Intel 8080

汇编语言及其程序设计

（下）

〔例7：二进制多位精度运行

在实际应用中，数的位数一定不大于8位二进制的情况并不如此。比如在精密模／数转换器中，在三角函数或指数函数中，开方根值，压强值以及其他数据处理场合，都要求计算大于8位二进制的数。

现在我们假定有两个分别为24位二进制的数进行相加。第一个加数按颠倒的字节次序依次存放于存贮单元41H，第二个加数也是按颠倒的字节次序依次存放于存贮单元51H。运算结果存放在第一个加数的位置。加数的字节数存放在40H地址。

我们假设的一段情况是：

(40H)=03
(41H)=29
(42H)=A4
(43H)=50
⋮
(51H)=FB
(52H)=37
(53H)=28

这是两个24位的二进制的数。即50A429H和2827FBH进行相加，结果是：

(41H)=24H=29H+FBH
(42H)=DCH=A4H+37H+CARRY0
(43H)=78H=50H+28H+CARRY1

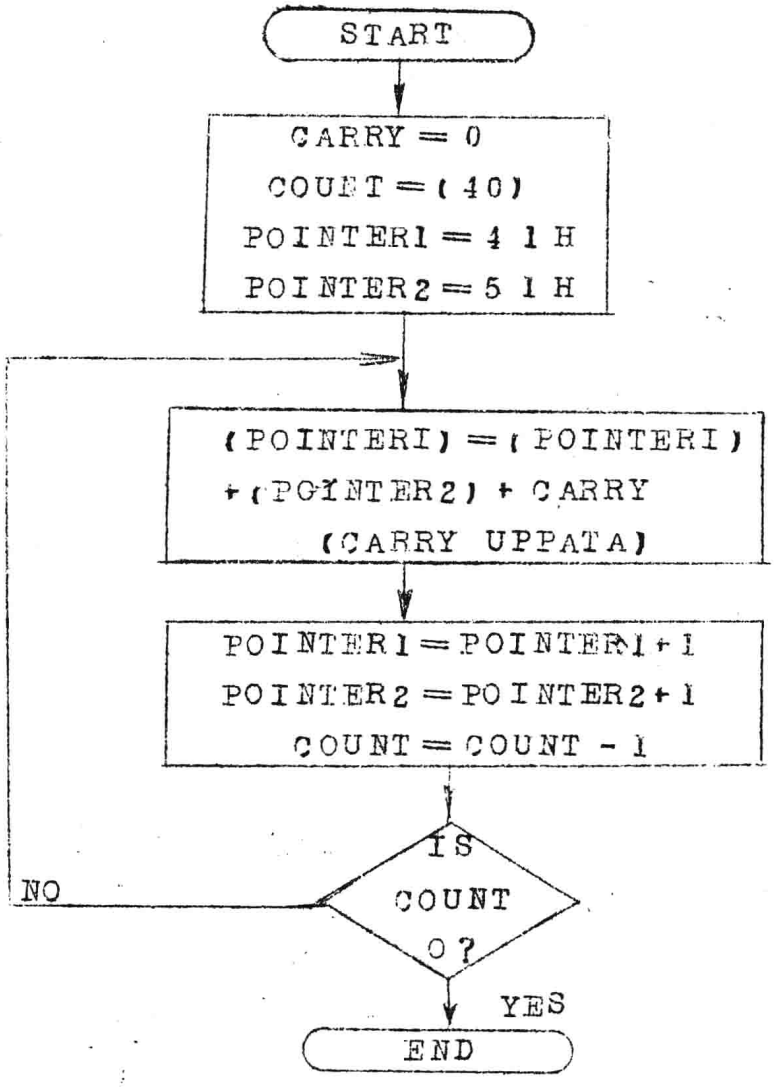
此处CARRY0 将最低字节相加向第二字节产生的进位；
CARRY1 将第二字节相加向第三字节产生的进位。

下面我们引进了程序流程图。

在该流程图中可以看出，它亦有一个初始化部份；有一个处理部分和一个循环控制部份。

然而，流程图中也有一个新特点：就是使用了CARRY，它是从

低向高传运。



现在我们列出源程序编码：

```
Infel 8080 Exdmple.7  MULTIPLE-PRECISION
                      ADDITION
        SUB A       ;START CARRY AT ZERO
        LXI H, 40H;COUNT=LENGTH OF NUMBERS
        MOV B, M
```

