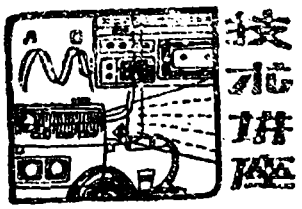
DOI:10.16157/j.issn.0258-7998.1985.08.015



四位微计算机的功能及其应用

第四讲　四位机程序设计初步(上)

温州电子技术研究所　缪晓胜

在四位机应用系统的设计中，除了进行必需的硬件电路(包括接口电路与辅助电路)设计之外，重要的一环是进行应用软件即专用程序的设计。下文先介绍基本的程序结构，然后介绍一些典型的实用程序。由于四位机的结构与指令都是大同小异的，因此掌握了一种四位机的编程方法后，再去掌握其它的四位机就轻而易举了。

流程图是程序执行次序的图解表示，是描述程序的最好方法，也是帮助人们编制程序、交流程序设计思想的有力工具。流程图的各种基本符号都很明了易懂，人们均已熟悉，这里从略。

本文中程序的书写，一般按以下格式：

标号：汇编符号指令；注解

一、四位机基本程序结构

正象任何复杂的数字电路都是由与、或、非这三种门电路组合成一样，各种复杂的应用程序也可由几种最基本的程序模块构筑而成。

1. 顺序(直线)程序、程序转移及分枝程序

按照指令的书写次序即计数顺序执行的程序谓之"顺序程序"见图 4-1。这时当前指令的执行结果不影响下条指令的地址。

由于 DG0040 PC 的特殊分页结构，在一页中程序执行完毕时不能自动转到下一页，需用双字节指令转到本区的下一页或另一页，或用三字节指令转到另一区中。当然，转移指令不一定非要排在最后，任意地址均可。转移后的地址也不必一定是第一个地址(0 号)，其情况见图4-2。

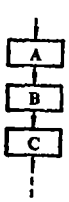


图4-1　顺序程序

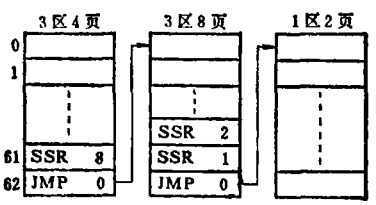
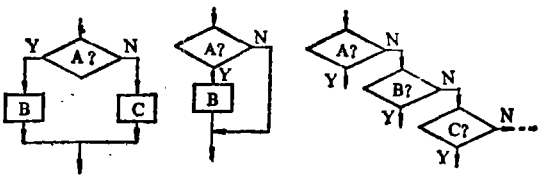


图4-2　程序跨区跨页转移

某些程序可根据中间结果自动选择运行的途径，称之为"分枝程序"。主要有如图4-3所示几种形式：

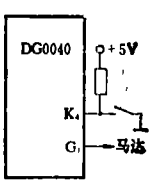


a.IF-THEN-ELSE结构　b.IF-THEN结构　c.树状分枝

图4-3　分枝程序

图4-3a对A框中的条件进行判别，如符合则执行B框，否则执行C框，然后均执行下面程序。图4-3.b中如条件不符合时直接转至下面，是a图的特殊形式。图 4-3.c 则是对一连串的条件进行判断，分别转至不同的地方。

例1：假如用G_1端控制电机启停，K_4端输入开关触点信号。如检测到开关闭合后("0"电平)，复位G_1，关闭马达。其控制程序如图4-4和图4-5。



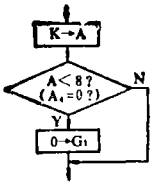


图4-4　马达控制示意图　　图4-5　马达控制流程图

```
KTA     ; K→A
ADX  8  ; A+8→A, C4=0跳
RG   1  ; 0→G1
```

该例中如增加一个条件：检测到开关断开时置位G_1，启动马达，其程序如下，流程图如图4-6。

```
KTA
ADX  8
JMP  RSG
SG   1
```

```
        JMP   NEXT
RSG:    RG    1
NEXT:
```

