第3章

数据运算与输入输出

数值计算是计算机的一项基本功能。本章学习算术运算指令、表达式计算、基本循环结构程序、控制台输入输出。本章的学习重点是：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。

3.1 算术运算

计算机内的数值数据有定点数和浮点数两大类，本节介绍定点整数的算术运算。

3.1.1 加法指令

有三条基本的加法指令：

(1) ADD(Addition)：加法指令。

格式：

```
ADD  目的操作数,源操作数
```

功能：目的操作数←目的操作数＋源操作数。

目的操作数：8 位、16 位、32 位的寄存器或存储器。

源操作数：与目的操作数同类型的寄存器、存储器或立即数。

例：

```
ADD  AX,SI                    ;AX←(AX)+(SI),16 位运算
ADD  X,3                      ;X←(X)+3,运算位数由 X 的类型确定
ADD  [EBX],EDX                ;DS:[EBX] ←DS:[EBX]+EDX,32 位运算
```

下面的指令无法确定操作数的类型，汇编时将报告错误：

```
ADD  [SI],5                   ;两个操作数都没有明确类型
```

如果目的操作数是 DS:[SI]指向的字节存储单元，可以修改如下：

```
ADD  BYTE PTR [SI],5
```

说明：

加法指令执行后，CPU 的状态标志 CF，OF，ZF，SF，PF，AF 按照运算结果被刷新；

操作数可以是 8 位、16 位或 32 位，源操作数与目的操作数应该有相同的类型，不能同时为内存操作数。

(2) ADC(Addition with Carry)：带进位的加法指令。

格式：

```
ADC  目的操作数,源操作数
```

功能：目的操作数←目的操作数＋源操作数＋CF。

目的操作数：8 位、16 位、32 位的寄存器或存储器。

源操作数：与目的操作数同类型的寄存器、存储器或立即数。

说明：

该指令对标志位的影响、对操作数的要求与 ADD 指令相同；

主要用于两个数据分段相加时高位的加法运算。

【例 3-1】 X＝33445566778899AAH，Y＝123456789ABCDEF0H，计算 $Z=X+Y$。

对上面的两个数据进行分段相加。假设对应的数据段已经定义，变量 X，Y，Z 用 DQ 定义，可以编写程序如下：

```
MOV  EAX,DWORD PTR X          ;取 X 的低 32 位,送入 EAX
ADD  EAX,DWORD PTR Y          ;X,Y 的低 32 位相加,结果在 EAX
                              ;(EAX)=778899AAH+9ABCDEF0H=1245789AH,CF=1
MOV  DWORD PTR Z,EAX          ;低 32 位的和送 Z 的低 32 位
MOV  EAX,DWORD PTR X+4        ;取 X 的高 32 位,送入 EAX
ADC  EAX,DWORD PTR Y+4        ;X、Y 的高 32 位及低 32 位的进位相加
                              ;(EAX)=33445566H+12345678H+CF=4578ABDFH,CF=0
MOV  DWORD PTR Z+4,EAX        ;高 32 位的和送 Z 的高 32 位
```

程序运行完毕：Z＝4578ABDF1245789AH，CF＝0。

【例 3-2】 用 A，B，C 表示 8 位无符号数，求它们的和，送入 SUM。

3 个 8 位无符号数的和可能超过 255，它们的和应保留为 16 位。

```
MOV  AL,A                     ;取第一个数
MOV  AH,0                     ;高 8 位清零,第一个数"零扩展"为 16 位
MOV  DL,B                     ;取第二个数
MOV  DH,0                     ;把第二个数"零扩展"为 16 位
ADD  AX,DX                    ;加第二个数
MOV  DL,C                     ;取第三个数,高 8 位已经为 0
ADD  AX,DX                    ;加第三个数
MOV  SUM,AX                   ;保存三个数的和
```

这个问题的解决还有另一种选择：

```
MOV  AL,A                     ;取第一个数
MOV  AH,0                     ;高 8 位清零,准备存放和的高 8 位
```

```
ADD  AL,B                ;加第二个数
ADC  AH,0                ;如果有进位,存入 AH
ADD  AL,C                ;加第三个数
ADC  AH,0                ;如果有进位,加入 AH
MOV  SUM,AX              ;保存三个数的和
```

注意：上面的方法仅限于无符号数，有符号数据的处理方法请看下例。

【例 3-3】 用 P,Q,R 表示 8 位有符号数，求它们的和，送入 TOTAL。

3 个 8 位有符号数的和可能大于 127，或者小于－128，它们的和应保留为 16 位。

```
MOV  AL,P                ;取第一个数
CBW                      ;扩展为 16 位
MOV  DX,AX               ;第一个数转存入 DX
MOV  AL,Q                ;取第二个数
CBW                      ;扩展为 16 位
ADD  DX,AX               ;加第二个数
MOV  AL,R                ;取第三个数
CBW                      ;扩展为 16 位
ADD  DX,AX               ;加第三个数
MOV  TOTAL,DX            ;保存三个数的和
```

