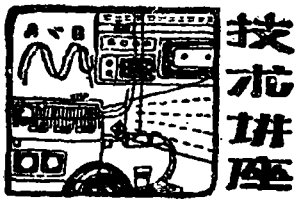
DOI:10.16157/j.issn.0258-7998.1985.09.013



四位微计算机的功能及其应用

第五讲　四位机程序设计初步(下)

温州电子技术研究所　缪晓胜

四、开平方程序

四位机常用于低速数据处理，如计算器或测试系统中物理量的换算等。这时，往往会遇到各种函数的计算问题。对此，首先要根据计算机的指令系统功能，内存容量及运算速度的要求，选择合适的算法。对于初等函数(如三角函数、对数函数等)，较流行的是采用座标旋转法，也可用级数展开法或其他一些巧妙的办法。限于篇幅，这里仅介绍一个最简单的函数—求平方根的算法及其程序流程。

有恒等式：$1+3+5+\cdots\cdots+(2n-1)=n^2$　　(1)

因此：$[1+3+5+\cdots\cdots+(2n-1)]\times 10^2=(10n)^2$

$=1+3+5+\cdots\cdots+(10\times 2n-1)$

对于给定的整数　$x=x_1x_2x_3\cdots x_n$(不妨假定n为偶数)

$x_1x_2=1+3+5+\cdots\cdots+(2n_1-1)+r_1,\ (r_1<2n+1)$

于是：$x_1x_2x_3x_4=x_1x_2\times 10+x_3x_4$

$=[1+3+5+\cdots\cdots(2n_1-1)]\times 10^2+r_1x_3x_4$

$=1+3+5+\cdots\cdots+(10\times 2n_1-1)$

$+(10\times 2n_1+1)+(10\times 2n_1+3)+\cdots\cdots$

$+(2n_2-1)+r_2$　　(2)

依此类推：$x=x_1x_2x_3\cdots\cdots x_n$

$=1+3+5+\cdots\cdots+(10\times 2n_1-1)$

$+(10\times 2n_1+1)+\cdots\cdots+(10+2n_2-1)$

$+(10\times 2n_2+1)+\cdots\cdots+(2n_l-1)+r_1,$

$(l=r/2)$

这样，对于一个有$2l$位数的整数，可把它分为l段，每段2位(如是奇数位，则第一段只有1位数)。根据恒等式(1)，用奇数级数逐次累减，减到不够减，把求得的末次累减项值$(2n_1-1)$加上1，就是这一段的平方根值的2倍。而由恒等式(2)可知，这样在高段所作的累减相当于在低段累减至$(10\times 2n_1-1)$项。因此移动地址指针后，可开始下一段的累减过程，直至减完最后一段。最终的累减值$(2n_l-1)$加1再除以2就得到平方根值n_l。

这种分段累减算法最多只需作$10l$次减法，而如果仅用(1)式直接从低位开始累减，最多则需作10^l次减法。当l较大时，二者相差极巨，由此可知算法之重要。

有小数的数值开方时首先把整数分段累减，然后对小数也同样是2位一段累减。

例13　12位浮点数的开方程序。

被开方数存在$M_{0(B\sim 0)}$，其小数位存在M_{0C}。$M_{1(B\sim 0)}$存累减值，开方根结果值及其小数位送回$M_{0(B\sim 0)}$和M_{0C}。M_{1C}存地址指针n，开始时送2，即对BL=C再减1二次，指向M_{1A}。指定段地址后首先送1($M_{11}=0$时执行：$1\rightarrow A$，$M_{11}+A\rightarrow M_{11}$，实即$1\rightarrow M_{11}$)，对被开方数$M_{0(B\sim 1)}$作累减。以后每次加上2($2\rightarrow A$，$M_{11}+A\rightarrow M_{11}$)，构成奇数级数作累减。减到不够减后，再判别$M_{0C}$(上一段累减余数的高位值)是否为0。如不为0(则必将为1，读者可自行分析)，将其冲0后继续累减，否则完成了该段的累减。由于这时累减已多做一次，所以应作一次加法以恢复余数$M_{0(B\sim 1)}$。

此时累减值为$(2n_l+1)$，已多作一次累减，将其减1。然后把被开方数的余数左移1位，高位送进M_{0C}，同时把段地址指针M_{1C}加1，这样实际上指针移动了2位，即1段。随后继续进行下一段的累减。

