行。

例 3.4 字变量 W 和字节变量 B 中分别存放着两个非压缩 BCD 数,编制程序求二者的商和余数,并分别存放到字变量 QUOT 和字节变量 REMA 中。

定义字变量 W 的数应为两位非压缩 BCD 数,其后要加 H。

由于是 BCD 数的除法,所以要先调整,因此先将 W 中的非压缩 BCD 数取到 AX 中,然后将 AX 中的非压缩 BCD 数调整为二进制数。二进制数的除法之后,又应用 AAM 指令将结果调整为非压缩 BCD 数。AAM 指令是将 AL 中的小于 100 的二进制数调整为非压缩 BCD 数,存入 AX 中,因此,调整前应将除法产生的余数存入 REMA 中。程序如下。

```
stack    segment stack 'stack'
         dw 32 dup(0)
stack    ends
data     segment
W        DW 0909H
B        DB 5
REMA     DB 0
QUOT     DW 0
data     ends
code     segment
begin    proc far
         assume ss: stack,cs: code,ds: data
         push ds
         sub ax,ax
         push ax
         mov ax,data
         mov ds,ax
         MOV AX,W
         AAD                ; 0909H→63H
         DIV B              ; 63H ÷5 = 13H … 4, AL = 13H, AH = 04H
         MOV REMA, AH
         AAM                ; 13H→0109H
         MOV QUOT,AX
         ret
begin    endp
code     ends
         end begin
```

调整指令都隐含着 AX 或 AL,都在 AX 或 AL 中进行。

3.1.3 顺序程序设计举例

例 3.5 从键盘上输入 0～9 中任一自然数 N,将其立方值送显示器显示。

求一个数的立方值可以用乘法运算实现,也可以用查表法实现。查表法运算速度比较快,是常用的计算方法。因只要送显示,故将 0～9 的立方值的 ASCII 码按顺序造一立方表。立方值最大值为 729,需三个单元存放它的 ASCII 码,表的每项的单元数相同,再在每项之后加一个“$”,所以立方表的每项均占 4 个字节。根据这种存放规律可推知,表的偏移首地址与自然数 N 的 4 倍之和,正是 N 的立方值和 $ 的 ASCII 码的存放单元的偏移首地址。

用查表法编制的程序如下。

```
stack     segment stack 'stack'
          dw 32 dup(0)
stack     ends
data      segment
INPUT     DB 'PLEASE INPUT N(0-9): $'
LFB       DB'  0$   1$   8$  27$  64$ 125$ 216$ 343$ 512$ 729$'
N         DB0
data      ends
code      segment
start     proc far
          assume ss:stack,cs:code,ds:data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
          MOV DX,OFFSET INPUT  ; 显示提示信息
          MOV AH,9
          INT 21H
          MOV AH,1             ; 输入并回显 N(1 号功能调用)
          INT 21H
          MOV N,AL
          MOV AH,2             ; 换行(2 号功能调用)
          MOV DL,0AH
          INT 21H
          MOV DL,N
          AND DL,0FH           ; 将'N'转换为 N
          MOV CL,2             ; 将 N 乘以 4
          SHL DL,CL
          MOV DH,0             ; 8 位 4N 扩展为 16 位
          ADD DX,OFFSET LFB    ; 4N+表的偏移地址
          MOV AH,9
          INT 21H
          ret
start     endp
code      ends
          end start
```

例 3.6 编写两个32位无符号数的乘法程序。

使用32位指令编写的程序如下。

```
.386
stack       segment stack USE16 'stack'
            dw 32 dup (0)
stack       ends
data        segment USE16
AB          DD 12345678H
CD          DD 12233445H
ABCD        DD 2 DUP(0)
data        ends
code        segment USE16
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV EAX, AB
            MUL CD
            MOV ABCD,EAX
            MOV ABCD + 4, EDX
            ret
start       endp
code        ends
            end start
```

若用16位指令编写该程序就要用16位乘法指令做4次乘法,然后把部分积相加,如图3-4所示,相应的程序如下。

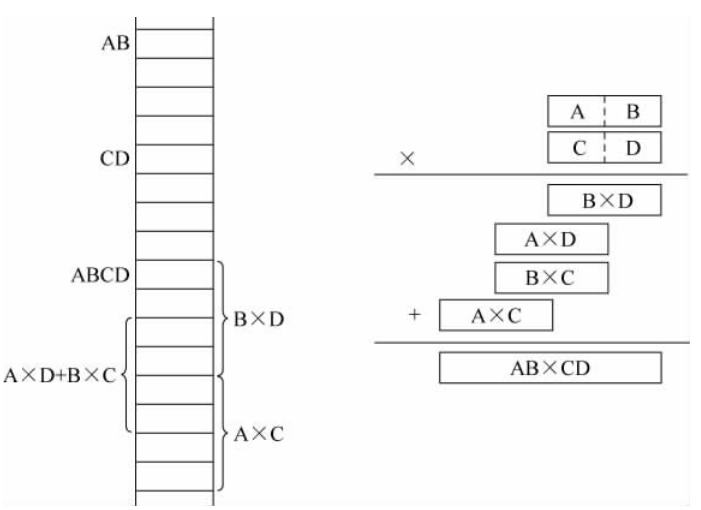

图3-4 32位无符号数的乘法

```
stack       segment stack 'stack'
            dw 32 dup (0)
stack       ends
data        segment
AB          DD 12345678H
CD          DD 12233445H
ABCD        DD 2 DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV BX,OFFSET AB
            MOV AX, [BX + 4]            ; d→AX,AX = 3445H
            MUL WORD PTR [BX]           ; d × b
            MOV [BX + 8], AX            ; 存 db 的低 16 位
            MOV [BX + 10], DX           ; 存 db 的高 16 位
            MOV AX, [BX + 4]            ; d→AX; AX = 3445H
            MUL WORD PTR [BX + 2]       ; d × a
            ADD [BX + 10], AX           ; d × a 的低 16 位加上 d × b 的高 16 位
            ADC DX, 0                   ; 上述加法若有进位,则加入 d × a 的高 16 位中
            MOV [BX + 12], DX           ; 存 d × a 的高 16 位
            MOV AX, [BX + 6]            ; c→AX; AX = 1223H
            MUL WORD PTR [BX]           ; c × b
            ADD [BX + 10], AX           ; c × b 的低 16 位加入
            ADC [BX + 12], DX           ; c × b 的高 16 位加入存放单元
            MOV BYTE PTR [BX + 14], 0   ; 清 0a × c 的高 16 位的存放单元
            ADC BYTE PTR [BX + 14], 0   ; c × b 的高 16 位加入时产生的进位存入
            MOV AX, [BX + 6]            ; c→AX; AX = 1223H
            MUL WORD PTR [BX + 2]       ; a × c
            ADD [BX + 12], AX           ; a × c 的低 16 位加入
            ADC [BX + 14], DX           ; a × c 的高 16 位加入
            ret
start       endp
code        ends
            end start
```

3.2 分支程序设计

顺序程序的特点是从程序的第一条指令开始,按顺序执行,直到最后一条指令。然而,许多实际问题并不能设计成顺序程序,需要根据不同的条件做出不同的处理。把不同的处理方法编制成各自的处理程序段,运行时计算机根据不同的条件自动做出选择判别,绕过某些指令,仅执行相应的处理程序段。按这种方式编制的程序,执行的顺序与指令存储的顺序失去了完全的一致性,称为分支程序。分支程序是计算机利用改变标志位的指令和转移指

令来实现的。

转移指令有JMP和Jcond两类。前者是无条件转移,后者是条件转移。JMP指令将控制转向其后的目的标号指定的地址。条件转移指令紧跟在能改变并设置状态标志的指令之后,根据设置的状态标志决定程序的走向,当条件满足时控制程序转向其后的目的地址,否则不发生程序转移而顺序向下执行。

3.2.1 条件转移指令

指令格式

```
Jcond short - lable
```

该指令的功能是,若条件满足则程序转移到目的标号short-lable,即short-lable的偏移地址送IP,否则顺序执行。

条件转移指令是相对转移指令。相对转移指令的转移范围,即从当前地址(执行该指令时的IP)到目的标号地址的偏移量为-128～127(从条件转移指令的地址到目的标号的地址则为-126～129),故条件转移指令只能实现段内转移。从80386开始扩大了转移范围,在实地址方式下能够转移到代码段的任何位置。

1. 简单的条件转移指令

简单的条件转移指令是仅根据一个可测试标志位实现转移的指令。简单的条件转移指令如表3-1所示。

表3-1 简单的条件转移指令

指令助记符	功　能	标志设置
JE/JZ	相等/等于0转移	ZF=1
JNE/JNZ	不相等/不等于0转移	ZF=0
JC	有进(借)位转移	CF=1
JNC	无进(借)位转移	CF=0
JS	为负转移	SF=1
JNS	为正转移	SF=0
JO	溢出转移	OF=1
JNO	无溢出转移	OF=0
JP/JPE	偶转移	PF=1
JNP/JPO	奇转移	PF=0

2. 无符号数条件转移指令

条件转移指令常根据比较指令比较两个数的关系的结果来实现转移。两个数的关系除了相等与否外,还有两个数中哪一个比较大。但这就有一个有趣的问题,如8位二进制数11111111大于00000000吗?答案既可肯定又可否定。因为若视这两个二进制数为无符号数,11111111当然大于00000000;若视这两个二进制数为符号数(补码),11111111为-1,就比0小了。为此要使用两种术语来区分无符号数和符号数的这种关系。如果把数作为符号数来比较,就使用术语"小于"和"大于";如果把数作为无符号数来比较,就使用术语"低于"和"高于"。因此8位二进制数11111111高于00000000,小于00000000,而00000001既

高于又大于 00000000。所以 80x86 设置了符号数的条件转移指令和无符号数的条件转移指令。

无符号数条件转移指令有 4 条，如表 3-2 所示。

表 3-2　无符号数条件转移指令

指令助记符	功　　能	指令助记符	功　　能
JB/JNAE	低于/不高于等于转移	JA/JNBE	高于/不低于等于转移
JNB/JAE	不低于/高于等于转移	JNA/JBE	不高于/低于等于转移

已知 AL 中有一个十六进制数的 ASCII 码，将它转换为十六进制数需判别该 ASCII 码是 0～9 的 ASCII 码还是 A～F 的 ASCII 码，可以用无符号数条件转移指令来判别，其程序段如下。

```
        CMP AL,'A'
        JB NS7              ; AL 低于 A 的 ASCII 码去 NS7
        SUB AL,7
NS7:    SUB AL,30H
        ⋮
```

也可以用简单的条件转移指令：

```
JC NS7
```

代替无符号数条件转移指令。因为 AL 低于 A 的 ASCII 码，AL 与'A'比较(即相减)后有借位。

3. 符号数条件转移指令

符号数条件转移指令有 4 条，如表 3-3 所示。

表 3-3　符号数条件转移指令

指令助记符	功　　能	指令助记符	功　　能
JL/JNGE	小于/不大于等于转移	JG/JNLE	大于/不小于等于转移
JNL/JGE	不小于/大于等于转移	JNG/JLE	不大于/小于等于转移

3.2.2　无条件转移指令

条件转移指令的转移有一定的范围，若超过这个范围时就要在这个范围的某处放一条无条件转移指令来实现转移。无条件转移指令没有范围限制。在分支程序中还要用无条件转移指令将各分支又重新汇集到一起。

无条件转移指令有直接转移和间接转移两类。

1. 无条件直接转移指令

格式

```
JMP target
```

功能：将控制转向目的标号 target，即 target 的偏移地址送(E)IP，target 的段基址送 CS(若 target 与该指令在同一段，则无此操作，只改变(E)IP)。

2. 无条件间接转移指令

格式

```
JMP dest
```

操作数可为寄存器和存储器。操作数若为寄存器,则是将寄存器的内容送(E)IP。操作数若为存储器则是存储器直接寻址;操作数若为字变量,则是将字变量送(E)IP(仅能实现段内转移);若为双字变量,则是将双字送CS和(E)IP(可实现段间转移)。例如:

```
JMP W(W为字变量)的操作是将W的内容送(E)IP;
```

JMP DUW(DUW为双字变量)的操作是将WORD PTR DUW的内容送(E)IP、WORD PTR DUW+2的内容送CS。又如:

JMP WORD PTR [BX]的操作是将[BX+1]和[BX]送(E)IP;

JMP DWORD PTR [BX]的操作是将[BX+1]和[BX]送(E)IP;[BX+3]和[BX+2]送CS。

3.2.3 分支程序设计举例

分支的实现有多种方法,这里仅介绍两种基本方法:利用比较转移指令实现分支和利用跳转表实现分支。

例3.7 编制计算下面函数值的程序(X、Y均为字节符号数)。

$$Z=\begin{cases}1 & X\geqslant 0,Y\geqslant 0\\ -1 & X<0,Y<0\\ 0 & X,Y\text{异号}\end{cases}$$

根据题意,先判断X、Y是否异号,异号Z赋0后结束;若不异号即同号,则只需再判断其中任一数的符号即可得知X和Y是大于等于0还是小于0。使用XOR指令判别X、Y是否异号,XOR指令执行X、Y按位加,X和Y的符号位按位加,若X、Y同号则按位加结果为0即SF=0,若X、Y异号则按位加结果为1即SF=1。为了减少分支,采用先赋值后判断的方法。赋0和1是用MOV指令完成的,赋-1是用对1求补即用求补指令NEG完成的。

```
stack     segment stack 'stack'
          dw 32 dup(0)
stack     ends
data      segment
X         DB -5
Y         DB 20
Z         DB 0
data      ends
code      segment
start     proc far
          assume ss:stack,cs:code,ds:data
          push ds
          sub ax,ax
          push ax
```

```
        mov ax,data
        mov ds,ax
        MOV AL,X
        XOR AL,Y                ; 根据 X、Y 的符号置 S 标志,相同为 0
        JS DIFF                 ; 相异为 1,X、Y 相异去 DIFF
        MOV Z,1
        CMP X,0                 ; 相同后,判断其中某数的符号
        JNS NOCHA               ; 大于等于 0 结束
        NEG Z                   ; 小于 0; Z 赋 - 1 结束
NOCHA:  RET
DIFF:   MOV Z,0
        ret
start   endp
code    ends
        end start
```

例 3.8 将字节变量 SB 中的 8 位二进制数以十六进制数形式在显示器上显示。程序如下。

```
stack   segment stack 'stack'
        dw 32 dup(0)
stack   ends
data    segment
SB      DB 9AH
OBUF    DB 3 DUP(0)
data    ends
code    segment
start   proc far
        assume ss:stack,cs:code,ds:data
        push ds
        sub ax,ax
        push ax
        mov ax,data
        mov ds,ax
        MOV CX, 204H
        MOV BX,0
        MOV AL,SB
AGAIN:  MOV AH, 3
        SHL AX,CL
        CMP AH,39H
        JBE NAD7
        ADD AH,7
NAD7:   MOV OBUF[BX],AH
        INC BX
        DEC CH
        JNZ AGAIN
        MOV OBUF[BX],'$'
        MOV DX,OFFSET OBUF      ;将输出数据区的偏移首地址送 DX
        MOV AH,9
        INT 21H
```

```
            ret
start       endp
code        ends
            end start
```