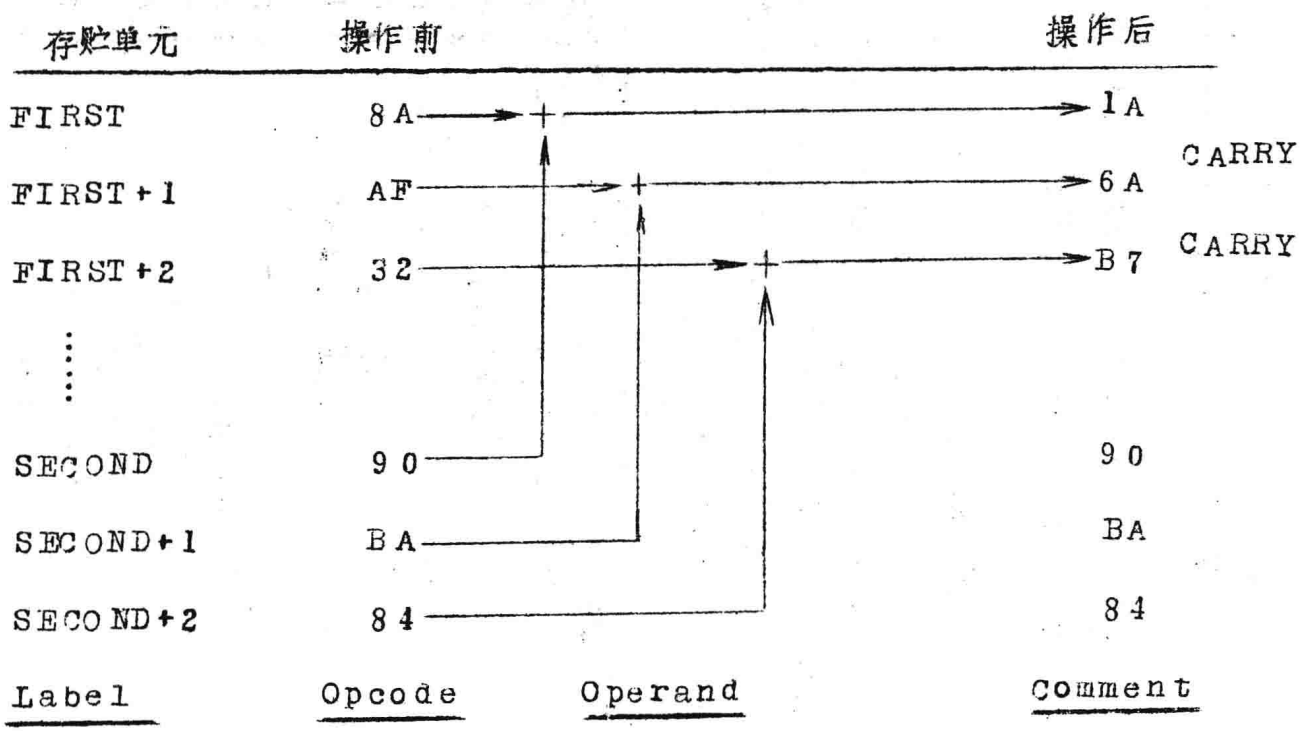
```
        LXI  D, 50H;POINT TO SECOND NUMBER
MPADD : INX  D
        INX  H
        LDAX D     ;GET 8 BITS OF SECOND
                    NUMBER
        ADC  M     ;ADD 8 BITS OF FIRST NUMBER
        MOV  M, A  ;STORE SUM IN FIRST NUMBER
        DCR  B
        JNZ  MPADD
        HLT
```

在Intel 8080 中，加法次序是颠倒的，另外，INXD，INX H，DCR B等指令均不影响CARRY，因此可以从一个迭代向下一个迭代传送信息。指令ADC M是将用H & L寻址的存贮单元内容与累加器的内容连同CARRY 进行相加。

下面给出该程序的汇编结果：

单元地址（16进制）	源程序	单元内容（16进制）
0000	SUB A	97
1	LXI H, 40H	21
2		40
3		00
4	MOV B, M	46
5	LXI D, 50H	11
6		50
7		00
8	MPADD: INX D	13
9	INX H	23
A	LDAX D	1A
B	ADC M	8E
C	MOV M, A	77
D	DCR B	05
E	JNZ MPADD	C2
0F		08
10		00
11	HLT	76