当指针$M_{1C}=12$时，已完成最末位M_{10}的累减。存在$M_{1(B\sim 0)}$中的是最终的累减值$(2n_l+1)$。最后完成$\frac{(2n_l+1)-1}{2}\rightarrow n_l$的工作。这里除以2的运算是用累加5次的办法实现的。

被开方数的小数位存在M_{0C}，因此$12-M_{0C}$就是被开方数的整数位。将其除以2便得到平方根的整数位，送M_{0D}中暂存。运算完毕后再作$12-M_{0D}$，求出平方根的小数位再送回M_{0C}。

注意，这个程序还不是完善的。如开头还需将被开方数左移消去前面的无效0以求得足够的有效平方根值。此外该程序实际上只进行了11段运算，因此最多只能得到11位有效数。

开方程序内存分配图及开方程序流程图见图5-1及图5-2。

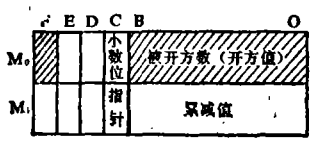


图5-1 开方程序内存分配图

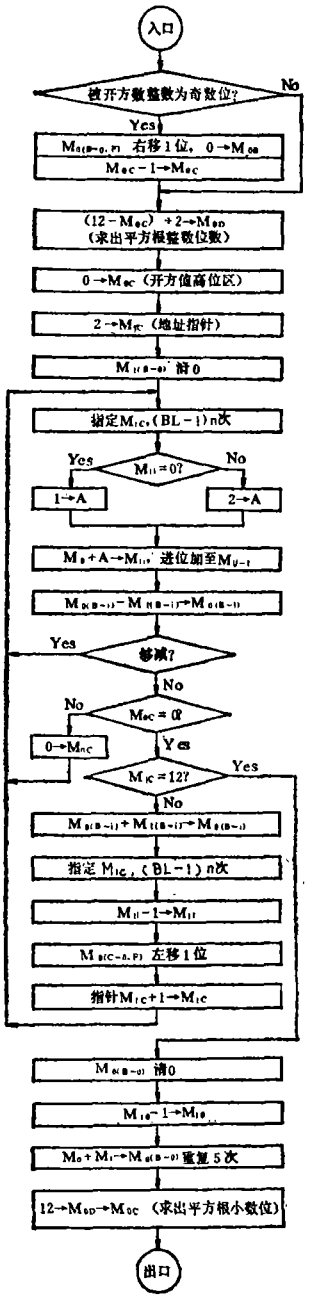


图5-2 开方程序流程图

五、延时/计数程序

计算机在处理外部事件时，往往有一定的"实时"时序要求，必须给出所需的时间间隔信号，即"定时"信号。

计算机实现定时有几种办法。一种是软件延时：让机器执行一般不影响机器工作任务的"空操作"程序，程序的运行时间即定时时间。另一种是利用内部或外部的硬件定时器，产生中断信号由计算机识别处理或产生溢出标志信号由计算机查询处理。这里只介绍前者。

由于机器指令执行时间很短，要提高程序的效率，必须尽量利用循环及嵌套的办法。原则上机内能进行计数判跳的寄存器都能作为延时循环变量单元，如A、RAM单元、BL、G等。但需注意执行时不能破坏机器的当时状态，而且RAM单元和BL不能同时使用，因为前者在作计数时BL值是固定的。

下例是一个双层循环的延时程序，内层由A控制，外层由BL控制。

A先送全1(F)，内层循环每次4条指令，对A减1直至0，共循环16次。外层循环由BL控制，当R=0(长格式)时也循环16次。总共执行指令(4×16+3)×16+2=1074条。当时钟频率为100KC时延时10.74ms。可用改变内层循环体的长度（增加或减少NOP指令）或改变外层循环BL的初值的办法来调整延时时间。

例14、利用BL及A计数的延时程序。 其流程图见图5-3。

```
        LB   F,0
DELAY1: LAM  F
DELAY2: NOP
        NOP
        ADX  F
        JMP  DELAY2
        DECB
        JMP  DELAY1
        RET
```

下例的程序既能作延时，也能用来对外部事件间隔进行计数。

例15、利用RAM单元的多用延时/计数程序。
其流程图见图5-4。

```
INIT;  LBS  0
       LB   0,0
       LAM  0
       LAM  0
```