例 3.9 从键盘上输入两位十六进制数,将其拼合成一个字节存入字节变量 SB 中。

本题主要是将十六进制数字符即其 ASCII 码转换为十六进制数的程序段的设计。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
IBUF        DB 3,0,3 DUP(0)
SB          DB 0
data        ends
code        segment
begin       proc far
            assume ss: stack, cs: code, ds: data
            push ds
            sub ax, ax
            push ax
            mov ax, data
            mov ds, ax
            MOV DX, OFFSET IBUF          ; 10 号功能调用,输入两位十六进制数
            MOV AH, 10
            INT 21H
            MOV AX, WORD PTR IBUF + 2    ; 输入的字符送 AX,高位字符在 AL 中
            SUB AX, 3030H                ; 将两个字符的 ASCII 码变为两位十六进制数
            CMP AL,0AH
            JB LNSUB7
            SUB AL,7
LNSUB7:     CMP AH,0AH
            JB HNSUB7
            SUB AH,7
HNSUB7:     MOV CL, 4                    ; 将 AX 中的两位十六进制数拼合成一个字节
            SHL AL ,CL
            OR AL, AH
            MOV SB,AL
            ret
begin       endp
code        ends
            end begin
```

例 3.10 某工厂的产品共有 8 种加工处理程序 P0～P7,而某产品应根据不同情况,做不同的处理,其选择由输入的值 0～7 来决定。若按 0～7 以外的键,则退出该产品的加工处理程序。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
```

```
data        segment
INPUT       DB 'INPUT(0～7): $ '
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
AGAIN:      MOV DX,OFFSET INPUT
            MOV AH,9
            INT 21H
            MOV AH,1
            INT 21H
            CMP AL,'0'
            JE  P0
            CMP AL,'1'
            JE P1
            ⋮
            CMP AL,'7'
            JE P7
            RET
P0:
            ⋮
            JMP AGAIN

            ⋮
P7:
            ⋮
            JMP AGAIN
start       endp
code        ends
            end start
```

该程序的编制方法是利用比较转移指令实现分支，每次比较转移实现二叉分支。这种方法编程条理清楚，容易实现。但各处理程序不能太长，且分支不能太多。因为分叉进入各处理程序所用的指令均是条件转移指令(此例为 JE Pi)，条件转移指令所允许的转移范围为－128～127，若各处理程序较长或者分支再多一些，就会超过条件转移指令所允许的范围。为了解决这个问题，可以利用跳转表法来实现这种多叉分支。

跳转表法实现分支的具体做法是，在数据区中开辟一片连续存储单元作为跳转表，表中顺序存放各分支处理程序的跳转地址。跳转地址在跳转表中的位置，即它们在表中的偏移始地址等于跳转表首地址加上它们各自的序号与所占字节数的乘积。要进入某分支处理程序只须查找跳转表中相应的地址即可。用跳转表法编制的程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
```

```
data        segment
INPUT       DB 'INPUT(0-7): $'
PTAB        DW P0,P1,P2,P3,P4,P5,P6,P7
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
AGAIN:      MOV DX,OFFSET INPUT
            MOV AH,9
            INT 21H
            MOV AH,1
            INT 21H
            CMP AL,'0'
            JB EXIT
            CMP AL,'7'
            JA EXIT
            AND AX,0FH
            ADD AL,AL
            MOV BX,AX
            JMP PTAB[BX]
EXIT:       RET
P0:
            ⋮
            JMP AGAIN

            ⋮
P7:
            ⋮
            JMP AGAIN
start       endp
code        ends
            end start
```

3.3 循环程序设计

顺序程序和分支程序中的指令,最多只执行一次。在实际问题中重复地做某些事的情况是很多的,用计算机来做这些事就要重复地执行某些指令。重复地执行某些指令,最好用循环程序实现。

循环程序一般有以下4部分。

(1) 循环准备。也称循环初始化,它为循环做必要的准备。这部分的主要工作是建立地址指针,置计数器,设置一些必要的常数,将工作寄存器或工作单元清0等。

(2) 循环体。完成循环的基本操作,是循环程序的实质所在。

(3) 循环的修改。修改或恢复某些内容,为下一轮循环做好必要的准备。修改的内容一般包括计数器、寄存器和基址或变址寄存器。有的循环还要恢复某些计数器,寄存器和基址或变址寄存器。

(4) 循环的控制。修改计数器,判断控制循环的继续或终止。

任何循环程序一般都应有这 4 部分,但各部分的界限并不是很清楚。有时为设计方便或为了节省存储空间或为了控制简单等原因,这 4 部分形成相互包含、相互交叉的情况,很难分出某条或某几条指令究竟属于哪一部分。

3.3.1 循环程序的基本结构

循环程序的基本结构有两种,如图 3-5 所示。一种(图 3-5(a))是“先执行,后判断”,这种结构的循环先执行一次循环体,后判断循环是否结束。这种结构的循环至少执行一次循环体,上面所举程序属于这种结构。另一种(图 3-5(b))是“先判断,后执行”,这种结构的循环首先判断是否进入循环,再视判断结果,决定是否执行循环体。这种结构的循环,如果一开始就满足循环结束的条件,会一次也不执行循环体,即循环次数可以为 0,若能确保一个循环程序在任何情况下都不会出现循环次数为 0 的情况,采用以上任一种结构都可以;当不能确保时,用后一种结构为好。

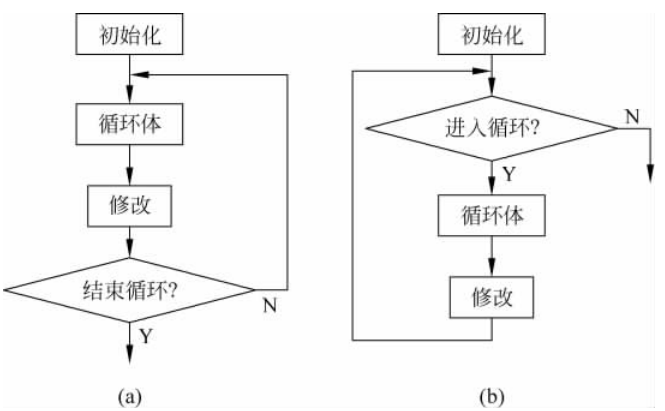

图 3-5 循环程序的基本结构

例如,编程统计字变量 W 中有多少位 1。

这个程序最好采用“先判断,后执行”的结构。先将统计 1 的位数的字节变量 N 清 0,再判断 W 是否为 0,如果为 0,则不必做统计工作了;如果不为 0,则将 W 左移或者右移一位,通过判断移出位是 1 还是 0,决定字节变量 N 是否加 1 来统计 W 中 1 的位数。其程序如下。

```
stack       segment stack 'stack'
            dw  32 dup(0)
stack       ends
data        segment
W           DW 1999H
N           DB0
data        ends
```

```
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV N,0
LOP:        CMP W,0
            JE DONE
            SHL W,1
            JNCLOP
            INC N
            JMP LOP
DONE:       ret
start       endp
code        ends
            end start
```

3.3.2 重复控制指令

循环程序必须要有指令来控制循环,重复控制指令在循环的首部或尾部确定是否进行循环。确定是否循环的方法通常是在计数寄存器CX或ECX中预置循环次数,重复控制修改CX或ECX,再判断CX或ECX。当CX或ECX不等于0时,循环至目的地址;否则顺序执行该重复控制指令的下一条指令。重复控制指令同条件转移指令一样,也是相对转移指令,重复控制指令的目的地址必须在本指令地址的-126～129字节的范围之内。这些指令对串操作和数据块操作是很有用的。重复控制指令有下述5条。

1. LOOP指令

指令格式

```
LOOP short-lable
```

指令的意义是将计数寄存器CX或ECX减1,然后判断计数寄存器CX或ECX是否等于0。若CX或ECX≠0,则控制程序转移到short-lable所指的指令,否则顺序执行。

使用LOOP指令之前,必须把循环次数送入计数寄存器中,一条LOOP short-lable指令,相当于DEC CX或者DEC ECX和JNZ short-lable两条指令。使用LOOP指令实现0次循环必须使用图3-5(b)的结构,且在循环准备时将CX或ECX置1。若将CX或ECX置0,则循环要进行65 536次或者4 294 967 296次。其原因是执行LOOP指令时,CX或ECX先减1,后判断CX或ECX是否为0。

2. LOOPZ / LOOPE指令

指令格式

```
LOOPZ short-lable 或 LOOPE short-lable
```

指令意义是先将计数寄存器减1,然后判断计数寄存器的内容和ZF标志的状态。若计数寄存器≠0,且ZF=1时,将程序转移到short-lable所指的指令,否则顺序执行。

3. LOOPNZ / LOOPNE 指令

指令格式

```
LOOPNZ short - lable 或 LOOPNE short - lable
```

指令意义是先将计数寄存器减 1,然后判断计数寄存器的内容和 ZF 标志的状态。若计数寄存器≠0,且 ZF=0 时,将程序转移到 short-lable 所指的指令,否则顺序执行。

4. JCXZ 指令

指令格式

```
JCXZ  short - lable
```

指令意义是若 CX=0,则将程序转移到 short-lable 所指的指令,否则顺序执行。

5. JECXZ 指令(80386 及其后继微处理器可用)

指令格式

```
JECXZ  short - lable
```

指令意义是若 ECX=0,则将程序转移到 short-lable 所指的指令,否则顺序执行。

3.3.3 单重循环程序设计举例

1. 计数控制的循环程序

此类循环程序的特点是循环次数已知,故可用某个寄存器或存储单元作为计数器,用计数器的值来控制循环的结束。

例 3.11 计算 Z=X+Y,其中 X 和 Y 是双字变量。

双字变量占 4 个字节,故其和可能占 5 个字节。采用 32 位指令编制的程序如下。

```
.386
stack       segment stack USE16 'stack'
            dw 32 dup (0)
stack       ends
data        segment USE16
X           DD 752028FFH
Y           DD 9405ABCDH
Z           DB 5 DUP(0)
data        ends
code        segment USE16
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV EAX, X
            ADD EAX,Y
            MOV DWORD PTR Z,EAX
            MOV Z+4, 0
            RCL Z+4,1
```

```
            ret
start       endp
code        ends
            end start
```

仅用16位指令编制的程序如下。

```
stack       segment stack 'stack '
            dw 32 dup (0)
stack       ends
data        segment
X           DD 752028FFH
Y           DD 9405ABCDH
Z           DB 5 DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV CX,4
            MOV SI,0
            AND AX,AX                    ; 清 CF,即 CF 为 0
AGAIN:      MOV AL,BYTE PTR X[SI]
            ADC AL,BYTE PTR Y[SI]
            MOV Z[SI],AL
            INC SI
            LOOP AGAIN
            MOV Z[SI],0
            RCL Z[SI],1
            ret
start       endp
code        ends
            end start
```

例 3.12 编写将某数据区中的十六进制数加密的程序,每个数字占一个字节。

在实际的应用中为了对某些信息保密,通常可通过硬件或软件的方法对信息进行加密,使用时再进行解密。软件的加密和解密是通过运行加密和解密程序实现的。加密程序是用与原数字对应的加密表中的信息代替原数字。解密程序则是通过解密表将加密信息还原。表3-4是任意设计的十六进制数字的加密数和相应的解密数。

表 3-4 十六进制数字的加密数和相应的解密数表

十六进制数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
加密数	A	9	8	E	F	1	0	B	2	5	D	3	7	4	6	C
解密数	6	5	8	B	D	9	E	C	2	1	0	7	F	A	3	4

加密程序如下。

```
stack       segment stack 'stack'
            dw 32 dup (0)
stack       ends
data        segment
HEXS        DB 1,2,… … ,0EH
NUMBER      EQU $ - HEXS
JMB         DB 0AH,9,8,0EH,0FH,1,0,0BH,2,5,0DH,3,7,4,6,0CH
JMHEX       DB NUMBER DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV BH,0                    ; JMB 表中的位移量的高 8 位为 0
            MOV SI,0                    ; HEXS 和 JMHEX 两个数据区的位移量
            MOV CX,NUMBER
AGAIN:      MOV BL,HEXS[SI]             ; 取十六进制数
            MOV AL,JMB[BX]              ; AL←[BX + JMB](十六进制数的加密数)
            MOV JMHEX[SI],AL
            INC SI
            LOOP AGAIN
            ret
start       endp
code        ends
end start
```

例 3.13 将字节变量 SB 中的 8 位二进制数以二进制数形式送显示器显示。

先将字节变量中的一位二进制数移入 AH 中,再将移入的二进制数变为 ASCII 码。为了避免通过 CF 来传递二进制数,先将 SB 中的 8 位二进制数送入 AL 中,再左移 AX,将一位二进制数直接移入 AH 中。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
SB          DB 9AH
OBUF        DB 9 DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
```

```
                mov ds,ax
                MOV CX, 8
                MOV BX,0
                MOV AL,SB
AGAIN:          MOV AH, 0
                SHL AX,1
                ADD AH,30H
                MOV OBUF[BX],AH
                INC BX
                LOOP AGAIN
                MOV OBUF[BX],'$'
                MOV DX,OFFSET OBUF          ;将输出数据区的偏移首地址送 DX
                MOV AH,9
                INT 21H
                ret
```

例 3.14 编写将输入的十进制数(−32 768～32 767)转换为二进制数的程序。

将 i 位十进制整数转换为二进制数的方法有多种,其中之一是使用算法$((0\times10+a_{i-1})\times10+\cdots)\times10+a0$。例如,将 548 转换为二进制数,计算机执行的二进制运算如图 3-6 所示。

图 3-6 中的计算机运算的结果是:10 0010 0100B=224H,即 548=224H。

5×10	(5×10)+4	(5×10+4)×10	(5×10+4)×10+8
00000101	00110010	00110110	1000011100
× 1010	+ 100	× 1010	+ 1000
101	00110110	110110	1000100100
+ 101		+110110	
00110010		1000011100	

图 3-6 将 548 转换为二进制数的二进制运算

非计算机计算即将十进制数转换为十六进制数的计算如下:

$$548=512+32+4=200H+20H+04H=224H$$

用该方法将输入的十进制数转换为二进制数比较适宜,因为输入的十进制数是一个单元一位按高位到低位的顺序存放在数据存储区中的,且十进制数的位数也是已知的。在转换之前,先判别该数是正数还是负数。为简化设计,正数则按习惯不输入+号。若是负数,则十进制数的位数要比输入的字符数少一位;另外在转换完后,还要将转换的结果进行求补。

数据段中定义两个变量:IBUF 和 BINARY。IBUF 共计定义 9 个单元用来存放输入的十进制字符串,因为输入的字符串连同负号最多 6 个字符,加 1 个回车,共计 7 个。另外,根据 10 号功能调用的入口参数的要求,还要在第 1 单元装入允许输入的字符数,并预留第 2 单元给 10 号功能调用存放实际输入的字符数。字变量 BINARY 用来存放转换的结果。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
IBUF        DB 7,0,7 DUP(0)
```

```
BINARY      DW 0
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV DX, OFFSET IBUF          ; 输入十进制数(10 号功能调用)
            MOV AH, 10
            INT 21H
            MOV CL, IBUF + 1             ; 十进制数(含"-"号)的位数送 CX
            MOV CH,0
            MOV SI,OFFSET IBUF + 2       ; 指向输入的第一个字符
            CMP BYTE PTR [SI], '-'       ; 判是否为负数
            PUSHF                        ; 保护零标志,供转换之后再判别
            JNE SININC                   ; 正数跳转,去 SININC
            INC SI                       ; 越过"-"号指向数字
            DEC CX                       ; 实际字符数少 1("-"号)
SININC:     MOV AX, 0                    ; 开始将十进制数转换为二进制数
AGAIN:      MOV DX, 10                   ; ((0×10 + a4)×10 + …)×10 + a0
            MUL DX
            AND BYTE PTR [SI], 0FH       ; 将十进制数的 ASCII 码转换为 BCD 数
            ADD AL,[SI]
            ADC AH,0
            INC SI
            LOOP AGAIN
            POPF                         ; 恢复判断是否为负数时的零标志 ZF
            JNZ NNEG                     ; 非 0 即为正数,则不求补
            NEG AX                       ; 负数对其绝对值求补
NNEG:       MOV BINARY, AX               ; 存放结果
            ret
start       endp
code        ends
            end start
```

例 3.15 对多个字符号数求和,结果不超出双字符号数,以十六进制数的形式显示其结果。

采用 32 位指令编制的程序如下。