由于 CBW 指令进行 8 位有符号数向 16 位的扩展限定使用 AL 和 AX 寄存器，上面程序用 DX 寄存器存放 3 个数的和。

使用 32 位 80x86 CPU 的 MOVZX，MOVSX 可以简化上面的程序，读者不妨一试。

(3) INC(Increment)：增量指令。

格式：

```
INC  目的操作数
```

功能：目的操作数←目的操作数＋1。

目的操作数：8 位、16 位、32 位的寄存器或存储器。

例：

```
INC  EBX                 ;EBX←(EBX)+1,32 位运算
INC  X                   ;X←(X)+1,运算位数由 X 的类型确定
INC  WORD PTR [BX]       ;DS:[BX]←DS:[BX]+1,16 位运算
```

说明：

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；

增量指令常常用来修改计数器和存储器指针的值。

3.1.2 减法指令

这里先介绍 4 条基本的减法指令，CMP(比较)指令留在以后介绍。

(1) SUB(Subtract)：减法指令。

格式：

```
SUB  目的操作数,源操作数
```

功能：目的操作数←目的操作数－源操作数。

目的操作数：8 位、16 位、32 位的寄存器或存储器。

源操作数：与目的操作数同类型的寄存器、存储器或立即数。

例：

```
SUB  EAX,ESI                ;EAX←(EAX)-(ESI),32 位运算
SUB  Y,20H                  ;Y←(Y)-20H,运算位数由 Y 的类型确定
SUB  WORD PTR [BP],5        ;SS:[BP]←SS:[BP]-5,16 位运算
```

说明：该指令对标志位的影响、对操作数的要求与 ADD 指令相同。

(2) SBB(Subtract with Borrow)：带借位的减法指令。

格式：

```
SBB  目的操作数,源操作数
```

功能：目的操作数←目的操作数－源操作数－CF。

目的操作数：8 位、16 位、32 位的寄存器或存储器

源操作数：与目的操作数同类型的寄存器、存储器或立即数。

说明：

该指令对标志位的影响、对操作数的要求与 ADD 指令相同；

主要用于两个数据分段相减时高位数据的减法运算。

(3) DEC(Decrement)：减量指令。

格式：

```
DEC  目的操作数
```

功能：目的操作数←目的操作数－1。

目的操作数：8 位、16 位、32 位的寄存器或存储器。

例：

```
DEC  CX                     ;CX←(CX) -1,16 位运算
DEC  X                      ;X←(X)-1,运算位数由 X 的类型确定
DEC  DWORD PTR [DI]         ;DS:[DI]←DS:[DI]-1,32 位运算
```

说明：

减量指令执行后，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；

减量指令常常用来修改计数器和存储器指针的值。

(4) NEG(Negate)：求补指令。

格式：

```
NEG  目的操作数
```

功能：目的操作数←0－目的操作数。

目的操作数：8 位、16 位、32 位的寄存器或存储器。

例：

```
NEG  Z                    ;Z←-Z,运算位数由 Z 的类型确定
```

由于有符号数均使用补码表示，所以该指令的操作等效于：

目的操作数←[目的操作数]$_{求补}$

3.1.3 乘法和除法指令

(1) MUL(Unsigned Multiplication)：无符号数乘法

格式：

```
MUL  源操作数
```

源操作数：8 位、16 位、32 位的寄存器或存储器。

功能：

8 位源操作数时：AX←(AL)×源操作数。

16 位源操作数时：DX,AX←(AX)×源操作数；

32 位源操作数时：EDX,EAX←(EAX)×源操作数。

说明：

两个 N 位操作数相乘，得到 $2N$ 位的乘积；

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。

(2) IMUL(signed integer multiplication)：有符号数乘法。

这条指令有三种格式。

① 格式 1：

```
IMUL  源操作数
```

源操作数：8 位、16 位、32 位的寄存器或存储器。

功能：

8 位源操作数时：AX←(AL)×源操作数；

16 位源操作数时：DX,AX←(AX)×源操作数；

32 位源操作数时：EDX,EAX←(EAX)×源操作数。

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；

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。

相同的两组二进制代码分别用 MUL 和 IMUL 运算，可能得到不同的结果：

例如：

```
(AL)=0FFH,(X)=2
```

```
MUL  X                    ;(AX)=01FEH,(255×2=510)
IMUL  X                   ;(AX)=0FFFEH,(-1×2=-2)
```