注意，上述程序是示意说明，在实际应用中还需编制相应软件，克服开关闭合时的抖动现象。

从上例可看出，条件转移分枝程序是利用判跳指令实现的。在DG0040机中，还可利用LTSPU指令实现多路分枝程序。其程序如下，流程图如图4-7。

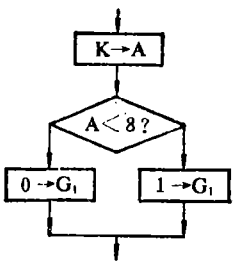


图 4-6

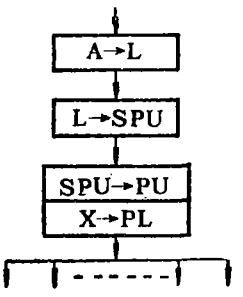


图4-7 多路分枝程序

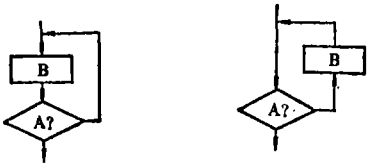
```
ATL       ; A→L
LTSPU     ; L→SPU
JMP    x  ; SPU→PU, x→PL
```

例如在一个应用系统中安排了10个功能键。通过键盘测试程序我们可求得每次按键的编码(如从0到9)存在A中。测键后如判别出有按键，应转到相应键的处理程序中去。对此，我们可把这些程序分别安排在0～9页。执行上述程序后，根据按键的不同，将自动转到相应的页去。这种分枝程序在分枝较多时效率很高。

2. 循环程序

对于一些要反复执行多次的重复过程，如用顺序程序来完成，就显得冗长累赘，这时用循环程序是最有效的。

循环程序有以下要素：循环变量，其初值、终值和步长以及循环体。循环变量控制循环的次数，循环体则完成所重复的实质性操作。一般可有如图 4-8 所示二种结构形式。



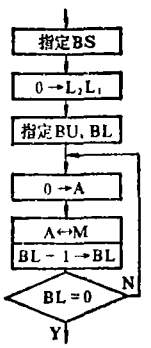
a.DO-UNTIL(作…直到) b.DO-WHILE(作…当)

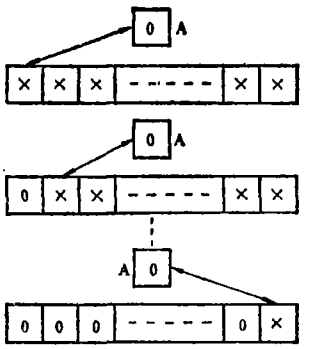
图4-8 二种循环结构形式

二者均在A框中对循环变量作修改并进行判别，条件满足后即脱离循环，否则继续循环。B框则完成实质性操作。二者的区别是图a中循环体至少执行一次，图b则可以完全不执行；在相同条件下，前者也要比后者多执行一次。

例2：清0程序，用以完成对RAM任一寄存器任意长度单元的清0。程序如下：

```
        LBS   x   ; x→BS, x→L2L1
        RNP       ; 0→L2L1
        LB    x,y ; y→BU, x→BL
CLR:    LAM 0;    0→A
        EXCD  0   ; A(0)↔M, BL-1→BL
                                并判跳
        JMP   CLR ; 若BL≠0, 返回CLR
                                继续清0
        RET
```





a.清0流程图 b.清0示意图

图4-9

清0流程图和示意图如图4-9所示。

本例中循环变量是BL，循环体是LAM 0及EXCD 0二条指令，即把A清0后和RAM当前单元内容交换，实现0→M的目的。同时对循环变量BL作减1(循环步长)修改，如不为0(注意此处BL为0判跳指BL减1修改前的值)，则返回到标号CLR处继续循环，直到0号单元也清0为止。从图4-9b可看出每次循环所做的工作。

由于EXCD指令要作BS⊕L2L1→BS的操作，所以增加了第二框0→L2L1，使得在执行EXCD指令后BS保持不变，在同一个寄存器中继续做清0的工作。

3. 子程序(参见图4-10)

在程序的几个地方要执行同一段程序，我们可把这段程序单独抽出来作为"子程序"，在要执行该段程序的地方用CALL指令实行"转子调用"。CALL指令执行时，计算机把下条指令的地址PC+1压入堆栈，同时将子程序入口地址送入PC(注意，三字节调用时PS、PU、PL均送，双字节调用时仅送PU、PL，单字节则仅送PL，图4-10中符号SBR仅指PL值，PS、

PU值由SSR指令送)。因此，计算机立即转去执行子程序。执行完毕遇到RET(或RETSK)指令时，又将堆栈依次弹出，最高级SA的内容送回PC，从而返回到主程序的原来地方(或跳过一条)继续执行。