对于多位字长二进制运算，我们可考虑下面更为直观的程序。该程序是假定：E寄存器中存放字节长度数；相加的两个数均需以颠倒的次序存放，并且它们存放的前地址单元分别为 FIRST 和 SECOND，结果以旧存放在第一加数原来的位置。



```
LABEL    OPCODE  OPERAND     COMMENT
 MADD:    LXI     B, FIRST   ;SETUP START ADDRESS
                             ;OF FIRST OPERAND
          LXI     H, SECOND  ;SET UP SFART ADDRESS
                             ;OF SECOND OPERAND
          MVI     E, 3       ;COUNT = 3
          XRA     A          ;CLEAR CARRY
 LOOP:    LDAX    B          ;LOAD DATA TO ACC
          ADC     M          ;ADD WITH CARRY
          STAX    B          ;RESULT SAVE TO FIRST
          DCR     E
          JZ      DONE
          INX     BH
          INX     H
          JMP     LOOP
 DONE:     —
           —
 FIRST:   DB      90H, OBAH, 84H
 SECOND;  DB      8AH, OAFH, 32H
```

仿照上面的例子，我们可直接引出多字节无符号数减法的程序：

```
Label     Opcode    Operand
MSUB:     LXI       B, FIRST    ;被减数低字节地址打入B & C
          LXI       H, SECOND   ;减数低字节地址打入H & L
          MVI       E, 2        ;字节数送计数器
          XRA       A           ;清进位标志
LOOP:     LDAX      B           ;将被减数低字节打入ACC
          SBB       M           ;从A减去减数低字节
          STAX      B           ;结果存入FIRST地址
          DCR       E
          JZ        DONE
          INX       B
          INX       H
          JMP       LOOP
DONE:     —
          —
FIRST:    DB        01H, 13H
SECOND:   DB        03H, 05H
```

〔例8：十进制运算程序〕

事实上，不仅Intel 8080 而且大多数8位微处理机都具有一种用于加十进制数的特殊指令。这种指令通常称做十进制调整指令。该指令使用CARRY和HALF-CARRY（或称AUXILIRY CARRY）两标志。在相当广泛的应用场合，十进制运算是十分必要的功能。这些应用场合有销售点终端，电话终端，台式计算器，游戏及导航系统等等。

我们知道，4位二进制数可以表示0～9的十进制数之一，但亦能表示10～15的十六进制数（A～F）。为了在加法操作时确保十进制运算结果，每当产生A至F之间的结果时，必须加数值6到4位二进制数值上面。

为了执行十进制加法：

$$\begin{array}{r} 2985 \\ +\ 4936 \\ \hline 7921 \end{array}$$

过程如下：

1. 请进位和加每个数的二个低位数（用一个字节2表示两个十进制数）

$$\begin{array}{rl} 85 & =1000\ 0101 \\ 36 & =0011\ 0110 \\ +\ CARRY & =\qquad\quad\ 0 \\ \hline & \begin{array}{l}1011\\ C=0\end{array}\quad\begin{array}{l}1011\\ AC=0\end{array} \end{array}$$

现在累加器中的内容为BB（H）

2. 执行一次DAA操作，由于低4值大于9，故应对其加6：

$$\begin{array}{rl} (A) & =1011\ 1011 \\ 6 & =\qquad\ 0110 \\ \hline & 1100\ 0001 \end{array}$$

由于高4位也大于9，故亦应加60：

```
(A)=1100 0001
+ 6 =0110
-----------
     0010 0001
     C=1
```

现在累加器内容是21H，进位为"1"，存这两个数。

3 加下一组两个数：

```
29= 0010 1001
49= 0100 1001
+CY=          1
-----------
     0111 0011
          └→AC=1
```

4 执行DAA操作，由于辅助进位特征位被置位，故累加器须加上06：

```
(A)=0111 0011
+ 6 =    0110
-----------
     ┌0111 1001
     └→CY=0
```