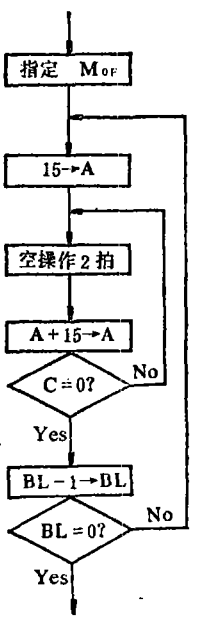


图5-3 延时程序流程

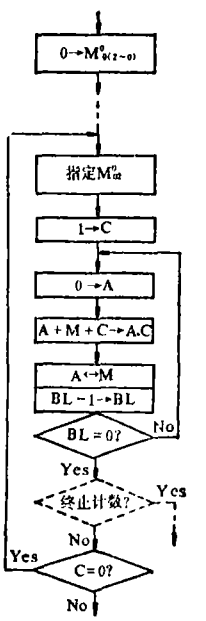


图5-4 延时/计数程序流程

```
        LAM   0
        LAM   0
        ……
CNT:    LBS   0
        LB    0,0
        INCB
        INCB
        SC
CNT1:   LAM   0
        ADC
        EXCD
        JMP   CNT1
        ……
        SKNC
        JMP   END
        JMP   CNT
END:    ……
```

该程序用3个RAM单元$M^0_{0(2\sim0)}$作循环计数变量，构成三位十六进制计数器，其中M^0_{02}为最低有效位(LSB)，M^0_{00}为最高有效位(MSB)。在执行前先由INIT程序将其清0或预置设定的值(以延时较短的时间)。从CNT1开始的四条指令完成加1计数工作，从低位(M^0_{02})向高位逐位运算。C事先置1，起加1的作用，同时作为低位向高位的进位，将3个单元联成为一个12位的计数器。最后计满为FFFH后再加1变成000H；计数溢出，从而C=1，由SKNC指令判跳转至END结束循环。

从CNT处开始，共执行$(3\times4+8)\times16\times16\times16=81920$条指令，当主频为100KC时，延时819.2ms。$M^0_{0(2\sim0)}$预置初始值后，延时时间最短可调整至$20\times1\times1\times1$条指令周期即200μs。在循环体中插入NOP指令，则可把延时时间拉长。

这个程序还可用来对外部事件的时间间隔计数。送出控制信号或接收到起始信号后进入本程序。在内层循环体后(或之前)插入一段测试判别程序，如框图中虚线框所示，(可从G口或K口输入测试信号)。未测到终止信号时继续循环计数，一旦测到立即脱离循环。由$(\overline{M^0_{02}}+1)\times(\overline{M^0_{01}}+1)\times(\overline{M^0_{00}}+1)\times$内层循环长度可算出外部事件的时间间隔。注意间隔不能超过程序的最长延时时间，否则必须增加计数单元。

此外，由于DG0040采用内部环形移相振荡器作时钟源，频率不稳定。在需要准确延时的场合，必须使用外部时钟源。

六、输入输出程序

DG0040机共有5组I/O端口：G、K、L、D和段译码端口，除后者是专用端口外，前面4个都是通用端口(其中L还兼作段译码的输入信号)。四位机的外设中，最常见的是键盘和数码管，这时常用D作为二者的扫描信号，用K作为键盘的扫描输入信号。而G端口可用来和外部进行双向数据通信或作为分离的输出控制信号。

在稍为复杂的应用系统中，上述I/O端口可能会不够用，这时应予以扩充。东光电工厂设计了专用接口片DG0046，功能很强，和G端口连接后能增加4个I/O端口和一个定时器/计数器。但在许多情况下，用简单的中小规模IC电路也能满足要求。

可用多种办法实现扩充：如对4位G信号进行译码，可得到16个独立的控制信号(见图5-5)；或用G作打入脉冲，从L端口输出数据，可得到4组4位并行锁存输出端口，G译码后则能得到16组；同样用G作选通脉冲，从K端口读入数据，则能增加多个4位并行输入端口，见图5-6。对这些扩充的外部输入输出电路，其控制程序均十分简单。下面介绍一个不占用G端口的I/O扩充实例。

