第3章

数据运算与输入输出

本章学习用于数值计算的指令，通过若干实例程序的编制，帮助学生初步掌握汇编语言程序设计的基本方法。

3.1 学习要点及疑点

学习要点是：数值计算与标志位；指针的使用；循环程序基本结构；输入输出与数制转换。本章的指令难度并不高，但是要注意许多细节。例如除法指令的被除数和除数的存放位置以及它们的数据类型都是有一定之规的，如果不注意，就会在编程时屡屡出错。在学习这些指令时，不仅要了解其功能还应牢记一些具体的规定。

3.1.1 算术运算

算术运算指令包括加、减、乘、除等运算指令，这些指令的执行会影响标志位。

1. 加法指令

有 3 条基本的加法指令，如表 3-1 所示。

表 3-1 加法运算指令

指令名称	操作码助记符	指令格式	指令功能	主要用途
加法指令	ADD(Add)	SUB dest,src	dest←dest+src	加法
带进位的加法指令	ADC (Add with Carry Out)	ADC dest,src	dest←dest+src+cf	分段加法的高位数据相加
增量指令	INC(Increment)	INC dest	dest←dest+1	修改计数器和指针

对于 ADD 和 ADC 指令：

- 目的操作数：8/16/32 位的寄存器/存储器。
- 源操作数：与目的操作数同类型的寄存器/存储器/立即数。
- 两种指令执行后，CPU 的状态标志 CF,OF,ZF,SF,PF,AF 按照运算结果被刷新。

对于 INC 指令：

- 目的操作数：8/16/32 位的寄存器/存储器。
- 增量指令执行后，CPU 的状态标志 OF，ZF，SF，PF，AF 按照运算结果被刷新，但是 CF 标志不受影响。

例 3-1 设 X，Y，Z 为三个 16 位数，内存分配情况如图 3-1 所示，求 S＝X＋Y＋Z。

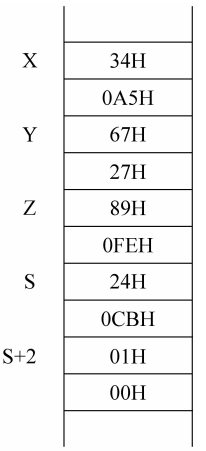

图 3-1 内存分配情况

解

```
DATA   SEGMENT
       X  DW  0A534H
       Y  DW  2767H
       Z  DW  0FE89H
       S  DW  2 DUP(?)     ;(或 S  DD  ?)
DATA   ENDS
CODE   SEGMENT
       ASSUME  DS:DATA, CS:CODE
START: MOV     AX, DATA
       MOV     DS, AX
       MOV     DX, 0        ;32 位累加器高 16 位清零
       MOV     AX, X
       ADD     AX, Y
       ADC     DX, 0        ;(DX)+0+(CY)→(DX), 进位加到 DX 中
       ADD     AX, Z
       ADC     DX, 0
       MOV     S+2, DX      ;运算结果高位送和的高字单元
       MOV     S, AX        ;运算结果低位送和的低字单元
       MOV     AX, 4C00H
       INT     21H
CODE   ENDS
       END     START
```

进行加法运算时，应该考虑到可能产生进位，和可能要占 32 位，所以在定义 S 时要定义 2 个字单元，或者可以定义成 1 个双字。如果 S 是双字，那么在传送运算结果时应该改成这样两句：

```
MOV  WORD PTR S+2, DX
MOV  WORD PTR S, AX
```

因为 S 的数据类型是双字，而 AX 是字，所以要利用类型 PTR 强行改变操作数的类型。

本题还可以这样解：

```
MOV  DX, 0
MOV  AX, 0
LEA  SI, X       ;X 的偏移地址→(SI),SI 就是指针
```

```
ADD  AX, [SI]   ;利用 SI 间接寻址找到 0A534H 与 AX 相加
ADC  DX, 0      ;(DX)+0+(CY)→(DX),进位加到 DX 中
INC  SI         ;修改指针,指向 Y
INC  SI
ADD  AX, [SI]   ;利用 SI 间接寻址找到 2767H 与 AX 相加
ADC  DX, 0      ;(DX)+0+(CY)→(DX),进位加到 DX 中
INC  SI         ;修改指针,指向 Z
INC  SI
ADD  AX, [SI]   ;利用 SI 间接寻址找到 0FE89H 与 AX 相加
ADC  DX,0       ;(DX)+0+(CY)→(DX),进位加到 DX 中
MOV  S+2, DX    ;运算结果高位送和的高字单元
MOV  S, AX      ;运算结果低位送和的低字单元
```