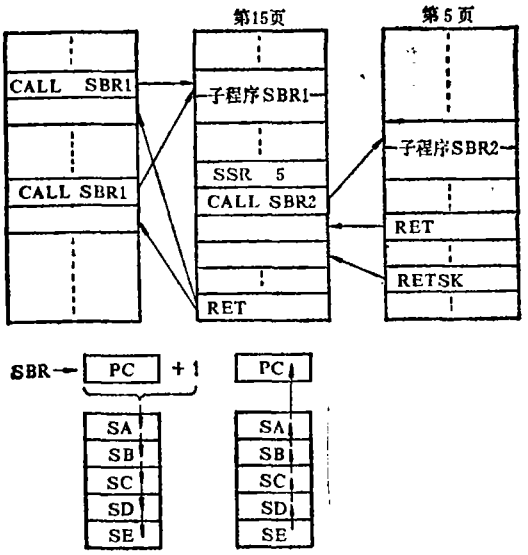


图4-10 子程序调用、嵌套及返回

显然，这将节省指令，提高了效率。在子程序中还可以进行调子，形成一层套一层的"嵌套"形式。当然，嵌套不能超过五级。安排在各区第15页的子程序，可用单字节指令进行调用，而安排在其它区、页的子程序，要用二字节或三字节指令调用。

例3：对RAM全部单元进行总清，一般用于开机时的初始化工作。我们可利用例2的CLR程序作子程序，并应使RAM处于长数据格式(R=0)的设置，采用二级嵌套子程序结构：

```
        LBS   0        ；指定0区
        CALL  CLRS     ；清0区
        LBS   1        ；指定1区
        CALL  CLRS     ；清1区
        LBS   2        ；指定2区
        CALL  CLRS     ；清2区
        LBS   3        ；指定3区
        CALL  CLRS     ；清3区
          ⋮
        同区第15页
CLRS:   RNP            ；0→L2L1
        LB    F,0      ；指定0号寄存器
        CALL  CLR      ；清0号寄存器
        LB    F,1      ；指定1号寄存器
        CALL  CLR      ；清1号寄存器
        LB    F,2      ；指定2号寄存器
        CALL  CLR      ；清2号寄存器
        LB    F,3      ；指定3号寄存器
        CALL  CLR      ；清3号寄存器
        RET
CLR:    LAM   0
        EXCD  0
        JMP   CLR
        RET
```

二、数据传送程序

数据传送是计算机最基本的操作，不管在数据处理或输入输出工作中使用都很频繁。DG0040是单累加器结构，所有的数据传送工作都必须通过累加器A。根据该机数据地址寄存器B的结构特点，数据以寄存器为单位存放为宜。在长数据格式时一个寄存器最多可存12～16位十进数，短格式则只能存放7～8位。数据传送程序可分为二类：寄存器之间的传送和移位程序。

1. **寄存器单向传送程序**：程序框图如图4-11，传送示意图如图4-12。

例4：$M^0_{0(F\sim0)} \rightarrow M^0_{1(F\sim0)}$ (R=0)

```
        LBS   0    ；0→BS
        LB    F,0  ；0→BU，F→BL
LOOP:   LDA   1    ；M→A，BU⊕01→BU
        EXCD  1    A↔M，BU⊕01→BU，
                   BL-1→BL并判跳
        JMP   LOOP；若BL≠0，则返回LOOP处
```

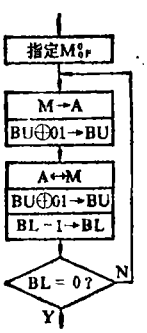


图 4-11

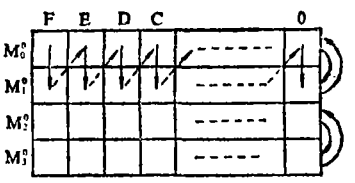


图 4-12

该程序把0区0号寄存器$M^0_{0(F\sim0)}$中16个单元的数据逐次通过A分别送到0区1号寄存器$M^0_{1(F\sim0)}$的相应单元中。其中从LOOP开始的循环体完成实质性操作。只要改变其初始条件(包括BS、BU、BL以及R、W)，该程序将完成完全不同的操作。表1略举数例。