```
            .386
stack       segment stack USE16 'stack'
            dw 32 dup(0)
stack       ends
data        segment USE16
NUM         DW 1111H,2222H,3333H,4444H,5555H,6666H,7777H,8888H,9999H
COUNT       EQU( $ - NUM)/2
RESULT      DD 0
OBUF        DB 10 DUP(0)
```

```
data        ends
code        segment USE16
begin       proc far
            assume ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV CX,COUNT
            MOV EBX,0
AGAIN1:     MOVSX EAX,NUM[EBX * 2]
            ADD RESULT,EAX
            INC EBX
            LOOP AGAIN1
            MOV DI,OFFSET OBUF
            MOV CX,8                    ; 将8位十六进制数拆转为ASCII字符
AGAIN2:     ROL RESULT,4
            MOV AL,0FH
            AND AL,BYTE PTR RESULT
            ADD AL,30H
            CMP AL,39H
            JNA NA7
            ADD AL,7
NA7:        MOV [DI],AL
            INC DI
            LOOP AGAIN2
            MOV WORD PTR[DI],'$H'
            MOV BX,OFFSET OBUF - 1      ; 去掉前面的0
CONT:       INC BX
            CMP BYTE PTR [BX],'0'
            JE CONT
            MOV DX,BX
            MOV AH,9
            INT 21H
            ret
egin        endp
code        ends
            end begin
```

例 3.16 从键盘上输入两个加数 N1 和 N2(1～8 位十进制数),求和并送显示器显示。

该程序分为 3 部分:输入两个加数 N1 和 N2、相加并将结果以 ASCII 码形式存入输出数据区 OBUF、输出结果。输入两个加数部分和输出结果部分有三个数据区:N1、N2 和 OBUF。用 BX 作数位较多加数的指针,用 SI 作数位较少加数的指针,用 DI 作存放和数的 ASCII 码的输出数据区 OBUF 的指针。相加并将结果以 ASCII 码形式存入输出数据区部分是本程序的主要部分。N1 和 N2 两个加数的数位不一定相等,哪个的数位多也未做规定。因此,应先按较少数位的位数进行两数的相加运算,然后进行数位较多加数的剩余位与进位的相加,最后再处理两数相加后的进位。例如 99 567+768,先做 567+768 三位的相加运算,再做 99 与进位的相加,最后再处理进位。程序框图如图 3-7 所示。程序如下。

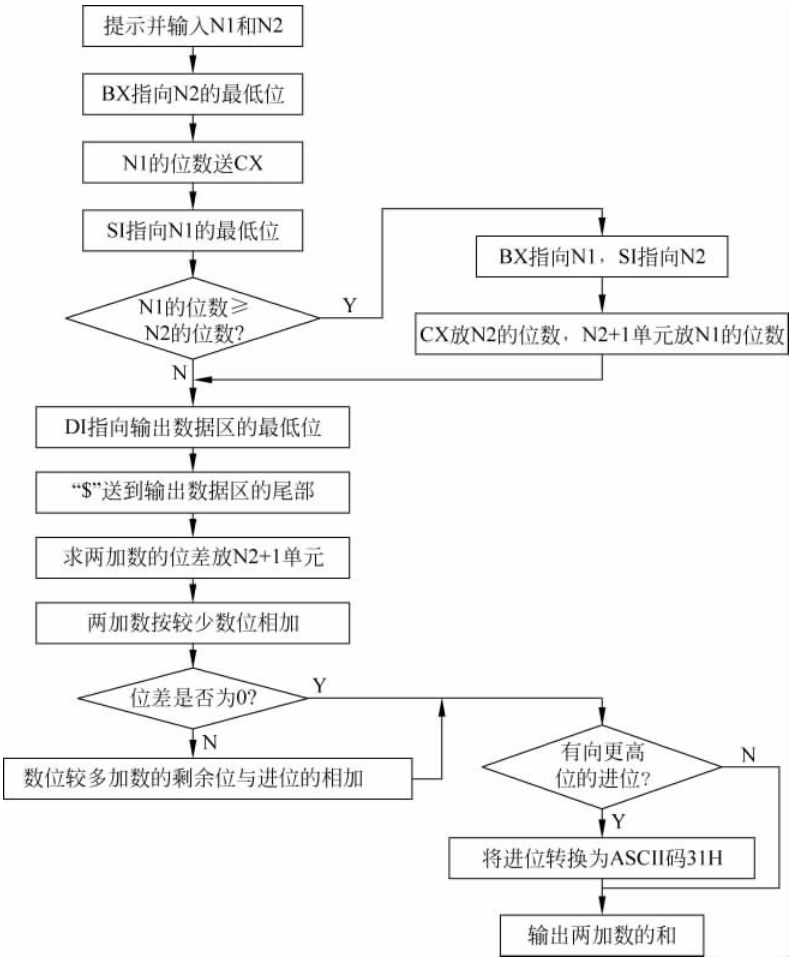

图 3-7　例 3.16 的程序框图

```
stack     segment stack 'stack'
          dw 32 dup(0)
stack     ends
data      segment
OBF1      DB'PLEASE INPUT N1: $ '
OBF2      DB'PLEASE INPUT N2: $ '
N1        DB 9,0,9 DUP(0)
N2        DB 9,0,9 DUP(0)
OBUF      DB 10 DUP(0)
data      ends
code      segment
start     proc far
          assume ss:stack,cs:code,ds:data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
```

```
        MOV DX, OFFSET OBF1          ; 提示并输入 N1
        MOV AH,9
        INT 21H
        MOV DX, OFFSET N1
        MOV AH,10
        INT 21H
        MOV DL,0AH                   ; 换行
        MOV AH,2
        INT 21H
        MOV DX, OFFSET OBF2          ; 提示并输入 N2
        MOV AH,9
        INT 21H
        MOV DX, OFFSET N2
        MOV AH,10
        INT 21H
        MOV DL,0AH                   ; 换行
        MOV AH,2
        INT 21H
        MOV BL,N2 + 1                ; N2 的位数送 BX
        MOV BH,0
        ADD BX,OFFSET N2 + 1         ; BX 指向 N2 的最低位
        MOV CL,N1 + 1                ; N1 的位数送 CX 和 SI
        MOV CH,0
        MOV SI,CX
        ADD SI,OFFSET N1 + 1         ; SI 指向 N1 的最低位
        CMP CL,N2 + 1                ; N1 与 N2 的位数相比
        JC NXCHG
        XCHG CL,N2 + 1               ; N1 大,CX 放 N2 的位数,N2 + 1 单元放 N1 的位数
        XCHG BX,SI                   ; BX 指向 N1,SI 指向 N2
NXCHG:  MOV DI,WORD PTR N2 + 1       ; DI 指向输出数据区的最低位
        AND DI,00FFH
        ADD DI,OFFSET OBUF
        MOV BYTE PTR[DI + 1],'$'     ; '$'送到输出数据区的尾部
        SUB N2 + 1,CL                ; 求两加数的位差放 N2 + 1 单元
        MOV AL,0                     ; 清 AL(进位)
AGAIN:  MOV AH, 0                    ; 清 AH,以便 AAA 指令放进位
        ADD AL,[BX]                  ; 将某加数的 1 位与低位的进位相加
        ADD AL,[SI]                  ; 加另一加数的 1 位
        AAA
        ADD AL,30H                   ; 一位和数转换为 ASCII 码
        MOV [DI],AL                  ; 存入输出数据区
        MOV AL,AH                    ; 向高位的进位放 AL 中
        DEC BX                       ; 调整指针 BX、SI 和 DI
        DEC SI
        DEC DI
        LOOP AGAIN                   ; 按较少数位的两数相加完否?
        MOV CL,N2 + 1                ; 两加数的位差送 CL
        AND CL,CL                    ; 判断位差是否为 0
        JZ DONE
AGAIN1: MOV AH,0                     ; 数位较多加数的剩余位与进位的相加
        ADD AL,[BX]
```

```
            AAA
            ADD AL,30H
            MOV [DI],AL
            MOV AL,AH
            DEC BX
            DEC DI
            LOOP AGAIN1
DONE:       AND AL,AL                    ; 判断是否有向更高位的进位
            JNZ DONE1
            INC DI                       ; 无向更高位的进位,调整指针
            JMP DONE2
DONE1:      ADD AL,30H                   ; 有则将进位转换为 ASCII 码 31H
            MOV [DI],AL
DONE2:      MOV DX,DI                    ; 将输出数据的偏移首地址送 DX
            MOV AH,9
            INT 21H
            ret
start       endp
code        ends
            end start
```

例 3.17 在屏幕中部 4 处分别显示黑桃、红心方块和草花,如图 3-8 所示。

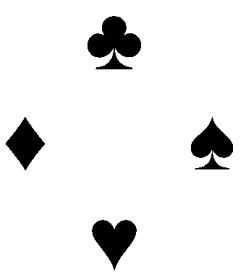

图 3-8 屏幕显示的黑桃、红心、方块和草花

将草花、方块、黑桃和红心的 ASCII 码、显示属性、显示的行和列按顺序排成数据表,用计数循环调用 BIOS 中的显示器服务程序。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
CDSH        DB 5,70H,10,40
            DB 4,74H,13,37
            DB 6,70H,13,43
            DB 3,74H,16,40
data        ends
code        segment
begin       proc far
            assume ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
```

```
            push ax
            mov ax,data
            mov ds,ax
            MOV AH,0                    ; 设置 80×25 彩色文本方式
            MOV AL,3
            INT 10H
            MOV AH,15                   ; 读显示方式
            INT 10H
            MOV SI,OFFSET CDSH
            MOV CX,4
AGAIN:      PUSH CX
            MOV AH,2
            MOV DH,[SI+2]               ; 置光标位置
            MOV DL,[SI+3]
            INT 10H
            MOV AH,9                    ; 写字符和属性
            MOV AL,[SI]
            MOV BL,[SI+1]
            MOV CX,1
            INT 10H
            ADD SI,4
            POP CX
            LOOP AGAIN
            ret
begin       endp
code        ends
            end begin
```

例 3.18 在屏幕中部画一条红线。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
code        segment
begin       proc far
            assume ss: stack,cs: code
            push ds
            sub ax,ax
            push ax
            MOV AH,0                    ; 设置 320×200 彩色图形方式
            MOV AL,5
            INT 10H
            MOV AH,0BH                  ; 设置背景色为黑色
            MOV BH,0
            MOV BL,0
            INT 10H
            MOV AH,0BH                  ; 设置彩色组 0
            MOV BH,1
            MOV BL,0
            INT 10H
            MOV CX,320                  ; 设置 320 个像点计数器
```

```
            MOV BP,0                    ; 设置像点列号初值
AGAIN:      PUSH CX
            MOV CX,BP                   ; 写像点,行号为 100,列号从 0 至
            MOV AH,0CH                  ; 319 像点为红色(彩色值为 2)
            MOV AL,2
            MOV DX,100
            INT 10H
            INC BP                      ; 像点列号加 1
            POP CX
            LOOP AGAIN
            ret
begin       endp
code        ends
            end  begin
```

2. 条件控制的循环程序

例 3.19 将存储器中的 16 位无符号二进制数转换成 BCD 数,在显示器上以十进制数形式显示出来。

将二进制数转换为十进制数的方法也有多种。可以采用除 10 取余法将二进制转换为十进制数,每除一次得到一位十进制数,最先得到最低位,最后得到最高位。由于仅由显示器显示,所以得到一位十进制数后即将它转换为它的 ASCII 码存入输出数据区中。输出数据区的尾部存入“$”供 9 号功能调用作结束符用。本例虽是 16 位二进制数,但不能用 16 位除以 8 位的除法,而要用 32 位除以 16 位的除法,这是因为 16 位二进制数除以 10 所得的商仅在 AL 中,AL 有可能装不下所得商! 如 32 768/10＝3276…8,即商为 3276,余数为 8,AL 装不下商 3276(0CCCH)。32 768/10＝3276…8 的二进制运算为：8000H/0AH＝0CCCH…08H(0CCCH＝800H＋400H＋80H＋40H＋0CH＝2048＋1024＋128＋64＋12＝3276)。16 位无符号二进制数的最大值为 5 位十进制数 65 535,再加上 9 号功能调用的结束符“$”,输出数据区 OBUF 最多只需 6 个单元。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
BINARY      DW 55H
OBUF        DB 6 DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV BX,OFFSET OBUF + 5
            MOV BYTE PTR [BX],'$'
            MOV AX,BINARY
            MOV CX, 10                  ; 做 32 位除以 16 位的除法,故将 10 送 CX
```

```
AGAIN:    MOV DX, 0                ; 无符号数扩展将16位扩展为32位
          DIV CX
          ADD DL,30H               ; 将DL中的一位十进制数转换为ASCII码
          DEC BX                   ; 调整指针
          MOV [BX],DL
          OR AX,AX                 ; 根据商是否为0,设置ZF
          JNZ AGAIN                ; 判断商是否为0,不为0继续除以10
          MOV DX,BX                ; 将输出数据区的偏移首地址送DX
          MOV AH,9
          INT 21H
          ret
start     endp
code      ends
          end start
```

例 3.20 从键盘上输入 0～9 中任一自然数 N,将 2 的 N 次方值在显示器的下一行显示出来。

求一个数的 N 次方值可以用查表法实现,也可以用乘法运算实现。用查表法求一个数的 N 次方值请见例 3.5,此处使用乘法运算来编制该程序。由于乘法运算都是乘 2 操作,故用逻辑左移实现。设其初值为 1,输入的 N 值就是对该初值移位的位数。求得的值是一个二进制数,为了输出还要将二进制数转换为十进制数的 ASCII 码。其最大值是 2 的 9 次方,$2^9=512$,最大值的 ASCII 码占三个单元,再加上回车、换行和'$',所以输出数据区 OBUF 最多 6 个单元。将二进制数转化为 BCD 数,采用"除 10 取余法",余数为 ASCII BCD 数,余数加上 30H 即可得到余数的 ASCII 码。除 10 操作一直进行到商等于 0 为止,本例中的最大商值为 51(=33H),故可以采用 16 位除以 8 位的除法操作。

使用逻辑移位指令求 2^N、简单的条件转移和除法指令将二进制数转换为 BCD 数编制的程序如下。

```
stack     segment stack 'stack'
          dw 32 dup(0)
stack     ends
data      segment
OBUF      DB 6 DUP(0)
data      ends
code      segment
start     proc far
          assume ss:stack,cs:code,ds:data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
          MOV AH,1
          INT 21H
          AND AL,0FH               ; 将'N'转换为N
          MOV CL,AL                ; 将移位的位数N送CL
          MOV AX,1
          SHL AX,CL
```

```
            MOV BX,5
            MOV OBUF[BX],'$'
            MOV CL,10                    ; 转换为十进制数的 ASCII 码
AGAIN:      DIV CL
            ORAH,30H
            DEC BX
            MOV OBUF[BX],AH
            MOV AH,0                     ; 将被除数(商)扩展为 16 位
            AND AL,AL                    ; 根据商是否为 0 置 ZF
            JNZ AGAIN
            SUB BX,2
            MOV WORD PTR OBUF[BX],0A0DH; 存入回车换行
            MOV DX,BX
            ADD DX,OFFSET OBUF
            MOV AH,9
            INT 21H
            ret
start       endp
code        ends
            end start
```