上面的例子使用 SI 寄存器存放“加数”的地址,通过“寄存器间接寻址”逐个找到“加数”。这个 SI 寄存器称为“指针”。现在还看不出这种编程方法的优点,等学习了简单的循环指令后,就能真正体会指针的“魅力”了。

2. 减法指令

有 4 条基本的减法指令(见表 3-2)。

表 3-2　减法运算指令

指令名称	操作码助记符	指令格式	指令功能	主要用途
减法指令	SUB(Subtract)	SUB dest,src	dest←dest－src	减法
带借位的减法指令	SBB(Subtract with Borrow)	SBB dest,src	dest←dest－src－cf	分段减法的高位相减
减量指令	DEC(Decrement)	DEC dest	dest←dest－1	修改计数器和指针
求补指令	NEG(Negate)	NEG dest	dest←0－dest	求相反数

对于 SUB 和 SBB 指令,

- 目的操作数:8/16/32 位的寄存器/存储器。
- 源操作数:与目的操作数同类型的寄存器/存储器/立即数。
- 两条指令执行后,CPU 的状态标志 CF,OF,ZF,SF, PF, AF 按照运算结果被刷新。

对于 DEC,NEG 指令,

- 目的操作数:8/16/32 位的寄存器/存储器。
- CPU 的状态标志 OF, ZF, SF, PF, AF 按照运算结果被刷新,但是 CF 标志不受影响。

例 3-2　X,Y,Z 在数据段中定义如下,求 Z＝X－Y。

```
X  DD  12345678H
Y  DD  08911273H
Z  DD  ?
```

解法 1　用 32 位寄存器进行运算：

```
MOV  EAX, X
SUB  EAX, Y
MOV  Z, EAX
```

解法 2　用 16 位寄存器进行运算：

```
MOV  AX, WORD PTR X
SUB  AX, WORD PTR Y     ;低 16 位相减
MOV  WORD PTR Z, AX     ;差的低 16 位送 Z 的低 16 位
MOV  AX, WORD PTR X+2
SBB  AX, WORD PTR Y+2   ;高 16 位相减,并减去低位可能产生的借位
MOV  WORD PTR Z+2, AX   ;差的高 16 位送 Z 的高 16 位
```

X,Y,Z 是双字属性,从中取 2 个字节时,必须用类型操作符 PTR 进行强制类型转换。

例 3-3　已知(AL)＝0CEH,分析下列指令的执行结果以及各标志位的意义。

(1) SUB　AL，40H

结果：(AL)＝8EH，OF＝0，CF＝0，ZF＝0，SF＝1

如果这是无符号数减法,由结果反推,(AL)＝0CEH＝206D,执行运算 206－64。CF＝0 表示没有借位,被减数＞减数;ZF＝0 表示结果为非 0,(AL)≠40H;OF 和 SF 无意义。

如果这是有符号数减法,0CEH 是(－50)的补码,执行运算(－50)－64。OF＝0 表示没有溢出,运算结果正确;ZF＝0 表示结果为非 0;SF＝1 表示相减的结果是负数,即被减数＜减数;CF 无意义。

(2) SUB　AL，64H

结果：(AL)＝6AH，OF＝1，CF＝0，ZF＝0，SF＝0

如果这是无符号数减法,由结果反推,(AL)＝0CEH＝206D,执行运算 206－100,CF＝0 表示没有借位,被减数＞减数;ZF＝0 表示结果为非 0;OF 和 SF 无意义。

如果这是有符号数减法,执行运算(－50)－100,OF＝1 表示有溢出,运算结果出错。此时,SF 的内容与正确结果相反,SF＝0 表示正确的符号位 SF＝1,被减数＜减数;ZF＝0 表示结果为非 0;CF 无意义。

3. 乘法指令和除法指令

乘法指令分为无符号数乘法和有符号数乘法指令(见表 3-3)。

表 3-3　乘法运算指令

指令名称	操作码助记符	指令格式	指令功能
无符号数乘法指令	MUL(Unsigned Multiplication)	MUL src	8 位源操作数：AX←(AL)×src 16 位源操作数：DX，AX←(AX)×src
有符号数乘法指令	IMUL(Signed Integer Multiplication)	IMUL src	