现在累加器中的内容为79，这样，运算结果是：

2985+4936=7921

我们可以列出实现该十进制加法的程序如下：

```
DADD : LXI  D, ALPHA   ;设定被加数的地址
       LXI  H, BETA    ;设定加数的地址
       MVI  C, 2       ;设定循环次数
       XRA  A          ;清累和器
```

```
LOOP : LDAX  D          ;〔(D)(E)〕→A
       ADC   M          ;〔(H)(L)〕+(A)→A
       DAA              ;累加器十进制调整
       STAX  D          ;(A)→〔(D)(E)〕
       INX   D
       INX   H
       DCR   C
       JNZ   LOOP
       HLT
ALPHA: DB  85H,29H
BETA : DB  36H,49H
```

下面我们来说明一下十进制减法程序。

十进制减法比十进制加法要复杂得多。通常，过程包括产生减数的以十为模的补码，然后加到被减数上面去。例如，为了从56中减去34，就需要形成34的以10为模的补码（99－34＝65，65＋1＝66），然后执行56＋66＝122，再通过削去高位的进位，我们最后得到22，结果是正确的。

由于进行多字节减法运算，就发生了处理借位的问题。当在减法运算中，如未发生借位，那么在下一个操作使用的减数应为以10为模的补码；如果发生借位，则应用以10为模的反码。

注意：在减法中，进位标志的意思被转化了，这是因为处理变了补的数据之 。这里，进位标志为"1"，表示无借位；进位标志为"0"，恰表示借位。这个被转化了的进位标志的设置，使在减去操作中形成减数的或者以10为模的反码，或者以10为模的补码。

两个多位十进制数的减去操作详细过程叙述如下：

1. 设置进位特征位＝1，以表示没有借位；
2. 以99H装入累加器，代表10进制数99；
3. 加0和进位标志到累加器，得到99H或9AH；
4. 从累加器中减去减数，产生不是以10为模的反码，就是以10为模的补码。

5. 加被减数到累加器；

6. 利用DAA指令，以便得到累加器中的结果是十进制形式，并且若进位标志＝0，表示借位；

7. 如果有更多的数要减，进行第二循环，否则停止。

为了执行十进制减法：

$$\begin{array}{r} 4358 \\ -\ 1362 \\ \hline 2996 \end{array}$$

过程如下：

1. 置进位标志＝1，表示无借位；

2. 以99H装入累加器；

3. 加0和进位到累加器，产生9AH

$$\begin{array}{rl} (A) = & 1001\ 1001 \\ 0 = & 0 \\ +\ CY = & 1 \\ \hline & 1001\ 1010 = 9A \end{array}$$

4. 从累加器减去减数62

$$\begin{array}{rl} (A) = & 1001\ 1010 \\ -\ 62 = & 0110\ 0010 \\ \hline & \left[\begin{array}{l} 0011\ 1000 \\ \hookrightarrow CY=1\ (\text{表示无借位}) \end{array}\right. \end{array}$$

5. 加被减数58到累加器

$$\begin{array}{rl} (A) = & 0011\ 1000 \\ 58 = & 0101\ 1000 \\ \hline & \left[\begin{array}{ll} 1001 & \left[\begin{array}{l} 0000 \\ \hookrightarrow AC=1 \end{array}\right. \\ \hookrightarrow CY=0 & \end{array}\right. \end{array}$$

6. 进行 D A A

结果得 （A）＝9 6 ，C Y ＝ 0（表示有借位）

7. 以 9 9 H 装入累加器

8. 加 0 和进位到累加器，保持（A）＝9 9 H 。

9. 从累加器减去减数 1 3

```
（A）＝1 0 0 1  1 0 0 1
-  1 3 ＝0 0 0 1  0 0 1 1
-------------------------
        1 0 0 0  0 1 1 0
```

10. 加被减数 4 3 H 到累加器

```
（A）＝1 0 0 0  0 1 1 0
+  4 3 ＝0 1 0 0  0 0 1 1
-------------------------
        1 1 0 0  1 0 0 1
```