表 1

序号	BS、BU、BL初始值	R	W	操作
A	$M^0_{1B}(L_2L_1=0)$	0	/	$M^0_{1(B\sim0)}\rightarrow M^0_{0(B\sim0)}$
B	$M^2_{3C}(L_2L_1=2)$	0	/	$M^2_{3(C\sim0)}\rightarrow M^0_{2(C\sim0)}$
C	$M^2_{3C}(0\rightarrow L_2L_1)$	0	/	$M^2_{3(C\sim0)}\rightarrow M^2_{2(C\sim0)}$
D	$M^0_{0F}(L_2L_1=0)$	1	0	$M^0_{0(F\sim8)}\rightarrow M^0_{1(F\sim8)}$
E	$M^0_{0F}(L_2L_1=0)$	1	1	$M^0_{0(F\sim8)}\rightarrow M^0_{1(1\sim0)}$

从表1中可以看出，数据传送方向及字长取决于以下因素：

(1) L_2L_1：是BS的异或修改条件，由于第一条LBS x指令同时执行x→L_2L_1，因此传送工作将在x区和0区之间进行(因x⊕x=00)。如在该指令后插入RNP指令将L_2L_1清0，则传送将在x区之间进行。例B和例C是这二种情况的说明。

(2) 循环体中指令LDA x及EXCD x中的x，是BU的异或修改条件(注意二个x必须相同)。改动x，将在不同的寄存器对之间传送。如将x改为3，寄存器对将变为0号和3号，1号和2号(00⊕11→11，01⊕11→10)。

(3) R、W：R决定数据格式，当R=1时处于短格式，还要由W的情况来决定传递方向。由于R、W在程序执行中可能会改变，需进行动态分析，记住其状态，否则同样的程序会得到完全不同的结果。

(4) BS、BU和BL的初始值：前二者确定所要传送的寄存器，后者则确定传送的位数。因为LB x,y指令只能指定5个单元(如长格式时是F、E、D、C、O)，对于较短字长的数据传送，需在指定地址后用INCB或DECB指令将BL增减至数据的起址。

例5：$M^0_{3(C\sim8)}\rightarrow M^0_{2(4\sim0)}$ (R=1，W=1)

```
        LBS   0
        LB    E,3
        DECB
        DECB
LOOP:   LDA   1
        EXCD  1
        JMP   LOOP
```

例4、例5都是按照从高至低的次序逐位传送的。也可以反过来先从低位做起。

例6：逆序传送程序$M^1_{0(0\sim6)}\rightarrow M^3_{2(8\sim E)}$ (R=1，W=1)

```
        LBS   1   ; 1→BS
        LAM   2   ; 2→A
        ATL       ; A→L
        LB    0,0 ; 指定M^1_00
LOOP:   LDA   2   ; M→A, BU⊕10, BS⊕10
        EXCI  2   ; A↔M, BU⊕10, BS⊕10,
                  BL+1并判跳
        JMP   LOOP
```

2. 寄存器双向交换程序

在许多场合，我们需要进行的是寄存器之间的数据交换，而不是单向传送。为此只需对上述程序略加修改。如例4改成：

例7：$M^1_{0(F\sim0)}\leftrightarrow M^2_{1(F\sim0)}$ (R=0)

```
        LBS   1
        LB    F,0
        LAM   3
        ATL
LOOP:   LDA   1
        EXC   1   } ;完成数据交换
        EXC   0
        LDA   0   ; 修改BS，回到1区
        DECB      ; BL减1并判跳
        JMP   LOOP
```

其中LDA 0这条指令仅完成修改BS的作用，取数功能实质上是空操作。只要适当改变初始条件，该程序也能完成不同寄存器对之间的传送。

例7的程序还可改写为下述形式，这样节省了二条指令，但失去了通用性，仅能用于固定的这二对寄存器间的传送。

```
        LB    F,0
LOOP:   LBS   1
        LAM   3
        ATL
        LDA   1
        EXC   1
        EXCD  0
        JMP   LOOP
```

3. 移位程序

移位程序是指将寄存器各单元内容左移或右移若干位。

例8：右移1位程序，流程图见图4-13。

```
        LBS   0
        LB    F,0
LOOP:   EXCD  0
        JMP   LOOP
```

不难分析，所做的工作是：A→$M^0_{0F}\rightarrow M^0_{0E}\rightarrow\cdots\rightarrow M^0_{00}\rightarrow$A。因此，如要将最高位$M^0_{0F}$清0，可在事先执行LAM 0指令先把A清0。反过来如在最后增加一条EXC 0指令，则M^0_{0F}单元将接收M^0_{00}单元的原有内