3. 双重控制的循环程序

例 3.21 已知字节变量 BUF 存储区中存放着以 0DH(回车的 ASCII 码)结束的十进制数的 ASCII 码。编程检查该字节变量存储区中有无非十进制数,若有显示“ERROR”;若无则统计十进制数的位数(小于 100)并送显示器显示。

结束本程序有两种情况:存储区中有非十进制数或者统计工作完毕。程序执行后,显示器显示 ERROR 或者显示统计的十进制数的位数(00～99)。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
BUF         DB '345678……',0DH
OBUF        DB 3 DUP(0)
ERR         DB 'ERROR$'
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AX,0                     ; 统计十进制数的位数(ASCII BCD 数)
            MOV BX,0                     ; 存储区的位移量
AGAIN:      CMP BUF[BX],0DH
            JE DONE
            CMP BUF[BX],'0'
            JB ERROR
```

```
            CMP BUF[BX],'9'
            JA ERROR
            INC AL                          ; AAA 不判 CF,所以可不用 ADD 指令
            AAA
            INC BX
            JMP AGAIN
DONE:       OR AX,3030H                     ; ASCII BCD 数转换为十进制数的 ASCII 码
            MOV OBUF + 1,AL
            MOV OBUF,AH
            MOV OBUF + 2,'$ '
            MOV DX,OFFSET OBUF
            MOV AH,9
            INT 21H
            RET
ERROR:      MOV DX,OFFSET ERR
            MOV AH,9
            INT 21H
            ret
start       endp
code        ends
            end  start
```

3.3.4 多重循环程序设计举例

多重循环指的是循环体内仍然是循环程序,也就是循环的嵌套。称具有嵌套的循环程序为多重循环程序。

例 3.22 编制将字节变量 BUF 存储区中存放的 n 个无符号数排序的程序。

排序问题可以采用逐一比较法或两两比较法。

逐一比较法的具体做法是:将第 1 个单元中的数与其后 $n-1$ 个单元中的数逐个比较,每次比较之后总是把较大的数放在一个寄存器中,经过 $n-1$ 次比较之后得到 n 个数中的最大数,存入第 1 个单元。接着将第 2 个单元中的数与其后的 $n-2$ 个单元中的数逐个比较,经过 $n-2$ 次比较得到 $n-1$ 个数的最大数(亦即 n 个数中的第二大数)存入第 2 个单元。如此重复下去,当最后两个单元中的数比较之后,从大到小的顺序就排好了。其程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
BUF         DB 20,19,…,250
COUNT       EQU $ - BUF
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
```

```
            mov ds,ax
            MOV SI,OFFSET BUF
            MOV DX,COUNT－1              ；设置外循环计数器
OUTSID:     MOV CX,DX                    ；设置内循环计数器
            PUSH SI
            MOV AL,[SI]
INSIDE:     INC SI
            CMP AL,[SI]
            JNC NEXCHG
            XCHG [SI],AL
NEXCHG:     LOOP INSIDE
            POP SI
            MOV [SI],AL
            INC SI
            DEC DX
            JNZ OUTSID
            ret
start       endp
code        ends
            end start
```

两两比较法的具体做法是：首先将第 1 个单元中的数与第 2 个单元中的数进行比较，若前者大于后者，两数不交换；反之则交换。然后将第 2 个单元中的数与第 3 个单元中的数进行比较，按同样原则决定是否交换。以此类推，最后将第 $n-1$ 个单元中的数与第 n 个单元中的数比较，也按同样的原则决定是否交换。如此经过 $n-1$ 次循环，n 个数中的最小数到了第 n 个单元。再经过 $n-2$ 次上述同样的比较与交换的循环，n 个数中的第 2 小数到了第 $n-1$ 单元。这样不断地循环下去，最多经过 $n-1$ 次这样的循环，就可以将这 n 个数按从大到小的顺序排好。在内循环中两两比较的次数，第 1 次为 $n-1$，第 2 次为 $n-2$，……一般情况下，无须经过 $n-1$ 次外循环，就可以将这 n 个单元中的数据按顺序排好。为了去掉不必要的外循环，可以设置一个标记，在每次内循环开始时，该标记置“1”。若在内循环中发生过交换，则修改该标记为 2。内循环结束以后，检查该标记，若不为 1，表示内循环发生过交换，即数的顺序未排好，继续进行外循环；若为“1”，则表示数已按顺序排好，就结束外循环。其程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
BUF         DB20,19,…,250
COUNT       EQU $ - BUF
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
```

```
            mov ds,ax
            MOV DX,COUNT-1              ; 循环次数
            MOV AH,1                    ; 未交换标记
OUTSID:     MOV SI,OFFSET BUF
            MOV CX,DX                   ; 设置内循环计数器
INSIDE:     MOV AL,[SI]
            INC SI
            CMP AL,[SI]
            JNC NXCHG
            XCHG AL,[SI]
            MOV [SI-1],AL
            MOV AH,2                    ; 置交换标记
NXCHG:      LOOP INSIDE
            DEC AH
            JZ  BACK                    ; 判断是否进行过交换,没交换则退出循环
            DEC DX                      ; 修改内循环次数
            JNZ OUTSID                  ; 判断外循环是否结束,没结束则继续循环
BACK:       ret
start       endp
code        ends
            end start
```

例 3.23 已知 $m\times n$ 矩阵 A 的元素 a_{ij}(80H 和～7FH,字节符号数)按行序存放在存储区中,试编写程序求每行元素之和 S_i(8000H 和～7FFFH,字符号数)。

程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
A           DB 11H,12H,13H,14H,15H
N           EQU $-A
            DB 21H,22H,23H,24H,25H
            DB 31H,32H,33H,34H,35H
            DB 41H,42H,43H,44H,45H
M           EQU ($-A)/N
S           DW M DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV SI,OFFSET A
            MOV DI,OFFSET S
OUTSID:     MOV CX,N
            MOV DX,0
INSIDE:     MOV AL,[SI]
```

```
            CBW
            ADD DX,AX
            INC SI
            LOOP INSIDE
            MOV [DI],DX
            ADD DI,2
            DEC M
            JNZ OUTSID
            ret
start       endp
code        ends
            end start
```

例 3.24　多位压缩 BCD 数与两位压缩 BCD 数相乘。

8086/8088 乘法指令可以实现 8 位或 16 位二进制数相乘；经过 AAM 指令调整还可以实现两个非压缩 BCD 数相乘。但对于两个两位压缩 BCD 数，就不能用乘法指令直接相乘，这是因为没有相应的调整指令。只能用累加的方法，编一个程序来实现。具体算法是对被乘数累加乘数所规定的次数。被乘数每次累加的和都要经过 DAA 指令调整；乘数每次减 1 之后也要用 DAS 指令调整。由两个两位压缩 BCD 数相乘的算法，可推知多位压缩 BCD 数与两位压缩 BCD 数乃至多位 BCD 数相乘的算法，如图 3-9 所示。程序如下。

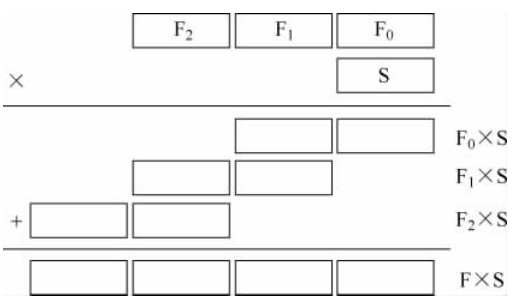

图 3-9　多位压缩 BCD 数与两位压缩 BCD 数相乘

```
stack       segment stack 'stack'
            dw 32 dup (0)
stack       ends
data        segment
FIRST       DB 78H,56H,…,12H
COUNT       EQU $ - FIRST
SECOND      DB  15H
THIRD       DB COUNT + 1 DUP(0)
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
```

```
                MOV SI,0
                MOV THIRD[SI],0
                MOV CX,COUNT
OUTSID:         MOV BL,SECOND
                MOV AX,0
INSIDE:         ADD AL,FIRST[SI]
                DAA
                XCHG AH,AL
                ADC AL,0
                DAA
                XCHG AH,AL
                XCHG AL,BL
                SUB AL,1
                DAS
                XCHG AL,BL
                JNZ INSIDE
                ADD AL,THIRD[SI]
                DAA
                MOV THIRD[SI],AL
                XCHG AH,AL
                ADC AL,0
                DAA
                INC SI
                MOV THIRD[SI],AL
                LOOP OUTSID
                ret
start           endp
code            ends
                end start
```

例 3.25 编制程序在显示屏幕的左上角显示一排“小人”,白色、红色、绿色、黄色各8个。

图 3-10 三个字符组成的小人

小人由三个字符组成,如图 3-10 所示。只要将组成“小人”的这三个字符分为三行在一列上显示出来,就可在显示屏幕上出现如图 3-10 所示的“小人”。再配以字符的显示属性就可以出现彩色“小人”。这三个字符的 ASCII 码分别是 01H,04H 和 13H。白色、红色、绿色和黄色这 4 种颜色字符的显示属性分别是 7、4、2 和 14。

按题意要求将这 32 个“小人”排在显示屏幕的第 0～2 行,则白色“小人”排在第 0～7 列;红色“小人”排在第 8～15 列;绿色“小人”排在第 16～23 列;黄色“小人”排在第 24～31 列。

将字符的 ASCII 码,显示属性和行列坐标(偏移量)组成一个数据表,用双重循环调取该表中的元素,即可以完成这 32 个“小人”的显示。

```
stack           segment stack 'stack'
                dw 32 dup(0)
stack           ends
data            segment
```

```
DATAB     DB 1,7,0,0,1,4,0,8,1,2,0,16,1,14,0,24
          DB 4,7,1,0,4,4,1,8,4,2,1,16,4,14,1,24
          DB 13H,7,2,0,13H,4,2,8,13H,2,2,16,13H,14,2,24
data      ends
code      segment
begin     proc far
          assume ss: stack,cs: code,ds: data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
          MOV AH,0                  ; 80×25 彩色字符方式
          MOV AL,3
          INT 10H
          MOV AH,15                 ; 读取当前页号
          INT 10H
          MOV SI,OFFSET DATAB
          MOV CX,3                  ; 3 行
AGAOT:    PUSH CX
          MOV CX,4                  ; 4 色
AGAIN:    PUSH CX
          MOV AH,2                  ; 光标位置(DH 和 DL)
          MOV DH,[SI + 2]
          MOV DL,[SI + 3]
          INT 10H
          MOV AL,[SI]               ; 写字符和属性(AL 和 BL)
          MOV AH,9
          MOV BL,[SI + 1]
          MOV CX,8
          INT 10H
          ADD SI,4
          POP CX
          LOOP AGAIN
          POP CX
          LOOP AGAOT
          ret
begin     endp
code      ends
          end begin
```

3.4 串处理程序设计

循环程序的设计几乎都是利用基址或变址寄存器建立地址指针，设置循环计数器，执行完循环体后修改地址指针和计数器，判断循环是否结束。宏汇编为了方便这类循环程序设计，设计了字符串操作指令以及重复前缀。它们的方便之处体现在，只要按要求设计好初始值，执行正确的串操作指令及重复前缀，就可以完成规定的操作，而不要考虑地址指针如何修改，循环次数如何控制等问题。而这两个问题也正是循环程序成功或失败的关键所在，从

而简化了程序设计。

串操作指令有一个共同的规定：源串的偏移地址指针用SI或ESI,在无段更换前缀的情况下,段地址取自DS段寄存器；目的串的偏移地址指针用DI或EDI,在无段更换前缀的情况下,段地址总是取自ES段寄存器；源串和目的串的偏移地址指针的移动方向由方向标志DF确定：DF=0,SI和DI增量,DF=1,SI和DI减量,增量或减量的量值由串属性决定。

3.4.1 方向标志置位和清除指令

1. 方向标志置位指令

指令格式

```
STD
```

指令的操作是将DF置“1”。

2. 方向标志清除指令

指令格式

```
CLD
```

指令的操作是将DF置“0”,即清除DF。

3.4.2 串操作指令

串操作指令有5条指令。它们是串传送指令MOVS、从源串中取数指令LODS、往目的串中存数指令STOS、串比较指令CMPS和串搜索指令SCAS。

1. 串传送指令

指令格式

```
(1) MOVS dest-string,source-string
(2) MOVSB
(3) MOVSW
(4) MOVSD
```

指令的意义是把DS所指向的数据段中(形式(1)的指令无段更换前缀的情况下)SI或ESI为偏移地址的源串中的一个字节、一个字或者一个双字(形式(1)的指令由串的属性确定),传送到ES所指向的数据段中DI或EDI为偏移地址的目的串；并且相应地修改SI或ESI和DI或EDI,以指向串中的下一个字节、字或双字。在有段更换的情况下才使用形式(1)的指令,源串和目的串的属性要相同。

SI或ESI和DI或EDI的修改方向和修改值由方向标志DF和串的属性决定：

DF=0,SI或ESI和DI或EDI增量,字节串增1,字串增2,双字串增4；

DF=1,SI或ESI和DI或EDI减量,字节串减1,字串减2,双字串减4。

2. 从源串中取数指令

指令格式

```
(1) LODS source - string
(2) LODSB
(3) LODSW
```

(4) LODSD

指令的意义是将 DS 数据段中 SI 或 ESI 为偏移地址的源串中的一个字节、一个字或一个双字取出送 AL、AX 或者 EAX；同时修改 SI 或 ESI 指向下一个字节、字或者双字。

3. 往目的串中存数指令

指令格式

```
(1) STOS dest - string
(2) STOSB
(3) STOSW
(4) STOSD
```

指令的意义是将 AL、AX 或者 EAX 中的内容存放到 ES 数据段中 DI 或 EDI 为偏移地址的目的串中；同时修改 DI 或 EDI 指向下一个字节、字或者双字。

4. 串比较指令

指令格式

```
(1) CMPS dest - string,source - string
(2) CMPSB
(3) CMPSW
(4) CMPSD
```

指令的意义是用 DS：SI 或 DS:ESI 指向的源串中的一个字节、字或者双字减去 ES：DI 或 ES:EDI 指向的目的串中的一个字节、字或者双字，减的结果既不送入源串也不送入目的串，仅根据减操作设置标置位；同时修改 SI 或 ESI 和 DI 或 EDI 指向下一个字节、字或者双字。

5. 串搜索(扫描)指令

指令格式

```
(1) SCAS dest - string
(2) SCASB
(3) SCASW
(4) SCASD
```

指令的意义是用 AL、AX 或者 EAX 减去 ES：DI 或 ES:EDI 指向的目的串中的一个字节、字或者双字，减的结果，既不送累加器也不送目的串中，减操作仅影响标志位；同时修改 DI 或 EDI 指向下一操作数。

3.4.3 重复前缀

重复前缀有三个：重复 REP、相等/为 0 重复 REPE/REPZ 和不相等/不为 0 重复 REPNE/REPNZ。

重复前缀只允许用在串操作指令之前，与串操作指令仅能用空格隔开。它的作用是使紧跟其后的串操作指令重复执行，重复执行的次数由 CX 或 ECX 的值决定。它与重复控制指令不同的是先判 CX 或 ECX 是否等于 0，然后确定是否重复，等于 0 不再重复，不等于 0 继续重复。每重复一次，CX 或 ECX 减 1。若 CX 或 ECX 的初值为 0，则串操作指令一次也不执行。