11. 执行 D A A

最后累加器为 2 9 ，C Y ＝ 1（表示无借位）

因此，全部计算结果为：

4 3 5 8 － 1 3 6 2 ＝ 2 9 9 6

我们可以列出实现上述十进制减法的程序如下：

```
DSUB : LXI  D, MINU ;设定被减数地址
       LXI  H, SBTAA;设定减数地址
       MVI  C, 2    ;循环次数为 2，每次循环减
                    ;两个数（一个字节），因而程序将减
                    ;4 个数
       STC          ;置进位标志＝1，表示无借位
LOOP : MVI  A, 99H  ;以 9 9 H 装入累加器
       ACI  0       ;加 0 和进位
       SUB  M       ;产生减数的补码
       XCHG         ;H & L 和 D & E 互换
```

```
        ADD  M          ;加被减数
        DAA             ;累加器十进制调整
        MOV  M, A       ;存结果
        XCHG            ;重新互换D & E和H & L
        DCR  C
        JZ   DONE
        INX  D          ;被减数下一个地址
        INX  H          ;减数的下一个地址
        JMP  LOOP       ;得到下两个十进制数
DONE :  NOP
MINU :  DB   58H
        DB   43H
SBTRA:  DB   62H
        DB   13H
```

〔例9：二进制数乘除法程序〕

两个无符号的8位数据字节的乘法运算可采用下列两段办法来进行：重复的加法，或利用一个寄存器的移位操作。我们知道：连续的加法形成乘积最简单，但连 最慢。例如2AH＊74H是可以将74H加到累加器（首先累加器需清“0”）2AH次两得到乘积。移位操作比上述为快地得到乘积。左移1位，相当于乘以2；而右移1位，相当于除以2。

下述的过程将产生一字节的乘数，乘以一字节的被乘数的正确的两字节的乘积：

A．检查乘数的低位，如果为“0”，故步骤B；如果为“1”，加被乘数到结果的高位字节；

B．全部二字节结果向右移一位；

C．重复步骤A和B，直到乘数的8位都被检查过为止。

假如，试计算乘法：2AH＊3CH＝09D8H。

1. 检查乘数第0位，因等于“0”，故将16位结果右移一位；

2. 检查乘数第1位，仍等于“0”，故将16位结果右移一位；

3. 检查乘数第2位，其值为“1”，加2AH到结果的高位字节，并将16位结果右移一位；

4. 检查乘数第3位，其值为“1”，加2AH到结果的高位字节，并将16位结果右移一位；

5. 检查乘数第4位，其值为“1”，加2AH到结果的高位字节，并将16位结果右移一位；

6. 检查乘数第5位，其值为“1”，加2AH到结果的高位字节，并将16位结果右移一位；

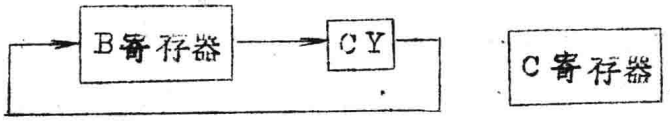
7. 检查乘数第6位，其值为“0”，16位结果右移一位；

8. 检查乘数第7位，其值为“0”，16位结果右移一位。

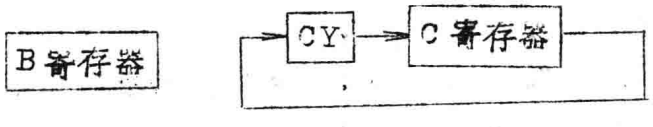
乘法运算过程就此完成。

由于上面叙的乘法程序利用了许多重要的编程技巧。程序使用B寄存器保持结果的高位字节，利用C寄存器保持结果的低位字节。16位结果在累加器中的右移是通过进位的循环右移指令来实现的。

首先进位标志置“0”然后B寄存器循环右移：



然后，C寄存器通过进位循环右移，以完成16位结果右移一位的使命：



现在我们列出实现两数相乘的程序：

```
MULT : MVI  D, MCAND  ; D=被乘数
       MVI  C, MCATOR; C=乘数
```

```
        MVI  B, OH      ;乘积的高位字节赋以初值
        MVI  E, 9       ;计数器赋以初值
MULTIO: MOV  A, C       ;乘数循环右移，最低位移出CY，
                        ;乘积的低位字节也相应移位
        RAR
        MOV  C, A       ;C循环右移后送回C
        DCR  E
        JZ DONE         ;YES，退出执行下面的程序
        MOV  A, B       ;NO，乘积高位字节送A
        JNC  MULTII     ;CARRY=0，跳转
        ADD  D          ;CARRY=1,加被加数到结果的高位字节
MULTII: RAR             ;结果的高位字节通过进位位循环右移
        MOV  B, A       ;结果高位字节通过进位循环右移后
                        回送B
        JMP MULTIO
DONE:   —
        —
```