容，实现了寄存器的循环移位。因为上述程序执行完毕后A中存放的是M^0_{00}原有值，而RAM地址已指向M^0_{0F}（注意：BL=0时再减1变成F，这是R=0时的情况。当R=1时BL=0再减1将变成7，BL=8时减1才变为F。因为R=1时BL_4脱离BL的计数链，仅受W的异或作用影响。）

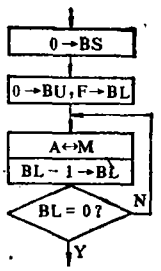


图4-13 右移程序流程图

左移程序与此类似。多位的移位程序，可多次调用上述移位子程序来完成，也可采用一些巧妙办法。读者可试编一下，对于掌握四位机的数据格式将会有所俾益。

三、四则运算程序

四位机当然也能作二进制、十六进制的运算，但是为了和输入输出时人们的习惯数制十进制一致(如键盘送数、数码管显示、打印等)，最常用的数制是BCD制。四位机的四位字长结构也最适宜于采用这种数制。

1. **BCD加法** 加法流程图和示意图如图4-14和图1-15。

例9：$M^0_{0(B\sim0)}+M^0_{1(B\sim0)}\rightarrow M^0_{1(B\sim0)}$

```
    LBS  0
    LB   0,0
ADDB: RC        ; 0→C
ADDL: LDA   1   ; M→A, BU⊕01
      ADC       ; A+M+C→A
```

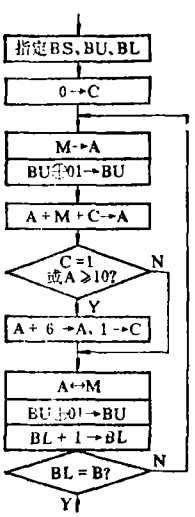


图4-14 加法流程图

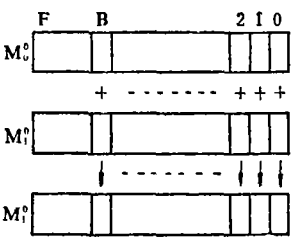


图4-15 加法示意图

```
    DAA         ; 十进制调整A
    EXCI  1     ; A↔M, BU⊕01, BL+1
                  并判跳
    JMP   ADDL
```

该程序的执行过程如下：每次从第一个被加数寄存器(本例为M^0_0)中的当前单元取数送至A，同时修改RAM地址指向第二个被加数寄存器(本例为M^0_1)的同一单元。然后执行带进位相加，并作十进制调整。进位触发器开始时要清0：0→C，以后保存本位相加结果的进位值，以向高位传递。运算后结果送进第二个被加数寄存器的同一单元，同时修改RAM地址，使其指向第一个寄存器的高一单元。如此循环直至由BL判跳值所限定的所有单元均相加完毕为止。

加法程序是在二个寄存器之间进行的，第二个寄存器还兼作结果寄存器，此时其源数据被破坏。寄存器及其字长范围的指定和数据传送程序一样，受RAM数据格式的影响，只要改变它们的初始条件及修改条件(LDA和EXCI二条指令)，就可以在任意二个区的二个寄存器的任意字长单元之间进行加法运算，长格式和短格式均可。

C保存各位的进位值，结束时表示运算结果有否溢出。

2. **BCD减法** 减法流程图如图4-16。

例10：$M^0_{0(B\sim0)}-M^0_{0(B\sim0)}\rightarrow M^0_{0(B\sim0)}$ (R=0)

```
    LBS  0
    LB   0,1
SUBB: SC        ; 1→C
SUBL: LDA   1   ; M→A, BU⊕01→BU
      CADCSC    ; Ā+M+C→A, C; C4=1
                  跳
      ADX   A   ; A+10→A
      EXCI  1   ; A↔M, BU⊕01→BU, BL
                  +1→BL并判跳
      JMP   SUBL
```

减法运算用补码进行，由CADCSC(SUB)指令完成。在作第1位运算前，必须先将C置1。以后C作为低位向高位的借位直接参与运算。够减时，C=1，不影响下一位的取补；不够减时，C=0，使下一位执行CADCSC时结果少1 ($\bar{A}+M+C=\bar{A}+M+0$)，从而实现了借位。同时不够减时还要作十进制调整，即减过6，由ADX A指令完成。注意该指令虽有判跳功能，但当参与运算的二个数都是BCD数时在任何情况下都不会跳步，读者可自行分析。