1. REP

REP作为串传送指令和往目的串中存数指令的前缀,使传送操作无条件地重复执行,直到CX=0或ECX=0为止。

2. REPE/REPZ

REPE/REPZ作为串比较指令和串搜索指令的前缀,使比较或搜索操作重复执行,直到CX=0或ZF=0或者ECX=0或ZF=0为止。

3. REPNE/REPNZ

REPNE/REPNZ作为串比较指令和串搜索指令的前缀,使比较或搜索操作重复执行,直到CX=0或ZF=1或者ECX=0或ZF=1为止。

3.4.4 串操作程序设计举例

串操作程序设计应注意以下三点。

(1) 源串一般用DS:SI或者DS:ESI间址,目的串一定用ES:DI或者ES:EDI间址。对于不很长的串操作,简单而又不易出错的方法是把源串和目的串都定义在同一个数据段中,且使DS和ES均指向该数据段。

(2) 一定要先设置方向标志DF,规定串操作的方向。

(3) 若使用重复前缀,则应将串长度送CX或ECX寄存器。

例3.26 用串操作指令和不用串操作指令两种方式编写将source-string传送到dest-string的程序。

用串操作指令编写的程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
SSTRING     DB'*FGDHFJGU#@…'
COUNT       EQU $ - SSTRING
data        ends
DATAE       SEGMENT
DSTRING     DB COUNT DUP(0)
DATAE       ENDS
code        segment
start       proc far
            assume ss: stack,cs: code,ds: data,ES: DATAE
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AX,DATAE
            MOV ES,AX
            MOV SI,OFFSET SSTRING
            MOV DI,OFFSET DSTRING
            MOV CX,COUNT
```

```
            CLD
            REP MOVSB
            ret
start       endp
code        ends
            end start
```

不用串操作指令编写的程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
SSTRING     DB'*FGDHFJGU#@…'
COUNT       EQU $ - STRING
data        ends
DATAE       SEGMENT
DSTRING     DB COUNT DUP(0)
DATAE       ENDS
code        segment
start       proc far
            assume ss: stack,cs: code,ds: data,ES: DATAE
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV AX,DATAE
            MOV ES,AX
            MOV SI,OFFSET SSTRING
            MOV DI,OFFSET DSTRING
            MOV CX,COUNT
AGAIN:      MOV AL,[SI]
            MOV ES: [DI],AL
            INC SI
            INC DI
            LOOP AGAIN
            ret
start       endp
code        ends
            end start
```

通过该例可以看出，串操作指令的功能完全可以用其他指令代替，带重复前缀的串操作也可以用循环程序来实现，只是使用串操作指令编程要方便一些、程序简短一些。串传送指令还可以实现存储单元之间的直接传送，而 MOV 指令要用寄存器作为桥梁才能实现存储单元之间的传送。

例 3.27 编制程序将一串字节符号数中的正、负数分别送到变量 PLUS 和 MINUS 的数据存储区中去，同时记录 0 的个数(小于 65 536)。

程序框图如图3-11所示,程序如下。

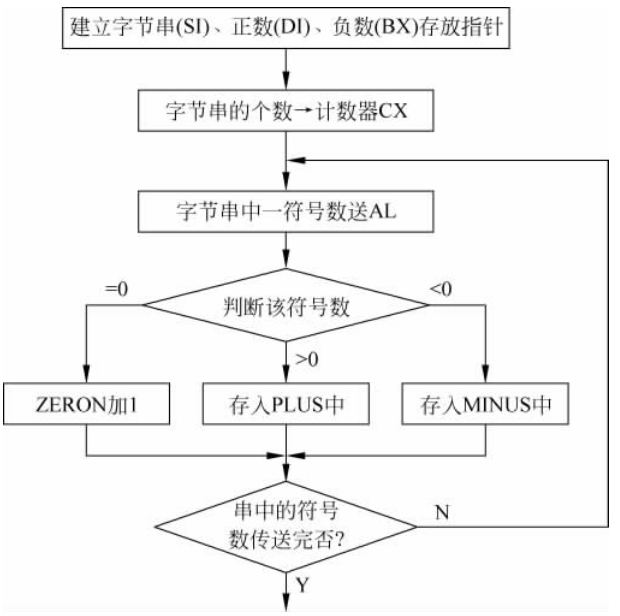

图3-11　例3.27的程序框图

```
stack     segment stack 'stack'
          dw 32 dup (0)
stack     ends
data      segment
STRING    DB 1, -1, 5, 10, 0, -25, 80, …
COUNT     EQU $ - STRING
PLUS      DB COUNT DUP(0)
MINUS     DB COUNT DUP(0)
ZERON     DW 0
data      ends
code      segment
start     proc far
          assume ss: stack, cs: code, ds: data
          push ds
          sub ax, ax
          push ax
          mov ax, data
          mov ds, ax
          MOV ES,AX
          MOV SI,OFFSET STRING        ; SI指向源串即字节串
          MOV DI,OFFSET PLUS          ; DI指向目的串即正数存放单元的偏移地址
          MOV BX,OFFSET MINUS         ; BX指向负数存放单元的偏移地址
          MOV CX,COUNT
          CLD
AGAIN:    LODSB                       ; 字节串中一符号数送AL
          AND AL,AL                   ; 判断符号
```

```
            JZ ZERO                 ; 为 0 去 ZERO
            JS MINU                 ; 为负去 MINU
            STOSB                   ; 为正存入 PLUS
            LOOP AGAIN
            RET
ZERO:       INC ZERON
            LOOP AGAIN
            RET
MINU:       MOV [BX], AL            ; 为负存入 MINUS
            INC BX
            LOOP AGAIN
            ret
start       endp
code        ends
            end start
```

不用串操作编写的程序段如下。

```
            XOR SI, SI              ; SI 指向字节串的第一个字符
            XOR DI,DI               ; DI 指向正数存放单元的第一个单元
            XOR BX,BX               ; BX 指向负数存放单元的第一个单元
            MOV CX,COUNT
AGAIN:      MOV AL,STRING[SI]       ; 字节串中一符号数送 AL
            INC SI
            AND AL,AL               ; 判断符号数
            JZ ZERO                 ; 为 0 去 ZERO
            JS MINU                 ; 为负去 MINU
            MOV PLUS[DI],AL         ; 为正存入 PLUS
            INC DI
            LOOP AGAIN
            RET
ZERO:       INC ZERON
            LOOP AGAIN
            RET
MINU:       MOV MINUS[BX], AL       ; 为负存入 MINUS
            INC BX
            LOOP AGAIN
            ⋮
```

例 3.28 编制判断两个串长相等的字符串 STRING1 和 STRING2 是否相同的程序。若不同,将不同处的偏移地址送 DIFF 字变量,否则将－1 送 DIFF。

用串操作编写的程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
STRING1     DB'SDFASDGDHHFJH…'
COUNT       EQU $ - TRING1
STRING2     DB'WRFERGHRHTYJU…'
DIFF        DW 0
```

```
data        ends
code        segment
start       proc far
            assume ss:stack,cs:code,ds:data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV ES,AX
            MOV SI,OFFSET STRING1
            MOV DI,OFFSET STRING2
            MOV CX,COUNT
            CLD
            REPE CMPS STRING1,STRING2
            MOV DIFF,-1
            JE SAME
            DEC SI
            MOV DIFF,SI
SAME:       ret
start       endp
code        ends
            end start
```

不用串操作编写的程序段如下。

```
            MOV BX,0
            MOV CX,COUNT
AGAIN:      MOV AL,STRING1[BX]
            CMP AL,STRING2[BX]
            JNE DIF
            INC BX
            LOOP AGAIN
            MOV DIFF,-1
            RET
DIF:        ADD BX,OFFSET STRING1
            MOV DIFF,BX
            ⋮
```

例 3.29 用串搜索指令编制“镜子”程序。

例 2.7 的“镜子”程序是利用输入并显示字符串和显示器输出字符串,即 10 号和 9 号两个系统功能调用,并根据 10 号系统功能调用的出口参数计算求得输入字符串的末地址,再将 $ 的 ASCII 码送入字符串的尾部完成的。因输入字符串一定是以回车结束的,所以可以在输入并显示字符串后用串搜索指令在输入字符串的存储区内搜索回车的 ASCII 码 0DH,找到后将其换为 $ 的 ASCII 码 24H,再利用 9 号功能调用输出显示该输入的字符串,完成“镜子”功能。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
```

```
data        segment
OBUF        DB'>',0DH,0AH,'$'
IBUF        DB 255,0,255 DUP(0)
data        ends
code        segment
start       proc far
            assume ss: stack,cs: code,ds: data
            push ds
            sub ax,ax
            push ax
            mov ax,data
            mov ds,ax
            MOV ES,AX
            MOV AH,9
            MOV DX,OFFSET OBUF
            INT 21H
            MOV AH,10
            MOV DX,OFFSET IBUF
            INT 21H
            MOV AL,0DH
            MOV CX,255
            MOV DI,OFFSET IBUF + 2
            CLD
            REPNZ SCASB
            DEC DI
            MOV BYTE PTR [DI],'$'
            MOV AH,2
            MOV DL,0AH
            INT 21H
            MOV DX,OFFSET IBUF + 2
            MOV AH,9
            INT 21H
            ret
start       endp
code        ends
            end start
```

3.5 子程序设计

子程序设计是程序设计中最主要的方法与技术之一。本节主要介绍子程序的概念、主程序与子程序之间的连接及参数传递方式，子程序设计的基本方法和调用方法。

3.5.1 子程序的概念

循环程序设计技术解决了同一程序中连续多次有规律重复执行某个或某些程序段的问题。但对于无规律的重复就不能用循环程序实现。更多的情况是在不同的程序中或在同一个程序的不同位置常常要用到功能完全相同的程序段，如数制之间的转换、代码转换、初等

函数计算等。对于这样的程序段,为避免编制程序的重复劳动,节省存储空间,往往把它独立出来,附加少量额外的指令,将其编制成可供反复调用的公用的独立程序段,并通过适当的方法把它与其他程序段连接起来。这种程序设计的方法称为子程序设计,被独立出来的程序段称为子程序。调用子程序的程序称为主程序或调用程序。主程序与子程序是相对的。如程序X调用程序Y,程序Y又调用程序Z,那么程序Y对于程序X来说是子程序,而对于程序Z来说,则是主程序。称进入子程序的操作为子程序调用。每次调用后,就进入子程序运行,运行结束后回到主程序的调用处继续执行。称子程序返回到主程序的操作为子程序的返回。上述的X、Y、Z三个程序之间的调用和返回关系如图3-12所示。

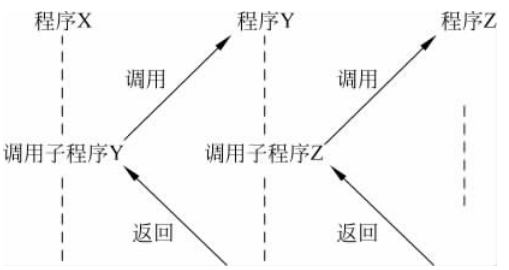

图3-12　子程序的调用和返回

子程序设计是使程序模块化的一种重要手段。当设计一个比较复杂的程序时,根据程序要实现的若干主要功能及各功能要调用的公用部分,将程序划分为若干个相对独立的模块。确定各模块调用关系和参数传递方式,为各模块分配不同的名字(入口地址),然后把每个模块都编制成子程序,最后将这些模块根据调用关系连成一个整体。这样既便于分工合作,又可避免重复劳动,节省存储空间,提高程序设计的效率和质量,使程序整洁、清晰、易读、便于修改和扩充。

设计包含子程序的程序时,应解决的问题如下。

1. 主程序与子程序之间的转返

子程序的调用和返回实质上就是程序控制的转移,原则上用一般的转移指令即可完成,可事实上并不那么简单。对于主程序,在什么时刻,应从什么位置进入哪个子程序,事先是很清楚的,因此主程序调用子程序是可以预先安排的。但对于子程序,每次执行完应返回到哪个调用程序以及调用程序的什么位置,子程序是无法预先安排的。因为子程序不能预先知道哪个主程序什么时候在什么位置调用它,因此也无法知道执行完后返回到哪个主程序的什么位置。该位置与主程序的调用位置有关。所以主程序与子程序间的转返是子程序设计必须解决的一个问题,这个问题是通过调用指令和返回指令解决的。

2. 主程序与子程序间的参数传递

主程序与子程序相互传递的信息称为参数。主程序提供给子程序以便加工处理的信息称为入口参数,经子程序加工处理后回送给主程序的信息称为出口参数。每个子程序的功能虽然是确定的,但每次调用它所完成的具体工作和传递的结果一般是不同的,即主程序与子程序间传递的参数对每一次来说一般是不一样的。为了实现主程序与子程序间参数的传递,就要约定一种主程序和子程序双方都能接受的参数传递方法。传递的参数可以是信息本身,还可以是信息的地址,其基本方法有如下三种。

1）寄存器法

该法就是主程序与子程序间传递的参数都在约定的寄存器中。当所需传递的参数较少时，一般用这种方法。在调用子程序前主程序将入口参数送到约定寄存器中，子程序直接从这些寄存器中取得这些参数进行运算处理。经加工处理后得到的结果，即出口参数也放在约定的寄存器中，返回主程序后，主程序就从该寄存器中得到结果。

2）堆栈法

堆栈法是把主程序与子程序间传递的参数都放到堆栈中。在调用子程序前，入口参数由主程序送到堆栈中。子程序从堆栈中取得这些参数，并将处理结果送到堆栈中。返回主程序后，主程序从堆栈取得结果。

3）参数赋值法

参数赋值法是把参数存放在主程序的调用子程序指令后面的一串单元中。对于入口参数，一般是信息的地址，当入口参数很少时，也可以是信息本身。对于出口参数，一般是信息本身，当出口参数较多时，也可以是信息的地址。若给出的是信息本身，称直接赋值法；若给出的是信息的地址，称间接赋值法。调用子程序指令执行后用于返转的专用存储器中自动存入的地址，正好是第一个入口参数的地址，或是第一个入口参数地址的地址，子程序引用很方便。如果有 N 个参数，那么紧接第 N 个参数后面的那条指令的地址就是返回地址。

还有一些传递参数的方法，如约定存储单元法。到底采用什么方法传递参数要根据具体情况而定。有时是几种方法混合使用。

以上所述是指主程序和子程序之间有参数传递的情况。也有的主程序和子程序之间无参数传递，子程序只是按规定完成某种功能操作，此时，自然不考虑参数的传递问题。

3. 主程序和子程序公用寄存器的问题

子程序不可避免地要使用一些寄存器，因此子程序执行后，某些寄存器的内容会发生变化，如果主程序在这些寄存器中已经存放了有用的信息，则从子程序返回主程序后，主程序的运行势必因原存信息被破坏而出错。解决这个问题的方法是在使用这些不能被破坏的寄存器之前，将其内容保存起来，使用之后再将其还原。前者称为保护现场，后者称为恢复现场。

保护现场与恢复现场的操作可以在主程序中完成，也可以在子程序中完成。一般情况下是在子程序中完成，其方法是在子程序的开始，将子程序要用到的寄存器的内容都保存起来，在子程序返回主程序之前再恢复这些寄存器的内容。保存和恢复操作可以通过进栈指令和出栈指令实现。如某子程序要用 AX、BX、CX、DX 4 个寄存器，则该子程序的保护现场和恢复现场的具体程序段如下。

```
PUSH AX
PUSH CX
PUSH DX
PUSH BX
 ┆
POP BX
POP DX
POP CX
POP AX
```

凡子程序用到的不是携带入口参数的寄存器,包括段寄存器(CS除外)一般都应保护。携带入口参数的寄存器,若问题无特殊要求,则不必保护。携带出口参数的寄存器,一般不能保护。

由上述设计包含子程序的程序时应解决的问题可知,子程序一般有如下结构:首先保护现场;其次取入口参数进行加工处理,并将处理结果送出口参数约定的寄存器或存储单元保存;然后恢复现场;最后返回主程序。