说明：

- 两个 N 位操作数相乘，得到 $2N$ 位的乘积。
- 源操作数不能为立即数。
- 对于无符号数乘法，如果乘积的高 N 位为 0，则 CF＝OF＝0，否则 CF＝OF＝1。其余标志位无意义。对于有符号数乘法，如果乘积高 N 位为低 N 位的符号扩展，则 CF＝OF＝0，否则 CF＝OF＝1，其余标志位无意义。

例 3-4 已知(AL)＝0FFH，(BL)＝02H，分别执行 MUL　BL 和 IMUL　BL 这两条指令，结果有什么不同？

解 执行 MUL　BL 后，(AX)＝01FEH，CF＝OF＝1

执行 IMUL　BL 后，(AX)＝0FFFEH，CF＝OF＝0

第一条指令是无符号数的乘法，即 0FFH×2H＝01FEH。CF＝OF＝1 表示乘积的高 8 位不等于 0。

第二条指令是有符号数的乘法，即 $-1\times2=-2$。0FFFEH 是 -2 的 16 位补码。CF＝OF＝0 表示乘积高 8 位为低 8 位的符号扩展。

286 以后的 CPU 新增了两条有符号乘法指令。

(1) IMUL　目的操作数，源操作数

说明：

- 本指令的操作数可以是 16/32 位，但是不能为 8 位。
- 两个 N 位操作数相乘，得到 N 位的乘积。
- 源操作数允许使用立即数。

(2) IMUL　目的操作数，源操作数 1，源操作数 2

说明：

- 目的操作数←源操作数 1×源操作数 2。
- 本指令的操作数可以是 16/32 位，但是不能为 8 位。
- 两个 N 位操作数相乘，得到 N 位的乘积。
- 源操作数 2 允许使用立即数。

以上两条指令，源操作数可以是立即数，使得编程更灵活，但是有一个限制，两个 N 位操作数相乘，乘积不能超过 N 位。

除法指令分为有符号数除法和无符号数除法指令(见表 3-4)。

表 3-4　除法运算指令

指令名称	操作码助记符	指令格式	指令功能
无符号数除法指令	DIV(Unsigned Division)	DIV src	8 位源操作数：(AX)÷src，AL←商，AH←余数 16 位源操作数：(DX，AX)÷src，AX←商，DX←余数
有符号数除法指令	IDIV(Signed Integer Division)	IDIV src	

说明：

- DIV 和 IDIV 指令分别用于无符号数和有符号数除法。

- 8 位源操作数时：(AX)÷源操作数，AL←商，AH←余数。
  16 位源操作数时：(DX，AX)÷源操作数，AX←商，DX←余数。
  32 位源操作数时：(EDX，EAX)÷源操作数，EAX←商，EDX←余数。
- 两个 N 位操作数相除，应首先把被除数扩展为 $2N$ 位。无符号数除法用零扩展，有符号数除法用符号扩展。
- 无符号数除法($2N$ 位)÷(N 位)的商大于 2^N-1，产生“除法溢出”错误。有符号数除法($2N$ 位)÷(N 位)的商大于 $2^{N-1}-1$ 或者小于 -2^{N-1}，产生“除法溢出”错误。
- 源操作数不能为立即数。

如果(AX)＝2000H，(BL)＝2H，执行 DIV BL 指令的结果怎样呢？首先，这是一条(16 位)÷(8 位)的无符号数除法指令。除法的正确结果应等于 1000H，超出了 8 位，所以会出现“除法溢出”错误。这条指令本身的“语法”并没有错，所以在汇编期间不会给出出错信息。这个错误在程序运行期间才会发生。那么，该怎样避免这种情况的发生呢？可以把被除数扩展成 32 位(MOV DX，0)，除数扩展成 16 位(MOV BH，0)，32 位(DX:AX)除以 16 位(BX)，商存放在 AX 中，是 16 位的，就不会溢出了。注意，无符号数的扩展是高位添 0。