运算后如C=1，说明够减。若C=0，说明不够减，且结果是BCD补码，需对该数再取补一次，并将符号变反，才能输出机外。

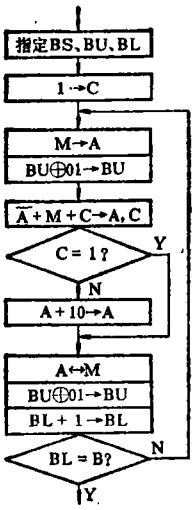


图4-16 减法流程图

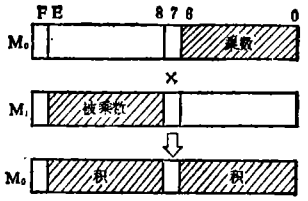


图4-17 乘法示意图

3. BCD乘法 其示意图如图4-17，流程图如图4-18。

例11：$M_{0(6\sim0)}\times M_{1(E\sim8)}\rightarrow M_{0(E\sim8)+(6\sim0)}$ (R = 1, W = 0)

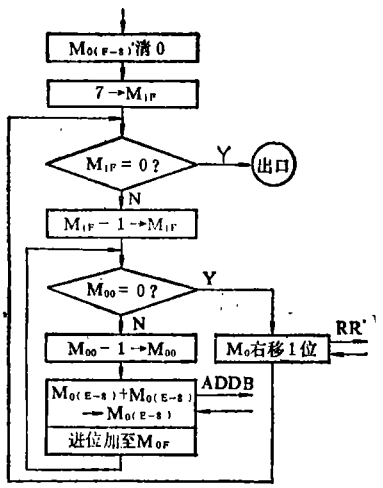


图4-18 乘法流程图

```
MUL:  LB F, 0; 指定M0F
      CALL CLR; M0(F~8)清0, 调用例2清
               0程序
      LB F, 1; 指定M1F
      LAM 7; 7→A
      EXC 0; 7→M1F
MUL1: LB F, 1; 指定M1F
      LDA 0; M→A
      ADX F; A+15→A, C4=0跳
      JMP MUL2; 若A≠0(C4=1),转下面
                MUL2
END:          ; 若A=0(C4=0), 结束
MUL2: EXC 0; (A+15)=(A-1)→M1F
MUL3: LB 0, 0; 指定M00
      LDA 0; M→A
      ADX F; A+15→A
      JMP MUL4; A≠0, 转MUL4
      LB F, 0; A=0, 指定M0F
      CALL RR; M0(F~8)右移1位, 高位
               送0
      LB 7, 0
      DECB; 指定M08
      CALL RR1; M0(8~0)右移1位
      JMP MUL1; 继续循环
MUL4: EXC 0; A+15=(A-1)→M00
      LB 8, 1; 指定M18
      CALL ADDB; 调用例9加法子程序
      LB F, 0; 指定M0F
      LAM 0; 0→A
      ADC; 加进位:(A+M+C)=(M+C)
           →A
      EXC 0; 交换进M
      JMP MUL3
RR:   LAM 0
RR1:  EXCD 0
      JMP RR1
      RET
```

乘法程序可分为定点和浮点二种。定点乘法的小数点位置是固定的，最简单的情况是整数的乘法，即小数位为0。

本例程序实现了7位整数对7位整数的乘法。被乘数7位存在$M_{1(E\sim8)}$，乘数7位存在$M_{0(6\sim0)}$，乘积14位存在$M_{0(E\sim8)+(6\sim0)}$。M_{1F}存乘法循环次数，实际即乘数的位数，事先送7。乘法运算的过程和手算时类似，每次将乘数的某1位乘上被乘数。先从最低位M_{00}乘起。首先判别一下M_{00}是否为0，为0就结束该位的乘法；不为0则减去1，同时将被乘数加至乘积区高位$M_{0(E\sim8)}$，并将相加的进位加至M_{0F}，然后返回MUL$_8$处继续该位的乘法，直至$M_{00}=0$为止。乘完1位后，将乘积及乘数串接起来右移1位(注意这里2次调用的右移子程序入口不一样，第一次是RR，首先做0→A的操作，以使M_{0F}清0。第二次是RR1，因这时A中存放的是原来M_{08}的值，需将其移至M_{00}，所以不能清0)。这样返回MUL1时，M_{00}中存放的是乘数的高一位，乘积区$M_{0(E\sim8)}$也相应地存放着高一位的乘积(原来$M_{0(F\sim9)}$的内容)，继续执行乘法循环体