3.5.2 子程序的调用指令与返回指令

为了方便地实现子程序的调用与返回,80x86专门设计了子程序的调用指令和返回指令。主程序通过调用指令对子程序进行调用,子程序执行完毕用返回指令返回到主程序的调用处。

调用指令有直接调用和间接调用两类,通常都使用直接调用。当子程序的调用因条件不同而有所选择,且只允许使用一条调用指令时,才使用间接调用。

1. 直接调用指令

指令格式

```
CALL target
```

其中,操作数target是子程序的标号即子程序的入口地址。直接调用指令的功能是将返回地址进栈保存后将程序控制转移到子程序target。

80x86允许子程序与调用它的主程序在同一代码段,此时target一般属于NEAR类型,这种调用方式称为段内调用;也允许子程序与调用它的主程序在不同的代码段,此时target一般属于FAR类型,这种调用方式称为段间调用。

段内调用指令执行后CS内容不改变,只改变IP的内容,而段间调用指令执行后,CS和IP的内容都要变。因此它们将返回地址进栈保存的操作有差别,段内调用只需要将IP进栈保存;而段间调用却要将CS和IP都进栈保存。

段内调用的具体操作如下。

```
[SP-2]←IPL、[SP-1]←IPH
SP←SP-2
IP←OFFSET target
```

段间调用的具体操作如下。

```
[SP-2]←CSL、[SP-1]←CSH
[SP-4]←IPL、[SP-3]←IPH
SP←SP-4
CS←SEG target
IP←OFFSET target
```

2. 间接调用指令

指令格式

```
CALL dest
```

间接调用指令的功能是将返回地址保存后将目的操作数的内容送IP或CS和IP,实现

程序转移到子程序。

间接调用也有段内和段间两类调用。间接段内调用指令的目的操作数可为寄存器和存储器。所执行的操作是将 IP 进栈保存后,将寄存器或字存储变量的内容送 IP。即:

$[SP-2]\leftarrow IP_L$、$[SP-1]\leftarrow IP_H$
$SP\leftarrow SP-2$
$IP\leftarrow$REG16/MEM16

间接段间调用指令的目的操作数为存储器。所执行的操作是 CS 和 IP 进栈保存后,将双字存储变量的内容送 CS 和 IP。即:

$[SP-2]\leftarrow CS_L$、$[SP-1]\leftarrow CS_H$
$[SP-4]\leftarrow IP_L$、$[SP-3]\leftarrow IP_H$
$SP\leftarrow SP-4$
$CS\leftarrow$MEM32 + 2
$IP\leftarrow$MEM32

3. 返回指令

指令格式

RET [N] (N 为正偶数,可省略)

指令的功能是将程序控制返回到主程序。段内返回和段间返回的符号指令的形式是一样的,都是 RET,它们的差别在于机器指令不同。段内返回即近返回的机器指令为 C3H;而段间返回即远返回(反汇编时给出的符号指令是 RETF)的机器指令为 CBH。

段内返回的操作是:

$IP_L\leftarrow[SP]$、$IP_H\leftarrow[SP+1]$
$SP\leftarrow SP+2$;

段间返回的操作是:

$IP_L\leftarrow[SP]$、$IP_H\leftarrow[SP+1]$
$CS_L\leftarrow[SP+2]$、$CS_H\leftarrow[SP+3]$
$SP\leftarrow SP+4$

带有正偶数 N 的返回指令的操作 SP 还要多加 N,加 N 的目的是废除栈中 N/2 个无用字。用堆栈法传递参数的子程序常用带有正偶数的返回指令返回主程序。

3.5.3 子程序及其调用程序设计

下面用几个实例说明如何解决设计包含子程序的程序应解决的三个问题。在设计包含子程序的程序之前应先明确两个问题:一是子程序所处的位置,子程序与调用它的主程序是同一个模块,还是分属两个模块;在同一模块时还要明确是在同一代码段,还是在不同的代码段。二是子程序与主程序的参数传递问题。

先看两个常用的数制转换子程序的编制方法。

例 3.30 编制将标准设备输入的一串十进制数的 ASCII 码(如键盘输入的十进制数)转换为 16 位二进制数的子程序。

入口参数:DS:SI←待转换十进制数的 ASCII 码的首地址

CX←ASCII十进制数的位数

出口参数：AX←转换结果，即16位二进制数

方法1　调用子程序的主程序与子程序在同一模块同一代码段，子程序过程应定义为NEAR过程，与主程序过程并列放在同一代码段中。子程序过程如下。

```
ABCDCB      PROC
            MOV AX, 0
ABCDC1:     PUSH CX
            MOV CX, 10                  ; Xi×10+Xi-1
            MUL CX
            AND BYTE PTR[SI], 0FH
            ADD AL, [SI]
            ADC AH, 0
            INC SI
            POP CX
            LOOP ABCDC1
            RET
ABCDCB      ENDP
```

方法2　子程序与主程序在同一模块但不在同一代码段，子程序应定义为FAR过程。子程序代码段如下。

```
SUBCODE     SEGMENT
            ASSUME CS: SUBCODE
ABCDCB      PROC FAR
            MOV AX, 0
ABCDC1:     PUSH CX
            MOV CX, 10                  ; Xi×10+Xi-1
            MUL CX
            AND BYTE PTR[SI], 0FH
            ADD AL, [SI]
            ADC AH, 0
            INC SI
            POP CX
            LOOP ABCDC1
            RET
ABCDCB      ENDP
SUBCODE     ENDS
```

方法3　子程序与主程序各自独立成模块。由于主程序和子程序不在同一模块，所以要用到模块通信伪指令PUBLIC和EXTRN。

说明公共符号伪指令PUBLIC的格式

```
PUBLIC 符号[,符号,…]
```

其功能是用来说明其后的符号是公共符号，可以被其他模块调用。该伪指令用于子模块。

说明外部符号伪指令EXTRN的格式

```
EXTRN 符号: 类型[,符号: 类型,…]
```

其功能是用来说明其后的符号是外部符号及该符号的类型。这些外部符号必须在它定义的模块中被说明是公共符号，符号的类型必须与它们原定义时的类型一致。该伪指令用于主

模块。

子模块和主模块如下。

```
PUBLIC      ABCDCB
CODE        SEGMENT
            ASSUME CS: CODE
ABCDCB      PROC FAR
            MOV AX, 0
ABCDC1:     PUSH CX
            MOV CX, 10                  ; Xi×10 + Xi - 1
            MUL CX
            AND BYTE PTR [SI], 0FH
            ADD AL, [SI]
            ADC AH, 0
            INC SI
            POP CX
            LOOP ABCDC1
            RET
ABCDCB      ENDP
CODE        ENDS
            END
EXTRN       ABCDCB: FAR
stack       segment stack 'stack'
            ⋮
code        segment
begin       proc far
            ⋮
            CALL ABCDCB
            ⋮
begin       endp
code        ends
            end begin
```

例 3.31 编制将 16 位二进制补码数转换为可供标准输出设备输出的十进制数的 ASCII 码(如用于显示器显示的十进制数)子程序。

入口参数：AX←待转换的二进制数。

ES：DI←转换后的十进制数的 ASCII 码存放首地址

```
BCABCD      PROC
            PUSH AX
            PUSH BX
            PUSH CX
            PUSH DX
            PUSH DI
            OR AX,AX                    ; 判断数的符号
            JNS PLUS
            MOV BYTE PTR ES: [DI],'-'   ; 为负,送负号至输出数据区,
            INC DI                      ; 并求该负数的绝对值
            NEG AX
PLUS:       MOV CX,0                    ; 将 AX 中的二进制数转换
```

```
            MOV BX,10                ; 为十进制数
LOP1:       MOV DX,0
            DIV BX
            PUSH DX                  ; 余数进栈
            INC CX                   ; 十进制数位数加1
            OR AX,AX                 ; 商不为0继续除以10
            JNZ LOP1
LOP2:       POP AX                   ; 将十进制数转换为ASCII码
            ADD AL,30H
            MOV ES:[DI],AL
            INC DI
            LOOP LOP2
            MOV AL,'$'
            MOV ES:[DI],AL
            POP DI
            POP DX
            POP CX
            POP BX
            POP AX
            RET
BCABCD      ENDP
```

例 3.32 调用例 3.30 和例 3.31 的子程序编制十进制数运算的加(或减、或乘、或除)法程序。要求显示“>”后输入算式“加数1+加数2”,经过运算再显示“=结果”。设两个加数和结果的范围均为-32 768～32 767。为使编程简单,正数输入和输出时均不带“+”号,负数前带“-”号,即使运算符“+”与符号“-”相连也不将负号与数字置于括号内。

该程序首先要接收从键盘输入的加法算式存入字节数据区 BUF 中。BUF+1 存放着算式的字符数,从 BUF+2 开始存放算式中各字符的 ASCII 码。然后要将两个加数各自转换成二进制数相加。最后将和转换为 ASCII BCD 数显示输出。ASCII BCD 数与二进制数间的转换调用例 3.29 和例 3.30 的子程序 ABCDCB 和 BCABCD 完成。因为本程序可在一个数据段内完成,所以转换后的十进制数的 ASCII 码的存放地址改由 DS:DI 间址。主程序的大部分指令是在求取两个加数各自的位数及偏移首地址,以便调用 ABCDCB 子程序。求取的方法是:从 BUF+2 开始至“+”号是第一加数,往后至 0DH 是第二加数;BUF+1 中的字符数去掉第一个加数和运算符即是第二个加数的位数。对于每个加数均将其绝对值转换为二进制数。若是负数则将转换后绝对值求补,得其补码。程序如下。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
data        segment
IBUF        DB 14,0,14 DUP(0)
OBUF        DB'=',7 DUP(0)
data        ends
code        segment
            assume cs:code,ss:stack,ds:data
begin       proc far
            push ds
            sub ax,ax
```

```
        push ax
        mov ax,data
        mov ds,ax
        MOV DL,'>'                  ; 显示提示符">"
        MOV AH,2
        INT 21H
        MOV DX,OFFSET IBUF          ; 输入算式
        MOV AH,10
        INT 21H
        MOV DL,0AH                  ; 换行
        MOV AH,2
        INT 21H
        MOV SI,OFFSET IBUF + 2      ; SI 指向输入算式的首址
        CMP BYTE PTR[SI],'-'        ; 判断第一加数的符号,并进栈保存
        PUSHF
        JNE NS1
        INC SI                      ; 为负,指针指向第一加数的数字位
        DEC IBUF + 1                ; 实际字符数减 1(符号)
NS1:    MOV CX,0
        PUSH SI                     ; 第一加数首址进栈
CONT:   CMP BYTE PTR[SI],'+'        ; 求得第一加数的位数
        JE DONE
        INC SI
        INC CX
        JMP CONT
DONE:   POP SI
        PUSH CX
        CALL ABCDCB                 ; 将第一加数的绝对值转换为二进制数
        POP CX
        POPF                        ; 恢复第一加数的符号所置的 Z 标志
        JNZ NNEG1
        NEG AX                      ; 为负则求补
NNEG1:  MOV BX,AX                   ; 暂存加数 1
        INC SI                      ; 跳过运算符指向第二加数
        DEC IBUF + 1                ; 实际字符数减 1(运算符)
        CMP BYTE PTR[SI],'-'
        PUSHF
        JNE NS2
        INC SI
        DEC IBUF + 1
NS2:    MOV AL,IBUF + 1             ; 求取第二加数的位数,并送 CX
        MOV AH,0
        SUB AX,CX
        XCHG AX,CX
        CALL ABCDCB
        POPF
        JNZ NNEG2
        NEG AX
NNEG2:  ADD AX,BX                   ; 两数相加
        MOV DI,OFFSET OBUF + 1      ; 建立结果的 ASCII BCD 数存放地址指针
        CALL BCABCD
```

```
                MOV DX,OFFSET OBUF
                MOV AH,9
                INT 21H
                RET
BEGIN           ENDP
ABCDCB          PROC
                MOV AX,0
ABCDC1:         PUSH CX
                MOV CX,10
                MUL CX
                AND BYTE PTR[SI], 0FH
                ADD AL,[SI]
                ADC AH,0
                INC SI
                POP CX
                LOOP ABCDC1
                RET
ABCDCB          ENDP
BCABCD          PROC
                OR AX,AX
                JNS PLUS
                MOV BYTE PTR[DI],'-'
                INC DI
                NEG AX
PLUS:           MOV CX,0
                MOV BX,10
LOP1:           MOV DX,0
                DIV BX
                PUSH DX
                INC CX
                OR AX,AX
                JNZ LOP1
LOP2:           POP AX
                ADD AL,30H
                MOV [DI],AL
                INC DI
                LOOP LOP2
                MOV BYTE PTR[DI],'$'
                RET
BCABCD          ENDP
code            ends
                end begin
```

上面三个例子都是用寄存器法传递参数。下面介绍堆栈法和参数赋值法传递参数的子程序的设计方法。假定主程序和子程序在同一模块同一代码段。

例 3.33 编制求某数据区中无符号字数据最大值的子程序及调用它的主程序。

方法 1 堆栈法——参数都通过堆栈传递、子程序存取参数都由 BP 间址。高级语言调用汇编语言子程序常用此法。

```
stack           segment stack 'stack'
```

```
          dw 32 dup(0)
stack     ends
data      segment
BUF       DW 63,76,857,829,323,66,21,888
COUNT     = ( $ - BUF)/2
SMAX      DW 0
data      ends
code      segment
begin     proc far
          assume ss: stack,cs: code,ds: data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
          MOV AX,OFFSET BUF          ; 入口参数进栈
          PUSH AX
          MOV AX,COUNT
          PUSH AX
          CALL MAX
          POP SMAX                   ; 最大值出栈,送 SMAX
          ret
begin     endp
MAX       PROC
          PUSH BP
          MOV BP,SP
          PUSH SI
          PUSH AX
          PUSH BX
          PUSH CX
          PUSHF
          MOV SI,[BP + 6]            ; BUF 的偏移地址送 SI
          MOV CX,[BP + 4]            ; COUNT 送 CX
          MOV BX,[SI]                ; 取第一个数据至 BX 中
          DEC CX                     ; 字数据个数减 1
MAX1:     ADD SI,2                   ; 指向下一个字数据
          MOV AX,[SI]                ; 取一个字数据至 AX 中
          CMP AX,BX
          JNA NEXT                   ; AX 不高于 BX,与下一个比较
          XCHG AX,BX                 ; AX 高于 BX,则将较大字数据送 BX
NEXT:     LOOP MAX1
          MOV [BP + 6],BX            ; 最大值存入堆栈
          POPF
          POP CX
          POP BX
          POP AX
          POP SI
          POP BP
          RET 2                      ; 返回后 SP 指向最大值
MAX       ENDP
code      ends
          end begin
```

下述5条指令执行之前或之后:①CALL MAX之前;②CALL MAX之后;③保护现场之后;④恢复现场之后;⑤RET 2之后,堆栈存储区中的有关内容及SP的变化如图3-13所示。

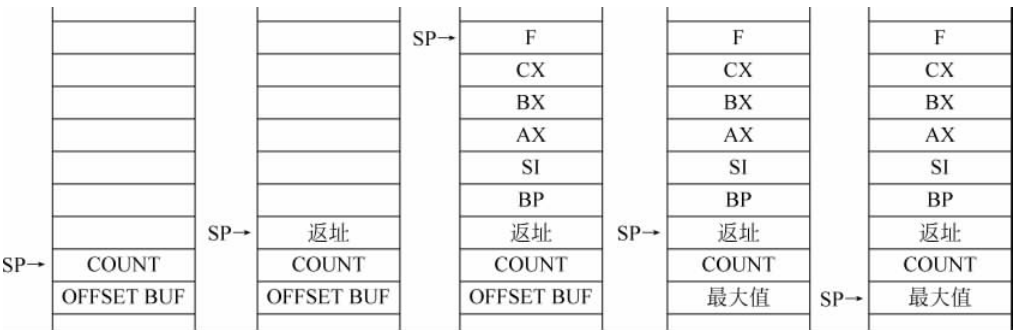

图3-13　例3.33图1

方法2　参数赋值法——将参数存放到CALL指令后的一串单元中,子程序通过返回地址存取参数并修改返回地址。代码段中的程序如下,传递的参数如图3-14所示。

图3-14　例3.33图2