例 3-5 编程计算下列算式：S＝10＊(X－Z)/(Y＋Z)＋(X＋Y＋Z)/2。

解

```
DATA  SEGMENT
      X  DW  200
      Y  DW  -50
      Z  DW  100
      S  DW  2 DUP (?)
DATA  ENDS
CODE  SEGMENT
    ASSUME  DS:DATA, CS:CODE
START: MOV  AX, DATA
       MOV  DS, AX
       MOV  AX, X
       SUB  AX, Z          ;(X-Z)→(AX)
       MOV  BX, 10
       IMUL BX             ;(X-Z)＊10→(DX:AX), 作为被除数待用
       MOV  CX, Y
       ADD  CX,Z           ;(Y+Z)→(CX)
       IDIV CX             ;10＊(X-Z)/(Y+Z)→(AX)
       MOV  S, AX          ;多项式的第一项值暂存到 S 单元
       MOV  AX, X
       ADD  AX,Y
       ADD  AX,Z           ;(X+Y+Z)→(AX)
       CWD                 ;(AX)扩展成(DX:AX)
       MOV  BX, 2
```

```
        IDIV BX          ;(X+Y+Z)/2→(AX)
        ADD  S, AX       ;多项式两项相加
        ADC  S+2, 0      ;如果产生进位,则存放在 S 的高位单元中
        MOV  AX, 4C00H
        INT  21H
CODE    ENDS
        END  START
```

3.1.2 循环

在实际应用中,常常会遇到某一段程序需要重复执行,解决这个问题的最好方法是使用循环结构。循环结构把一些冗长的程序减缩为非常短小的程序,且逻辑上也很清晰。

基本循环指令格式:

```
LOOP  标号
```

功能:

```
CX←(CX)-1        ;如果(CX)≠ 0,转向"标号"处执行,否则执行下一条指令
```

说明:

- LOOP 可以改变指令的执行次序,称为"控制指令"。
- LOOP 指令使一段程序重复地执行,称为"循环"。重复执行的次数由 CX 寄存器中的值决定。CX 寄存器因此也称为"计数器"。

例 3-1 中的另一种解法,现在可以改成以下形式:

```
        MOV  DX, 0
        MOV  AX, 0
        MOV  CX, 3
        LEA  SI, X       ;第一个数的偏移地址→(SI),SI 是指针
NEXT:   ADD  AX, [SI]    ;利用 SI 间接寻址找到加数与 AX 相加
        ADC  DX, 0       ;进位加到 DX 中,(DX)+0+(CY)→(DX)
        INC  SI
        INC  SI          ;修改指针,指向下一个数
        LOOP NEXT        ;(CX)-1→(CX),若(CX)≠0,转向 NEXT
                         ;重复执行,共 3 次
        MOV  S+2, DX     ;运算结果高位送和的高字单元
        MOV  S, AX       ;运算结果低位送和的低字单元
```

这里,利用寄存器间接寻址的方式的优越性得以体现了:如果有 100 个数相加,只要将 100 送给 CX 即可。如果不用指针方式,而用直接寻址,那就无法用循环指令了。使用循环可以重复利用一段代码,完成较为复杂的功能,充分发挥计算机高速、自动运行的特点。

使用循环指令要注意以下几点。

(1) 要设置初始条件,这些指令在循环开始前被执行。

(2) 在循环体内部,一定要有修改计数器(CX)的语句,最终使(CX)=0,这样才能避免程序进入死循环。

(3) 循环结束后,要处理循环的结果。

例 3-6 编程计算 1+2+3+…+100,结果送 SUM 单元。

解 数据段部分定义:

```
SUM    DW    ?
```

部分代码段如下:

```
      MOV    CX, 100        ;初始化部分,设置循环次数
      MOV    AX, 0          ;初始化部分,设置累加和初值
      MOV    BX, 1          ;初始化部分,设置参与累加的数的初值
NEXT: ADD    AX, BX         ;累加,循环部分开始
      INC    BX             ;修改参与累加的数
      LOOP   NEXT           ;循环 100 次,此句隐含对计数器的修改动作
      MOV    SUM, AX        ;循环结束后,存放累加和到 SUM 单元
```

利用本题数据的特殊性,程序也可以简化成这样:

```
      MOV    CX, 100
      MOV    AX, 0
NEXT: ADD    AX, CX
      LOOP   NEXT
```