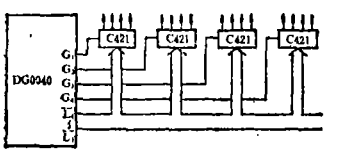


图 5-5

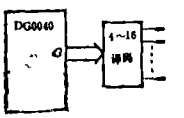


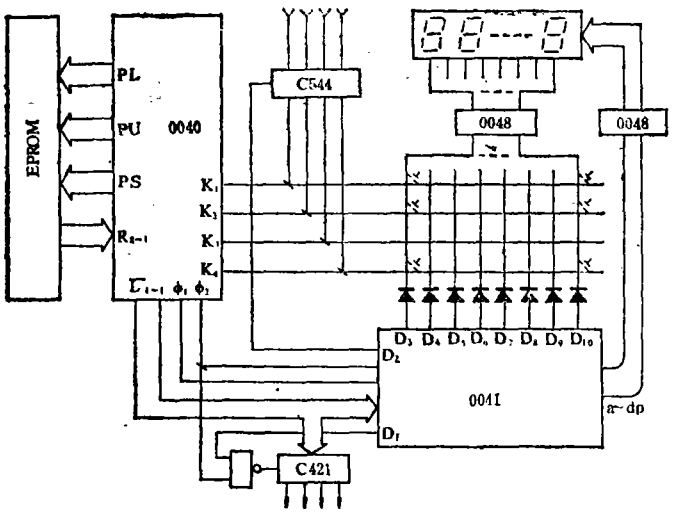
图 5-6

例15、输入输出实例。

本例在最小系统(由40、41和EPROM构成)的基础上，扩充了一个4位锁存输出端口和一个4位输入端口。D_1作为4位输出锁存器C421的打入脉冲信号，D_2作为4位输入选通门C544的允许信号。注意，由于D输出和L输出是在RNP指令中同时复位的，因此D_1必须和ϕ_2相与予以同步选通，在ϕ_2的后沿将数据存在C421。$D_3\sim D_{10}$则用于键盘和显示数码管的位扫描信号，$D_{11}\sim D_{15}$未用，见图5-7。

由于D是串行移位寄存器，为防止各条D输出线之间互相影响，在每次执行输入输出操作之前，必须先清除所使用的有关D寄存器(本例为$D_1\sim D_{10}$)，由下述子程序完成：

```
CLRD:   RNP
        LAM   9
LOOP:   SHD0
        ADX   F
        JMP   LOOP
        RET
```



注意：0041的D输出端及0048输出端均为OC门，需加负载电阻，图中未画出。

图5-7　输入输出实例

实际上是将D送0移位10次，由于D_{11}以后未用，可以不用考虑。

A内容输出至C421的程序如下：

```
        CALL  CLRD;  CLRD应存在第15页
OUT;    SHD1          ; 1→D1
        LDA   0       ; M→A,输出值先送至A
        SNP           ; A→L,1→NP:D1输出
        RNP           ; 0→L,0→NP:关D1
```

注意A内容送L寄存器后变反码($\overline{L}$)输出，如想得到原码信号，应从C421的$\overline{Q}$端引出。从C544输入的程序如下：

```
IN;  CALL  CLRD
     SHD1
     SHD0          ; 1→D2
     SNP           ; D2输出
     KTA           ; K→A
     RNP           ; 关D2
```

由于C544仅是一个门控，所以这种输入方式是非缓冲的，即在执行KTA指令的时候，输入信号必须出现在C544的输入端。

显示部分采用8位平板荧光数码管，每位的段驱动阳极已并在一起，由DG0041的段信号端驱动。各位的栅极分别由D_3～D_{10}信号驱动。由于荧光管阳栅极需高压驱动，东光电工厂研制了配套的DG0048八位高反压驱动电路，这里用了二片。如显示位数＞8位，则需增加1片(图中灯丝交流4.8V电路未画出。)

D_3～D_{10}还同时作为键盘矩阵的行扫描线，K_1～K_4作为列输入线，在每个交叉点可安排一个按键，总共可设8×4=32个键。为了防止同一列中二个键同时按下造成行线之间的短路，行扫描线用二极管予以隔离。