② 格式 2：

```
IMUL  目的操作数,源操作数
```

目的操作数：16 位或 32 位的寄存器。

源操作数：与目的操作数相同类型的寄存器、存储器或立即数。

功能：目的操作数←目的操作数×源操作数。

说明：

两个 N 位操作数相乘，得到 N 位（注意，不是 $2N$ 位）的乘积，

本指令的操作数不能为 8 位。

例：

```
IMUL  AX,BX               ;AX←(AX)×(BX),两个 16 位数相乘,得到 16 位积
IMUL  EDX,VERB            ;EDX←(EDX)×(VERB),变量 VERB 用 DD 定义
IMUL  EDX,3               ;EDX←(EDX)×0000 0003H
```

③ 格式 3：

```
IMUL  目的操作数,源操作数 1,源操作数 2
```

目的操作数：16 位或 32 位的寄存器。

源操作数 1：与目的操作数相同类型的寄存器或存储器。

源操作数 2：与目的操作数相同类型的立即数。

功能：目的操作数←源操作数 1×源操作数 2

说明：

两个 N 位操作数相乘，得到 N 位的乘积；

本指令的操作数不能为 8 位。

例：

```
IMUL  AX,BX,-5            ;AX←(BX)×(-5),16 位数相乘,得到 16 位积
IMUL  EDX,X,300           ;EDX←(X)×300,X 必须为双字变量,32 位数相乘
```

对于 IMUL 指令，除 8 位、16 位的单操作数指令外，其余均为 286 或 386 新增的，需要在程序的首部用.386 加以声明。

(3) DIV(unsigned division)：无符号除法。

格式：

```
DIV   源操作数
```

源操作数：8 位、16 位、32 位的寄存器或存储器。

功能：

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。

说明：

两个 N 位操作数相除,应首先把被除数零扩展为 $2N$ 位；

例如,要进行除法(AX)÷(BX),假设 AX、BX 内均为无符号数：

```
MOV  DX,0              ;32 位被除数高 16 位清零
DIV  BX                ;(DX,AX)÷BX,AX←商,DX←余数
```

如果($2N$ 位)÷(N 位)的商大于 2^N-1,会产生除法溢出错误。

源操作数不能为立即数。

例如,要进行除法(AX)÷5,首先应确定是 16 位÷8 位还是 32 位÷16 位；

如果能确定(AX)÷5 的商小于 255,可以执行 16 位÷8 位除法；

```
MOV  BL,5          ;除数存入 BL 寄存器
DIV  BL            ;16 位÷8 位,AL←商,AH←余数
```

如果不能确定(AX)÷5 的商小于 255,可以执行 32 位÷16 位除法；

```
MOV  BX,5              ;除数存入 BX 寄存器
MOV  DX,0              ;32 位被除数高 16 位清零
DIV  BX                ;(DX,AX)÷BX,AX←商,DX←余数
```

(4) IDIV(signed integer division)：有符号数除法。

格式：

```
IDIV  源操作数
```

源操作数：8 位、16 位、32 位的寄存器或存储器。

功能：

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。

说明：

两个 N 位操作数相除,应首先把被除数符号扩展为 $2N$ 位；

例如,要进行除法(AX)÷(BX)：

```
CWD                ;被除数 AX 符号扩展到 DX,AX
IDIV    BX         ;(DX,AX) ÷(BX),AX←商,DX←余数
```

两个有符号数相除,余数与被除数同号：

```
-10  IDIV  -3 : 商=3,余数=-1
-10  IDIV  3  : 商=-3,余数=-1
```

如果($2N$ 位)÷(N 位)的商大于 $2^{N-1}-1$ 或者小于 -2^{N-1},会产生除法溢出错误。

源操作数不能为立即数；

相同的两组二进制代码分别用 DIV 和 IDIV 运算,可能得到不同的结果。

例如：(AX)＝0FFFFH，(CL)＝1，进行 16 位÷8 位运算：

```
DIV  CL          ;0FFFFH÷1= 0FFFFH,产生除法溢出
IDIV  CL         ;(AL)=0FFH,(AH)= 0(－1÷1= －1……0)
```