```
stack     segment stack 'stack'
          dw 32 dup(0)
stack     ends
data      segment
BUF       DW 63,76,857,829,323,66,21,888,……
COUNT     = ( $ —BUF)/2
data      ends
code      segment
begin     proc far
          assume ss: stack,cs: code,ds: data
          push ds
          sub ax,ax
          push ax
          mov ax,data
          mov ds,ax
          CALL MAX
          DW BUF,COUNT
SMAX      DW 0
          ret
begin     endp
MAX       PROC
          MOV BP,SP
          MOV DI,[BP]              ; 取返回地址
          MOV SI,CS: [DI]          ; 取 BUF 的偏移地址
          MOV CX,CS: [DI + 2]      ; 取 COUNT
          ADD DI,6                 ; 修改返址
          MOV [BP],DI              ; 修改后的返址存入堆栈
          MOV BX,0                 ; 求最大值并保存在 BX 中
```

```
MAX1:    MOV AX,[SI]
         ADD SI,2
         CMP AX,BX
         JNA NEXT
         XCHG AX,BX
NEXT:    LOOP MAX1
         MOV CS:[DI-2],BX          ;存最大值
         RET
MAX      ENDP
code     ends
         end begin
```

在参数的传递方法中,寄存器法最简单;堆栈法最节省存储单元;参数赋值法最直观。但它们各有不足,如寄存器法不能传递较多的参数,堆栈法和参数赋值法编程较麻烦。经验证明,参数不多时用寄存器法最适宜;参数较多时可用堆栈法或参数赋值法。

例 3.34 编制两个 64 位无符号二进制数相加的子程序及其调用程序。

假定子程序及其调用程序在同一代码段。本例说明用参数赋值法传递两个加数和结果本身及它们的存放地址两种方法的编程方法。

方法 1 用参数赋值法传递两个加数与结果的存放首地址。代码段中的程序如下,传递的参数如图 3-15 所示。

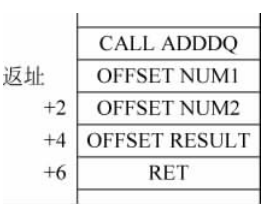

图 3-15 例 3.34 图 1

```
stack    segment stack 'stack'
         dw 32 dup(0)
stack    ends
data     segment
NUM1     DQ 7654321089ABCDEFH
NUM2     DQ 0FEDCBA9801234567H
RESULT   DT 0
data     ends
code     segment
start    proc far
         assume ss:stack,cs:code,ds:data
         push ds
         sub ax,ax
         push ax
         mov ax,data
         mov ds,ax
         CALL ADDDQ
         DW NUM1,NUM2,RESULT
         ret
start    endp
ADDDQ    PROC
         PUSH BP
         MOV BP,SP
         PUSH AX
         PUSH BX
         PUSH CX
         PUSH DX
         PUSH SI
```

```
            PUSH DI
            PUSHF
            MOV BX,[BP+2]           ; 取返回地址
            MOV SI,CS:[BX]          ; 第一加数存放首地址
            MOV DX,CS:[BX+2]        ; 第二加数存放首地址
            MOV DI,CS:[BX+4]        ; 结果存放首地址
            XCHG BX,DX
            MOV CX,4
            CLC
AGAIN:      MOV AX,[SI]             ; 取 NUM1 中 16 位二进制数
            ADC AX,[BX]             ; 加上 NUM2 中 16 位二进制数
            MOV [DI],AX             ; 存入 RESULT 中
            INC SI                  ; 修改指针
            INC SI
            INC DI
            INC DI
            INC BX
            INC BX
            LOOP AGAIN
            MOV WORD PTR[DI],0      ; 清除结果的最高字存放单元
            RCL WORD PTR[DI],1      ; 将向最高字的进位移入
            ADD DX,6                ; 修改返回地址
            MOV [BP+2],DX           ; 修改后的返址存入堆栈
            POPF
            POP DI
            POP SI
            POP DX
            POP CX
            POP BX
            POP AX
            POP BP
            RET
ADDDQ       ENDP
code        ends
            end start
```

方法 2　用参数赋值法直接传递两个加数与结果。代码段中的程序、两个加数和结果的存放形式如图 3-16 所示。

```
stack       segment stack 'stack'
            dw 32 dup(0)
stack       ends
code        segment
start       proc far
            assume ss:stack,cs:code
            push ds
            sub ax,ax
            push ax
            CALL ADDDQ
NUM1        DQ 7654321089ABCDEFH
NUM2        DQ 0FEDCBA9801234567H
```

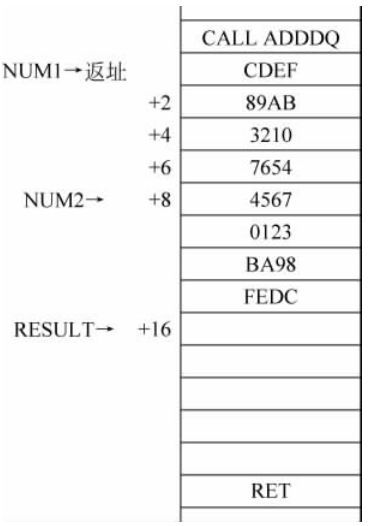

图 3-16　例 3.34 图 2

```
RESULT      DT0
            ret
start       endp
ADDDQ       PROC
            PUSH BP
            MOV BP,SP
            PUSH AX
            PUSH BX
            PUSH CX
            PUSHF
            MOV BX,[BP + 2]             ; 取返回地址,即 NUM1 的偏移地址
            MOV CX,4
            CLC
AGAIN:      MOV AX,CS: [BX]             ; 取 NUM1 中 16 位二进制数
            ADC AX,CS: [BX + 8]         ; 加上 NUM2 中 16 位二进制数
            MOV CS: [BX + 16],AX        ; 存入 RESULT 中
            INC BX
            INC BX
            LOOP AGAIN
            PUSHF
            ADD BX,16                   ; 修改指针指向结果的最高字
            POPF
            MOV WORD PTR CS: [BX],0
            RCL WORD PTR CS: [BX],1
            ADD BX,2                    ; 修改指针指向返回地址(RET 指令的地址)
            MOV [BP + 2],BX             ; 修改后的返回地址存入堆栈
            POPF
            POP CX
            POP BX
            POP AX
            POP BP
            RET
```

```
ADDDQ       ENDP
code        ends
            end start
```


3.6 宏功能程序设计

对于程序中经常要使用的独立功能的程序段,可以将它设计成子程序的形式,供需要时调用。但使用子程序需要付出一些额外的开销。有些简单功能的重复,若也将其设计成子程序,则额外开销就可能会超过执行功能的指令。为了达此目的,又减少额外开销,IBM PC宏汇编语言提供了宏指令功能。宏指令是用户为重复指令序列定义的名字。宏指令名像指令操作助记符一样,定义后便可在程序中用这个名字代替这个指令序列,并允许传递参数,参数传递方式也比子程序简单。同时用户还可以将经常使用的宏指令集中在一起建立宏库,当程序中需要调用宏库中的宏指令时,不必另外定义,只需按规定的条件调用即可。IBM PC宏汇编还提供了条件汇编,条件汇编可使宏汇编有选择地汇编源程序。条件汇编与宏指令结合,使宏指令功能更强,技术更完善。

3.6.1 宏指令

宏指令的调用要经过宏定义、宏调用和宏扩展三个步骤。宏定义和宏调用由用户自己完成,宏扩展由宏汇编程序在汇编期间完成。

1. 宏定义

宏指令的定义是使用伪指令MACRO和ENDM来实现的。

```
宏定义的格式   宏指令名   MACRO [形参][,形参,…]
                          ⋮
                        ENDM
```

宏定义必须由伪指令MACRO开始,ENDM结束。MACRO和ENDM间的程序段称为宏体。用MACRO和ENDM规定宏指令名,宏体和形参为宏定义。

宏指令名的构成同符号,它可以和指令助记符、伪指令相同,以便重新定义指令和伪指令的功能。它用于产生一条宏指令,使之和指令助记符一样出现在源程序中。形参是临时变量的名字,可以出现在宏体中的任何地方。形参可有可无,个数不限,各形参间用逗号或空格隔开。

宏体是宏指令代替的程序段,它由一系列指令和伪指令组成。

宏指令一定要先定义后调用。因此,宏定义一定要放在它的第一次调用之前。

宏指令名可以与指令助记符、伪指令同名,且具有比指令、伪指令更高的优先权,即当它们同名时,宏汇编程序则将它们一律处理成相应的宏扩展,而不管与它同名的指令或伪指令原来的功能如何。如要恢复原功能,可用取消宏定义伪指令PURGE。

伪指令PURGE的格式

```
PURGE   宏指令名[,宏指令名,…]
```

定义宏指令的最简单形式是用来缩写一段程序。例如,某程序中要经常运行9号功能调用和10号功能调用。9号功能调用和10号功能调用的指令完全相同,只是数据区的首

址和功能调用号不同。可以将 9 号功能调用和 10 号功能调用定义成一条宏指令，将数据区首地址和功能调用号选定为形式参数。其宏定义为：

```
WRD     MACRO A,B
        MOV DX,OFFSET B
        MOV AH,A
        INT 21H
        ENDM
```

其中，形参 A 为功能调用号，形参 B 为数据区的变量名。

形式参数不仅可以出现在指令的操作数部分，也可以出现在指令的助记符部分。若形参前面有字符，则还应在形参前面加上符号“&”。如将任一寄存器或存储器进行算术右移、逻辑右移或算术/逻辑左移，因这三条指令的助记符的第一字符相同，其后的字符不同，故可将不同的字符部分定义为形参，而在第一字符后加符号“&”。若将这三条指令助记符中不同字符部分定义为形参 A，寄存器或存储器定义为形参 B，移位次数定义为 C，则该宏定义为：

```
SHIFT     MSCRO A,B,C
          MOV CL,C
          S&A B,CL
          ENDM
```

2. 宏调用和宏扩展

宏指令名在源程序中的出现称为宏调用。

宏调用的格式

```
宏指令名[实参][,实参,…]
```

宏指令名必须与宏定义中的宏指令名一致，其后的实参可以是数字、字符串、符号名或尖括号括起来的带间隔符的字符串。实参在顺序、属性、类型上要同形参保持一致，个数与形参一般应相等。

实参代替形参并将宏体插到宏调用处称为宏扩展。宏扩展出的指令前用符号“+”标志。

宏调用 WRD 9,BUF1 的宏扩展是：

```
+         MOV DX,OFFSET BUF1
+         MOV AH,9
+         INT 21H
```

宏调用 WRD 10,BUF2 的宏扩展是：

```
+         MOV DX,OFFSET BUF2
+         MOV AH,10
+         INT 21H
```

宏调用 SHIFT AL,AX,4 的宏扩展是：

```
+         MOV CL,4
+         SAL AX,CL
```

宏调用 SHIFT HR,DX,6 的宏扩展是:

```
+          MOV CL,6
+          SHR DX,CL
```

宏调用 SHIFT AR,DX,10 的宏扩展是:

```
+          MOV CL,10
+          SAR DX,CL
```

3. 宏体中的标号和变量

若宏体中定义了标号或变量,当该宏指令被多次调用时,就会产生多个重名的标号或变量,造成多重定义的错误。如有以下宏定义:

```
CHANG     MACRO
          CMP AL,10
          JB NADD7
          ADD AL,7
NADD7:    ADD AL,30H
          ENDM
```

该宏定义的功能是将 AL 中的一位十六进制数(高 4 位为 0)转换为 ASCII 码。若某程序对此宏定义做两次调用:

```
          ⋮
          CHANGE
          ⋮
          CHANGE
          ⋮
```

宏汇编程序在汇编这两条宏指令时,所得到的宏扩展为:

```
                    ⋮
+                   CMP AL,10
+                   JB NAAD7
+                   ADD AL,7
+ NADD7:            ADD AL,30H
                    ⋮
+                   CMP AL,10
+                   JB NADD7
+                   ADD AL,7
+ NADD7:            ADD AL,30H
                    ⋮
```

标号 NADD7 的定义出现重复。为解决这个问题,宏汇编提供了伪指令 LOCAL。

伪指令 LOCAL 的格式

```
LOCAL 符号表
```

LOCAL 的功能是,在宏扩展时,宏汇编将按其调用顺序和符号表中的符号顺序自动为其后的符号指定特殊的符号(?? 0000～?? FFFF),并用这些特殊的符号替换宏体中相应的符号,从而避免了符号多重定义的错误。LOCAL 伪指令只能用在宏定义中,且必须是宏体中指令或伪指令的第一条。上面的宏定义可修改成以下形式。

```
CHANG     MACRO
          LOCAL NADD7
          CMP AL,10
          JB NADD7
          ADD AL,7
NADD7:    ADD AL,30H
          ENDM
```

两次宏调用后的宏扩展为：

```
                ⋮
+               CMP AL,10
+               JB ??0000
+               ADD AL,7
+ ??0000:      ADD AL,30H
                ⋮
+               CMP AL,10
+               JB ??0001
+               ADD AL,7
+ ??0001:      ADD AL,30H
                ⋮
```

4. 宏指令与子程序的区别

宏指令与子程序都可以用来处理程序中重复使用的程序段，缩短源程序的长度，使源程序结构简洁、清晰。但它们是两个完全不同的概念，有着本质的区别。其区别如下。

(1) 处理的时间和方式不同。宏指令是在汇编期间由宏汇编程序处理的；宏调用是用宏体置换宏指令名、实参置换形参；汇编结束，宏定义也随之消失。而子程序是目标程序运行期间由 CPU 直接执行，子程序调用不发生代码和参数的置换，是 CPU 将控制由主程序转向子程序。

(2) 目标程序的长度和执行速度不同。对每一次宏调用都要进行宏扩展，因而使用宏指令会导致目标程序长，占用内存空间大。而子程序的调用，无论多少次，子程序的目标代码只会出现一次，因此目标程序短，占用内存空间小。由于子程序调用需要 CPU 现实转返、需要保护现场和恢复现场、需要指令传递参数，而宏指令不存在这些额外的开销，因此宏指令的执行速度较子程序快。

(3) 参数传递的方式不同。宏调用可以实现参数的代换，代换方法简单、方便、灵活，参数的形式也不受限制，可以是指令、寄存器、标号、变量、常量等。子程序传递的参数一般为地址或数据，传递方式由用户编程时具体安排，比较麻烦。

究竟是采用宏指令还是子程序，要权衡内存空间、执行速度、参数的多少。当重复的程序段不长时，速度是主要矛盾，通常用宏指令。而当程序段较长时，额外开销的时间就不明显了，节省内存空间是主要矛盾，则通常采用子程序。

3.6.2 条件汇编与宏库的使用

对于经常使用的宏指令，可将它们集中在一起，建成宏库供自己或他人随时调用。当程序中需要调用时，应使用伪指令 INCLUDE 将宏库加入自己的源文件中，然后按宏库中各宏指令的规定调用即可。

伪指令 INCLUDE 的格式