这段程序中,CX 既充当了计数器又充当了参与累加的数,完成了 0+100+99+…+2+1。

3.1.3 十进制数运算

进行 BCD 数运算时,80x86 系列微处理器先用二进制运算指令进行计算,然后按照 BCD 数运算规则进行调整。BCD 数的除法运算是例外:先调整,后运算。

压缩 BCD 数运算的操作有以下两种。

(1) DAA(decimal adjust after addition)十进制加法调整

格式:

```
DAA
```

功能:对 AL 中的二进制加法结果进行 BCD 运算调整。

(2) DAS(decimal adjust after subtraction)十进制减法调整

格式:

```
DAS
```

功能:对 AL 中的二进制减法结果进行 BCD 运算调整。

例 3-7 用 BCD 数进行运算:9899+1111,分析 DAA 指令执行前后 AL,CF 的变化情况。

解

```
DATA  SEGMENT
      X      DW 9899H
      SUM    DD ?
DATA  ENDS
CODE  SEGMENT
      ASSUME  DS:DATA, CS:CODE
START: MOV  AX, DATA
       MOV  DS, AX
       MOV  AX, X
       ADD  AL, 11H              ;(AL)=0AAH, (CF)=0
       DAA                       ;(AL)=10H, (CF)=1
       MOV  BYTE PTR SUM, AL
       MOV  AL, AH
       ADC  AL, 11H              ;(AL)=0AAH, (CF)=0
       DAA                       ;(AL)=10H, (CF)=1
       MOV  BYTE PTR SUM+1, AL
       ADC  BYTE PTR SUM+2, 0    ;进位加入和的高位单元
       MOV  BYTE PTR SUM+3, 0    ;和的最高字节单元清零
       MOV  AX, 4C00H
       INT  21H
CODE   ENDS
       END  START
```

程序执行后,SUM 单元内容为 00011010H，就是 9899＋1111＝11010。

(3) AAA(ASCII Adjust after Addition)非压缩十进制加法调整

格式：

```
AAA
```

功能：对 AL 中的加法结果进行非压缩 BCD 数运算调整。

(4) AAS(ASCII Adjust after Subtraction)非压缩十进制减法调整

格式：

```
AAS
```

功能：对 AL 中的减法结果进行非压缩 BCD 数运算调整。

3.1.4 逻辑运算指令

与算术运算不同,逻辑运算指令将每一位二进制单独进行运算,各位之间没有相互进位/借位的关系。逻辑运算指令执行之后,CF,OF 标志位固定为 0。SF、PF、ZF 按照运算结果的特征设置。

逻辑运算指令有如下几种：

(1) AND　逻辑乘(逻辑与)指令。

(2) OR　逻辑加(逻辑或)指令。

(3) XOR　逻辑异或指令(半加)。

(4) NOT　逻辑非(取反)指令。

下面举例说明各条指令的用途。

例 3-8　在 BUF 开始的内存单元,存放了数字字符,把这串字符转换成它们各自对应的数值。

解

```
DATA   SEGMENT
       BUF  DB  '1234567890'
DATA   ENDS
CODE   SEGMENT
       ASSUME   DS:DATA, CS:CODE
START: MOV  AX, DATA
       MOV  DS, AX
       MOV  CX, 10
       MOV  BX, 0              ;BX 相当于 BUF 数组的下标
       AND  BUF[BX], 0FH       ;把 ASCII 码的高 4 位清 0,保留低 4 位
       INC  BX                 ;修改指针,准备指向下一个 ASCII 码
       LOOP NEXT
       MOV  AX, 4C00H
       INT  21H
CODE   ENDS
       END  START
```

总结:逻辑与操作可以清 0 某些位。某位与 0 相与得 0,与 1 相与保持不变。

例 3-9　在 BUF 开始的内存单元,存放了 10 个一位十进制数,把它们转换成各自对应的 ASCII 码。

解

```
DATA   SEGMENT
       BUF  DB  1, 2, 3, 4, 5, 6, 7, 8, 9, 0
DATA   ENDS
CODE   SEGMENT
       ASSUME   DS:DATA, CS:CODE
START: MOV  AX, DATA
       MOV  DS, AX
       MOV  CX, 10
       MOV  BX, 0              ;BX 相当于 BUF 数组的下标
NEXT:  OR   BUF[BX], 30H       ;把 ASCII 码的高 4 位置为 0011,保留低 4 位
       INC  BX                 ;修改指针,准备指向下一个数
       LOOP NEXT
       MOV  AX, 4C00H
       INT  21H
```