3.1.4 表达式计算

使用上述指令可以进行整数表达式的计算。与高级语言程序不同的是，必须由程序员按照各级运算符的优先级，合理地安排计算次序和数据类型。

【例 3-4】 用 A、B、C、D 表示有符号字变量，计算：$Z=\dfrac{A+B}{2}+\dfrac{3(B+C)}{A-C}$。

首先确定计算顺序如下：

$A+B$→$(A+B)/2$→暂存中间结果；

$A-C$→暂存中间结果；

$B+C$→$(B+C)*3$→$(B+C)*3/(A-C)$→$(B+C)*3/(A-C)+(A+B)/2$→保存最终结果。

可以看出，上面的运算顺序与表达式的书写顺序有所不同。

其次，确定各次运算的数据类型：

$A+B$ 扩展为 32 位，$(A+B)/2$ 结果为 16 位；

$(B+C)*3$ 结果为 32 位，$A-C$ 结果为 16 位，$(B+C)*3/(A-C)$结果为 16 位，最终结果为 16 位。

用汇编语言编写程序如下：

```
MOV    AX,A              ;取操作数 A
ADD    AX,B              ;进行运算"A+B"
CWD                      ;把被除数扩展为 32 位,有符号数用符号扩展
MOV    BX,2              ;除数转入寄存器
IDIV   BX                ;进行运算(A+B)/2,有符号数除法
MOV    BX,AX             ;把商转存到 BX,AX 留作下次乘、除法使用
MOV    CX,A              ;取分母第一个操作数
SUB    CX,C              ;进行运算"A-C",保存在 CX 内
MOV    AX,B              ;取分子第一个操作数
ADD    AX,C              ;进行运算"B+C"
MOV    DX,3              ;乘数转入寄存器
IMUL   DX                ;进行运算(B+C)*3,有符号数乘法
IDIV   CX                ;进行运算(B+C)*3/(A-C),有符号数除法
ADD    AX,BX             ;进行运算(B+C)*3/(A-C)+(A+B)/2
MOV    Z,AX              ;保存最终结果
```

上面的程序不包含输出部分，因此直接运行无法观察到运行结果，需要通过 TD 来运行该程序：

用 TD 调出可执行程序；

对照源程序，找出变量 Z 的存储地址；

单步或连续运行程序，直到指令 MOV　Z,AX 被执行；

在数据窗口中观察运行结果。

3.2 循环

程序的重复执行称为**循环**(loop)。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。

3.2.1 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：

```
    MOV  CX,10
L1: ...              ;需要重复执行的若干条指令
    ⋮
    LOOP  L1
```

上面的程序将 L1 到 LOOP 指令之间的一段程序重复执行 10 次。指令 MOV　CX,10 称为装载循环计数器，应在循环之前完成。

如果将上面的程序写成如下的情形：

```
L1:  MOV  CX,10
    ...              ;重复执行的若干条指令
    ⋮
    LOOP  L1
```

这个程序将无限制地运行下去，称为死循环，显然这不是希望见到的。

3.2.2 程序的循环

利用 LOOP 指令，可以使一段程序反复执行。这样，程序和指令的利用率得到提高。但是，完成这些功能花费的时间并没有因此缩短。

【例 3-5】 用循环的方法，将字节数组 ARRAY 的 20 个元素清零。

```
DATA   SEGMENT
```

```
  ARRAY  DB  20 DUP(?)                ;定义数组 ARRAY
DATA   ENDS
CODE   SEGMENT
       ASSUME   DS:DATA,CS:CODE
START: MOV      AX,DATA
       MOV      DS,AX
       LEA      BX,ARRAY             ;把数组 ARRAY 首地址装入 BX
       MOV      CX,20                ;装载循环计数器的初始值
ONE:   MOV      BYTE PTR[BX],0       ;把数组 ARRAY 的一个元素清零
       INC      BX                   ;修改 BX 的值,为下一次操作做准备
       LOOP     ONE                  ;计数循环
       MOV      AX,4C00H
       INT      21H
CODE   ENDS
       END      START
```

上面程序里,BX 存放数组元素的地址,称为**指针**,每次使用后,要及时修改它的值(地址),以便下一次使用。同样原因,装载 BX 初值(ARRAY 数组首地址)的指令也要放在循环开始之前。

把上面程序部分内容修改如下:

```
      MOV  CX,10                    ;装载循环计数器的初始值
ONE:  MOV  WORD PTR[BX],0           ;把数组 ARRAY 的两个元素清零
      INC  BX                       ;修改 BX 的值,为下一次操作做准备
      INC  BX
      LOOP ONE                      ;计数循环
```

经过这样修改,程序变长了,但是重复执行的次数减少了,总的执行时间缩短了。

还可以进一步作如下修改:

```
      MOV  CX,10                ;装载循环计数器的初始值
      MOV  AX,0                 ;
ONE:  MOV  [BX],AX              ;把数组 ARRAY 的两个元素清零
      INC  BX                   ;修改 BX 的值,为下一次操作做准备
      INC  BX
      LOOP ONE                  ;计数循环
```

程序又一次变长了,但使用 8086CPU 时执行时间会进一步缩短,这是因为指令 MOV [BX],AX 长度为 2B,指令 MOV WORD PTR[BX],0 长度为 4B,前者执行时所耗费的取指令时间和执行时间都短于后者。在编制对时间、空间特别敏感的程序时,需要适当考虑类似的细节。不同指令的代码长度和执行时间,可以参考相关资料。

3.2.3 数据的累加

编制程序时,经常会遇到求若干个数据的和的问题。它的基本方法是,安排一个容器(例如:寄存器),将它清零;把需要求和的数据逐个加入这个容器;加法结束时,容器中的