```
INCLUDE 宏库名
```

该伪指令的功能是将宏库进行汇编,将宏库汇编完后再继续汇编后面的程序。由于宏指令要先定义后调用,所以该伪指令应放在源文件的首部。

宏汇编程序汇编源程序时要经过两遍扫描,第一遍是宏处理,第二遍是生成机器代码。因此第二遍可跳过宏定义,加快汇编速度。使用条件汇编即可达到此目的,其使用格式是:

```
IF1
        INCLUDE  宏库名
    ENDIF
```

上述条件汇编使得第一遍扫描时将宏库调入,以便完成宏扩展。第二遍扫描将跳过INCLUDE,从而加快汇编速度。

宏库中有多个宏定义,一个程序中不一定全部用得上。对于用不着的宏定义可以用伪指令 PURGE 取消,一旦取消宏汇编就不会处理它们,这样也加快了汇编的速度。

条件汇编还有其他应用,但在宏指令之外,仅有简单的应用,只有与宏指令结合起来使用,才显示出它的优越性。所以它的详细使用本书就不介绍了。

INCLUDE 伪指令可将任何文本文件加入源程序一起汇编,使用格式是用加入源程序的文本文件代替宏库名。但不能像宏库那样将加入操作限制在第一次扫描中。

3.6.3 宏功能程序设计举例

利用编辑程序建立宏库 MACRO.LIB 如下。

```
INIT    MACRO CSNAME SSNAME DSNAME ESNAME     ; 初始化
        ASSUME CS: CSNAME,SS: SSNAME,DS: DSNAME,ES: ESNAME
        PUSH DS
        SUB AX,AX
        PUSH AX
        MOV AX,DSNAME
        MOV DS,AX
        MOV AX,ESNAME
        MOV ES,AX
        ENDM
STACK0  MACRO A                               ; 定义堆栈段
STACK   SEGMENT STACK 'STACK'
        DW A DUP (0)
STACK   ENDS
        ENDM
WRD     MACRO B1,B2                           ; 9 号或 10 号功能调用
        MOV DX,OFFSET B2
        MOV AH,B1
        INT 21H
        ENDM
OUT     MACRO C                               ; 2 号功能调用
        MOV DL,C
        MOV AH,2
        INT 21H
        ENDM
```

```
ABCDCB  MACRO                          ; 第 3.5.3 节例 3.30
        LOCAL ABCDC1
        MOV AX,0
ABCDC1: PUSH CX
        MOV CX,10
        MUL CX
        AND BYTE PTR[SI],0FH
        ADD AL,[SI]
        ADC AH,0
        INC SI
        POP CX
        LOOP ABCDC1
        ENDM
SHIFT   MACRO D1,D2,D3                 ; 将 D2 逻辑或算术移 D3 位
        MOV CL,D3
        S&D1 D2,CL
        ENDM
CHANGE  MACRO REG8                     ; 将 REG8 的低 4 位转换为 ASCII 码
        LOCAL NADD7
        AND REG8,0FH
        CMP REG8,10
        JB NADD7
        ADD REG8,7
NADD7:  ADD REG8,30H
        ENDM
AXSDI   MACRO                          ; 将 AX 右移 4 位后再将其低 4 位
        SHIFT HR,AX,4                  ; 转换为 ASCII 码送 ES:[DI]
        PUSH AX
        CHANGE AL
        STOSB
        POP AX
        ENDM
```

例 3.35 利用上述宏库编程将键盘输入的十进制数(范围为－32 768～32 767)转换为十六进制数,并从显示器输出。

```
IF1
        INCLUDE MACRO. LIB
ENDIF
        STACK0 32
data    segment
IBUF    DB 7,0,7 DUP(0)
OBUF    DB' = ',5 DUP(0)
data    ends
code    segment
begin   proc far
        INIT CODE,STACK,DATA,DATA
        OUT'>'
        WRD 10,IBUF                    ; 输入
        MOV CL,IBUF + 1                ; 输入字符数送 CX
        MOV CH,0
```

```
        MOV SI,OFFSET IBUF + 2              ; 建立输入字符的指针
        MOV AL,[SI]
        CMP AL,'-'                          ; 判断第一字符的符号是否为负
        PUSHF
        JNE PLUS
        DEC CX                              ; 若为负,调整字符个数和地址指针
        INC SI
PLUS:   ABCDCB
        POPF
        JNZ NNEG
        NEG AX                              ; 为负数则求其绝对值
NNEG:   STD
        MOV DI,OFFSET OBUF + 5              ; DI 指向输出数据区尾部
        PUSH AX
        MOV AL,'$'                          ; $ 的 ASCII 码送入
        STOSB
        POP AX
        PUSH AX
        CHANGE AL
        STOSB
        POP AX
        AXSDI
        AXSDI
        AXSDI
        OUT 0AH                             ; 显示器换行
        WRD 9,OBUF                          ; 输出换行后的十六进制数
        OUT 'H'
        ret
begin   endp
code    ends
        end begin
```

例 3.36 利用宏功能编制“镜子程序”。

```
IF1
        INCLUDE MACRO.LIB
ENDIF
        PURGE OUT,ABCDCB,SHIFT,CHANGE,AXSDI
        STACK0 32
data    segment
MEM1    DB'>',0DH,0AH,'$'
MEM2    DB 255,256 DUP(0)
data    ends
code    segment
begin   proc far
        INIT CODE,STACK,DATA,DATA
        WRD 9,MEM1
        WRD 10,MEM2
        MOV AX,OFFSET MEM2 + 2
        MOV BX,OFFSET MEM2 + 1
        ADD AL,[BX]
```

```
        ADC AH,0
        MOV BX,AX
        MOV BYTE PTR[BX],'$'
        WRD 9,MEM1+2
        WRD 9,MEM2+2
        ret
begin   endp
code    ends
        end begin
```

习题与思考题

3.1　写出执行下列程序段的中间结果和结果。

1.
```
MOV AX,0809H
MUL AH              ; AX = _________
AAM                 ; AX = _________
```

2.
```
MOV AX,0809H
MOV DL,5
AAD                 ; AX = _________
DIV DL              ; AX = _________
MOV DL,AH
AAM                 ; AX = _________,DL = _________
```

3.
```
MOV AX,0809H
ADD AL,AH
MOV AH,0            ; AX = _________
AAA                 ; AX = _________
```

4.
```
MOV AX,0809H
MOV DL,10
XCHG AH,DL
MUL AH              ; AX = _________
AAM                 ; AX = _________
ADD AL,DL           ; AX = _________
```

5.
```
MOV AL,98H
MOV AH,AL
MOV CL,4
SHR AH,CL
AND AL,0FH
AAD                 ; AL = _________ H
```

6.
```
    MOV CL,248
    XOR AX,AX
    MOV CH,8
AG: SHL CL,1
    ADC AL,AL
    DAA
    ADC AH,AH
    DEC CH
    JNZ AG
```

结果：AX=_____H

3.2 编写程序，将字节变量BVAR中的压缩BCD数转换为二进制数，并存入原变量中。

3.3 编写程序，求字变量W1和W2中的非压缩BCD数之差(W1－W2、W1≥W2)，将差存到字节变量B3中。

3.4 编写求两个4位非压缩BCD数之和，将和送显示器显示的程序。

3.5 编写求两个4位压缩BCD数之和，将和送显示器显示的程序。

3.6 编写程序，将字节变量BVAR中的无符号二进制数(0～FFH)转换为BCD数，在屏幕上显示结果。

3.7 用查表法求任一输入自然数N(0≤N≤40)的立方值送显示器显示，并将其存入一字变量中。

3.8 已知字变量WA中存放有4位十六进数a_3，a_2，a_1，a_0，现要求将a_i(i由键盘输入)存入字节变量BA的低4位，试编写该程序。

3.9 有一原码形式的双字符号数，试编制求其补码的程序。

3.10 设平面上一点P的直角坐标为(X，Y)，X，Y为字符号数，试编制若P落在第i象限内，则令k=i；若P落在坐标轴上，则令k=0的程序。

3.11 将键盘输入的8位二进制数以十六进制数形式在显示器上显示出来，试编写这一程序。

3.12 将键盘输入的十进制数(－128～127)转换为二进制数，以十六进制数形式在显示器上显示出来，试编写这一程序。

3.13 编写将字变量SW中的16位无符号二进制数以十制数形式送显示器显示的程序。

3.14 编程序将符号字数组ARRAYW中的正负数分别送入正数数组PLUS和负数数组MINUS，同时把“0”元素的个数送入字变量ZERON。

3.15 下列各数称为Fibonacci数：0，1，1，2，3，5，8，13…。这些数之间的关系是：从第三项起，每项都是它前面两项之和。若用a_i表示第i项，则有a_1=0，a_2=1，a_i=a_{i-1}+a_{i-2}，…试编写显示第24项Fibonacci数(两字节)的程序。

3.16 从键盘输入一字符串(字符数>1)，然后在下一行以相反的次序显示出来(采用9号和10号系统功能调用)。

3.17 编写求输入算式“加数1＋加数2”的和并送显示的程序(加数及其和均为4位BCD数)。

3.18 编写程序将某字节存储区中的10个非压缩的BCD数以相反的顺序送到另一个字节存储区中，并将这两个存储区中的数字串分两行显示出来。

3.19 编写ASCII码的查询程序。要求该程序运行后显示提示信息：“The ASCII code of”，待查询者输入欲查字符后再显示“is”和该字符的ASCII码，换行后又输出提示信息“The ASCII code of”待查，如此不断循环。直至查询者输入回车符输出“is 0DH”后结束该程序的运行。

3.20 编写将26个英文字母字符“ABCD…Z”存到字节变量中的程序。

3.21 已知BUF1中有N1个按从小到大的顺序排列互不相等的字符号数，BUF2中有

N2 个按从小到大的顺序排列互不相等的字符号数。试编写程序将 BUF1 和 BUF2 中的数合并到 BUF3 中，使在 BUF3 中存放的数互不相等且按从小到大的顺序排列。

3.22　在字节字符串 STR1 中搜索子串“AM”出现的次数送字变量 W。试编写其程序。

3.23　设有一稀疏数组 $a_i(i=1,2,\cdots,1000)$存放在字变量 BUFA 的存储区中，现要求将数组加以压缩，使其中的非 0 元素仍按序存放在 BUFA 存储区中，而 0 元素不再出现。试编写实现上述功能的程序。

3.24　源程序如下，阅读后做如下试题。

(1) 该程序共分为 5 部分，在分号后给这 5 部分加上注释。

(2) 列举具有代表性的 4 例，说明该程序的功能。

(3) 画出上述 4 例的数据存储图。

```
stack    segment stack 'stack'
         dw 32 dup(0)
stack    ends
data     segment
IBF      DB 5,0,5 DUP(0)
OBF      DB 9 DUP(0)
data     ends
code     segment
begin    proc far
         assume ss:stack,cs:code,ds:data
         push ds
         sub ax,ax
         push ax
         mov ax,data
         mov ds,ax
         MOV DX,OFFSET IBF            ;
         MOV AH,10
         INT 21H
         MOV BX,1                     ;
         MOV CH,IBF[BX]
         MOV CL,4
         XOR AX,AX
AGAIN:   INC BX
         SUB IBF[BX],30H
         CMP IBF[BX],0AH
         JB NS7
         SUB IBF[BX],7
NS7:     SHL AX,CL
         OR AL,IBF[BX]
         DEC CH
         JNZ AGAIN
         AND AH,AH                    ;
         JNZ NAP
         CBW
NAP:     MOV BX,OFFSET OBF + 8
         MOV BYTE PTR[BX], '$'
```

```
        MOV CX,10
        AND AX,AX
        PUSHF
        JNS NNEG
        NEG AX
NNEG:   AND AX,AX
        JZ JOUT
        MOV DX,0
        DEC BX
        DIV CX
        ADD DL,30H
        MOV [BX],DL
        JMP NNEG
JOUT:   POPF                          ;
        JNS PLUS
        DEC BX
        MOV BYTE PTR[BX], '-'
PLUS:   DEC BX
        MOV BYTE PTR[BX], '='
        DEC BX
        MOV BYTE PTR[BX],0AH
        MOV DX,BX                     ;
        MOV AH,9
        INT 21H
        ret
begin   endp
code    ends
        end begin
```

3.25　编制计算 N 个($N<50$)偶数之和(2+4+6+…)的子程序和接收输入 N 及将结果送显示的主程序。要求用以下三种方法编写：①主程序和子程序在同一代码段；②在同一模块但不在同一代码段；③各自独立成模块。

3.26　编写程序，实现接收输入的一串以逗号分隔的十进制正数(十进制数均小于10 000，个数不超过51个)，并将其中的最大值送显示器显示。要求把输入的ASCII码形式的十进制数转换为压缩BCD数、求十进制数的个数、求最大值的程序分别编写为子程序。

3.27　编写程序，接收输入的一串以逗号分隔的十进制符号数，并按从小到大的顺序显示出来(仍以逗号分隔)。

3.28　源程序如下，阅读后做如下试题。

(1) 在分号后给指令或(向下)给程序段加上注释。(实质是做什么？例如，第一个注释若注为“将2送BX”，则视为非实质注释，不给分。)

(2) 列举实例，说明该程序的功能。(输入什么？显示什么?)

(3) 画出实例的数据存储图。

```
stack   segment stack 'stack'
        dw 32 dup(0)
stack   ends
data    segment
IBUF    DB  255,0,255 DUP(0)
```

```
ABCD    DB 0AH,'ABCD: ', 255 DUP(0)
MNOPQ   DB 0AH,0DH,'MNOPQ: ', 255 DUP(0)
data    ends
code    segment
begin   proc far
        assume cs:code,ss:stack,ds:data
        push ds
        sub ax,ax
        push ax
        mov ax,data
        mov ds,ax
        MOV DX,OFFSET IBUF
        MOV AH,10
        INT 21H
        MOV BX,2                    ;
        MOV SI,OFFSET ABCD + 7
        MOV DI,OFFSET MNOPQ + 9
AG1:    CMP IBUF[BX],'- '           ;
        JNE P1
        CALL MP
AG2:    CMP IBUF[BX - 1],0DH        ;
        JE EXIT
        JMP AG1
P1:     XCHG SI,DI                  ;
        CALL MP
        XCHG SI,DI
        JMP AG2
EXIT:   MOV BYTE PTR[SI - 1],'$ '
        MOV BYTE PTR[DI - 1],'$ '
        MOV AH,9
        MOV DX,OFFSET ABCD
        INT 21H
        MOV DX,OFFSET MNOPQ
        INT 21H
        ret
begin   endp
MP      PROC                        ;
        MOV AL,IBUF[BX]
        MOV [DI],AL
        INC DI
        INC BX
        CMP IBUF[BX - 1],0DH
        JE  BACK
        CMP IBUF[BX - 1],','
        JNE MP
BACK:   RET
MP      ENDP
code    ends
        end begin
```

3.29　编写一宏定义 BXCHG,将一字节的高 4 位与低 4 位交换。

3.30 已知宏定义如下：

```
XCHG0    MACRO A,B
         MOV AL,A
         XCHG AL,B
         MOV A,AL
         ENDM
OPP      MACRO P1,P2,P3,P4
         XCHG0 P1,P4
         XCHG0 P2,P3
         ENDM
```

展开宏调用：OPP BH,BL,CH,CL。

3.31 编写以下宏定义。

(1) IO 实现 9 号和 10 号系统功能调用。

(2) CRLF 实现输出回车换行。

(3) STACKM 实现堆栈段定义。

将以上宏定义建立一个宏库，利用该库编写“镜子”程序。

3.32 数据区中有 4 个升序排列的四字节无符号数，试利用上述各题中的宏定义，编写程序将它们以降序输出。