下面再来简明地介绍一下二进制除法程序。

与乘法的例子相类似，一个无符号的16位数被一个无符号的8位数来除。这里，过程是使用减法，而不是加法；用循环左移指令代替上述程序的循环右移指令。程序是用B和C寄存器分别存放被除数的高位字节和低位字节，D寄存器存放除数，8位商存放在C寄存器中，余数在B寄存器。循环计数器仍使用E寄存器。

现在我们列出实现两数相除的程序：

```
DIV:  MVI   B, DIVNDH ; B=被除数高位字节
      MVI   C, DIVNDL ; C=被除数低位字节
      MVI   D, DIVDER ; D=除数
      MVI   E, 9      ;位计数器
      MOV   A, B
DIVO: MOV   B, A
```

```
        MOV   A, C
        RAL                    ;循环右移，最高有效位移至进位
        MOV   C, A
        DCR   E
        JZ    DONE
        MOV   A, B
        RAL
        SUB   D
        JNC   DIVI             ;YES，跳转到DIVI
        ADD   D                ;NO，恢复除数
        JMP   DIVO
DIVI :  RAL
        MOV   B, A
        MVI   A, 0FFH          ;求补商
        XRA   C
        MOV   C, A
        MOV   A, B
        JMP   DIVO
DONE :  —
```

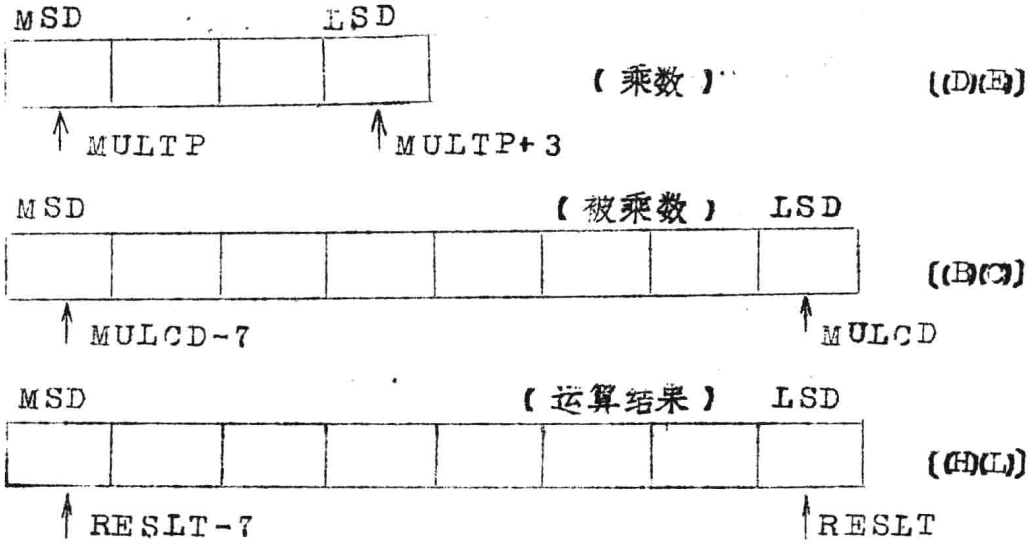
〔例10：十进制乘法运算程序〕

我们假定：

乘数的地址存放在DE寄存器（MULTIP + 3）

被乘数的地址存放在BC寄存器（MULCD）

运算结果的地址存于HL寄存器（RESULT）



以上可知，乘数只占用了4个存贮单元，共放8位十进制数。被乘数占用了8个存贮单元，共放16位十进制数。运算结果占用了8个单元，故存放16位十进制运算结果。本例列出的程序，执行十进制16位×8位运算大约为20毫秒时间。程序编码如下：

```
MULTN:LXI   H, RESLT    ;设定运算结果的地址
      MVI   B, 8        ;设定循环次数，'8'→B
      XRA   A           ;清除累加器
LOOPI:MOV   M, A        ;
      DCX   H           ;  清除运算结果存放区
      DCR   B           ;
      JNZ   LOOPI       ;
      LXI   H, MULCD-7  ;设定被乘数地址
      MVI   D, 4        ;设定循环次数'4'→D
LOOP2:MOV   M, A        ;
      INX   H           ;  被乘数高8位清除
      DCR   D           ;
      JNZ   LOOP2       ;
```