时，实际上是进行高一位的乘法。直到$M_{1F}=0$为止，把7位乘数全部乘完，结束乘法过程。

该程序可在RAM任一区中运行，但指定BS后需将L_2L_1清0（因此符号中未指定区号）。

在实际应用中被乘数和乘数的位数变动范围很大，在内存中存放的格式也不尽相同，这就要根据实际情况编制相应的乘法程序。本例中的被乘数和乘数可以在7位以下任意变动，但是M_{1F}必须等于乘数位数（当然也可选用其它单元）。当（被乘数位+乘数位）≤7位时还可少占一个寄存器（短格式的）。而当（被乘数位+乘数位）>14位时则必须用长格式的数据格式。

在作带小数的定点乘法时，乘积的小数位=（被乘数小数位+乘数小数位）（小数位数一般从低位算起，如12.345，其小数位数等于3）。如要限定乘积的小数位，只须把乘积右移（或左移）固定几位即可。

浮点的运算稍微复杂一些。必须用一个单元存放小数位及符号。如本例中可存放在M_{1F}（被乘数）及M_{07}（乘数）。单元中一般用最高二进制位M_{03}作符号位，后三位存小数位（即假定小数位在0～7之间）。相乘时先由被乘数的符号和乘数的符号求出乘积的符号，并把二者的小数位相加，得出乘积的小数位。相乘后如要限定乘积的位数（例如要求和被乘数一样多），那么还要作移位、四舍五入、溢出判别（即整数部分已存放不下）等工作。

4. **BCD除法** 示意图如图4-19，流程图如图4-20。

例12：$M_{0(6\sim0)}+M_{1(E\sim8)}\rightarrow M_{0(8\sim0)}$（R=1，W=0）

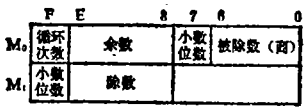


图4-19 除法示意图

```
DIV :  LB      F, 1   ; 指定M1F
       LDA     0      ; M→A
       LB      7, 0   ; 指定M07
       SC             ; 1→C
       CADCSC         ; Ā+M+C→A, C
       NOP            ; 空操作
       EXC     0      ; M07 - M1F→M07
       LB      F, 0
       LAM     6
       EXCD    0      ; 6→M0F
       CALL    CLR    ; M0(E~8)清0；调用例
                        2清0程序
```

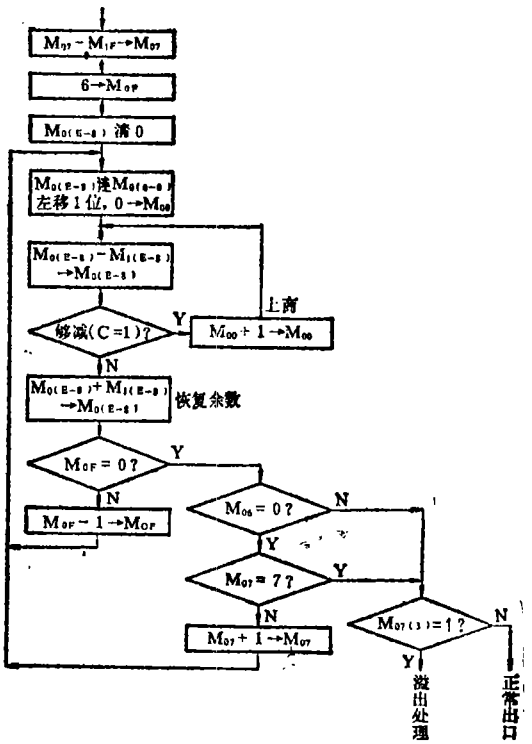


图4-20 除法流程图

```
DIV1:  LB      0, 0
       CALL    RL
       LB      8, 0
       CALL    RL1
DIV2:  LB      8, 1
       CALL    SUBB   ; M0(E~8) - M1(E~8)→
                        M0(E~8)，调用例10
                        减法子程序
       SKNC
       JMP     INC    ; 转去作M00+1
       LB      8, 1
       CALL    ADDB   ; 恢复余数，调用例9
                        加法子程序
       LB      F, 0
       LDA     0
       ADX     F      ; M0F - 1
       JMP     DIV3   ; 够减，存回M0F
       LB      7, 0
       DECB           ; 指定M00
       LAM     0
       TAM            ; M00=0?
       JMP     DIV4   ; 否
```

```
        LB      7, 0    ; 是
        LAM     7
        TAM             ; M07=7?
        JMP     DIV5    ; 否
        JMP     DIV4    ;
DIV5:   LAM     1
        ADD
DIV3:   EXC     0       ; M07+1→M07
        JMP     DIV1
DIV4:   LB      7, 0
        SKM8
END:                    ; 正常出口
ER  :                   ; 溢出处理
INC :   LB      0, 0
        LAM     1
        ADD
        EXC     0       ; M00+1→M00
        JMP     DIV2
RL  :   LAM     0
RL1 :   EXCI    0
        JMP     RL1
        RET
```

除法运算的小数位变动较大。举例来说，二个4位整数相除，最大值是9999÷1=9999，最小值是1÷9999≈0.0001，数值相差10^8倍，小数位数从0～4位。因此在除法中为了不丢失有效位数，一般应采取浮点运算，然后再根据实际要求将数值移位规格化成所需的格式。

本例是7位无符号数的浮点除法，被除数存在$M_{0(6\sim0)}$，其小数位值存在高位M_{07}，除数存在$M_{1(E\sim8)}$，其小数位值存在高位M_{1F}。相除后的商及其小数位取代被除数的位置，被除数被破坏，余数则存在$M_{0(E\sim8)}$。

除法的过程和乘法类似。逐次将被除数的高位部分减去除数，累减至不够减为止，再将除数加回被除数，以恢复该次余数值。累减的次数(M_{00}值)就是该位的商。这里把余数区作为被除数的高位延伸区，事先清0，由于一般来说事先不知道除数的实际位数是多少，所以把被除数逐位移入余数区。这样一共作7次，循环次数由M_{0F}单元控制。循环计数值减量及结束判跳放在循环体的后面，是DO-WHILE型结构，因此M_{0F}的初值应是7－1=6。

7次循环后被除数全部移进延伸区(即余数区)，位置和除数对齐，这时所得到的商的小数位等于事先求出存在M_{07}中的值，其数值在－7～＋7之间（负值是补码形式)。此时商可能已有7位（最高位$M_{06}\neq0$)，因此先判别一下M_{06}，非零就结束运算。否则再判别一下小数位(M_{07}值)是否已到7(这是我们假定所允许的最高小数位数，即全部7位数都是小数，在实际应用中不必一定如此)，为7也结束运算，如不为7则将其加1后继续求下一位的商。这时M_{07}可能为负值，即$M_{07(3)}=1$（补码时－1=1111，－2=1110，…)，说明商的整数部分尚未求全。当被除数的小数位数小于除数的小数位数时，求出7位有效数后($M_{06}\neq0$)，有可能M_{07}还是负数，说明商的整数部分大于7位，这称为"溢出"，即给出的数据格式无法容纳计算出的实际数据，应按要求作适当的处理。如在计算器中一般是显示"E"字。

（待续）

（上接第19页）

是完全可行的。尤其是一些社会需求量大的仪器，生产厂家很多，电路结构也基本相同，哪一个厂家重视仪器结构的改进，该厂家的产品就会在社会上具有竞争力。事实上，不断注意改进仪器结构的厂，仪器的电气技术性能也较好。譬如一些电子仪器厂生产的BT-3型扫频仪，现在省去了强迫风冷电扇，使用中也没发现仪器过热而影响技术特性的现象。侧板也改用松不落螺钉，这样整个仪器的造价不但下降，而且也降低了噪声，方便了用户的维修。与此同时，还有一些厂家生产带电扇的BT-3型扫频仪，其它地方还没什么改进，长期下去就会失去本厂产品在市场上的竞争能力。

七、关于仪器零配件的供应

仪器生产厂家往往重视整机生产而忽视零配件的供应，这也给用户造成了很多不应有的损失。譬如XFG-7型高频信号发生器的"载波调节"与"调幅度"调节电位器是线绕电位器，在使用中需经常调节，属易损件，但又是非通用件、市场上买不到。我们曾到仪器生产厂买过该零件，但该厂零件也基本上是按计划生产，只能满足我们的部分要求。XFG-7型高频信号发生器是一老产品，估计全国有近万台了，如果只因这两个电位器损坏而造成仪器报废，累计损失该有多大！希望生产厂重视对用户的零配件供应，可采用邮购的方法满足用户维修的需要。对社会需求量大的一些专用件，建议也应该提供给市场。

以上一些想法与意见，是本人在工作实践中因遇到具体问题而产生的，很不全面，如能给仪器生产厂家一些有益的启示，笔者将感到满足。