显示和测键的扫描同时进行。D_3～D_{10}的每一位依次送出一个适当宽度的脉冲，驱动对应位数码管的栅极，同时将该位的数值由0040的A中送往0041的笔段译码器中，译成段信号后驱动数码管的阳极。这样每次显示一位，由于视觉的暂留性，看到的是一组完整稳定的数字信号。这种动态扫描显示的办法，电路简单，成本较低。但应注意显示时每位停留的时间（即位脉冲宽度）要适中，如太短，会减少有效驱动时间，降低亮度；太长则会导致数字闪耀。一般在1ms左右即可。显示测键信号波形图如图5-8所示。

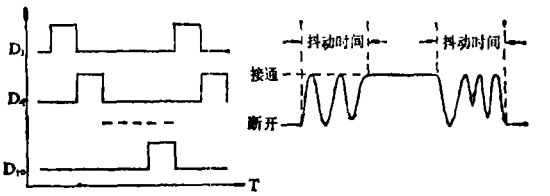


图5-8　显示测键信号波形　　图5-9　按键抖动波形图

在每位扫描脉冲的末了对键盘进行测试：把列信号从K端口读进A，由程序判别分析该行四个键的按键情况。

按键接下和松开时触点的通断是经过一段时间(约1～10ms)的抖动才到达稳定状态。图5-9示出按键抖动波形情况。为了避免一次按键接收一连串信号的错误，计算机通常用软件消除抖动现象：在第一次测试到按键信号后，延时一般时间避开抖动区(可重新测试)予以确认。在测试到按键释放后再延时一般时间予以确认。具体实现可有几种办法：①用专门一般延时子程序避开抖动，按键的确认可在接通延时之后或释放延时之后。后者等按键松开后才执行，响应较慢。②对已有机内定时基准的情况(如硬件定时器、软件延时循环等)，可设软件计数器对定时基准计数，以延时固定的间隔来避开抖动。这种办法刊载介绍的较少，本文予以详述。

在本例的情况下，我们用显示测键程序的延时间隔作定时基准。如每位停留显示0.5ms左右，那么8位一共是4ms，4次循环就构成了16ms。在内部RAM中设立1个计数器T和2个标志位F_1、F_2。F_1是本次测键有无按键的标志，F_2是以前的状态。当F_1=1、

$F_2=0$时是键刚按下的情况，这时把计数器送初值4，同时置位F_2，表示按键已按下。在以后四次测键中，将不进入分析程序，仅对计数器减1处理后立即返回。这样累计四次测键共延时16ms，避开了抖动区。按键松开时的情况也是同样。按键去抖流程图如图5-10，按键分析状态表见表5-1。

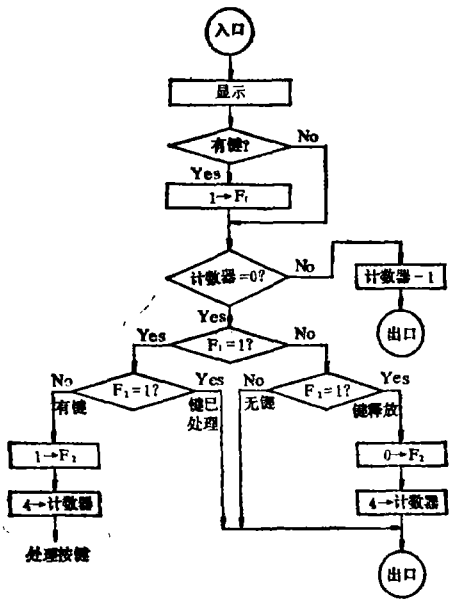


图5-10　按键去抖流程图

表5-1　按键分析状态表

	F_1	F_2	操　　作
无按键	0	0	
刚按下	1	0	计数器初始化，1→F_2
按下后	1	1	
刚放开	0	1	计数器初始化，0→F_2

实际的程序及流程图如图5-11所示。用机内硬件标志触发器Z作为F_1，RAM位$M^0_{0F(3)}$作为F_2。计数器则由低3位$M^0_{0F(2\sim0)}$构成。初始化时送全1(111)，四次减1后变成011，因而$M^0_{0F(2)}$从1变0，这样程序判别十分方便。确定有按键后，需再行快速扫描一次，以确认该按键的行号和列号的编码，它们分别存放在M^0_{0E}和A中。本例在硬件构置上及程序中都没有占用G这个唯一的攻向I/O端口，给实际应用提供了方便。

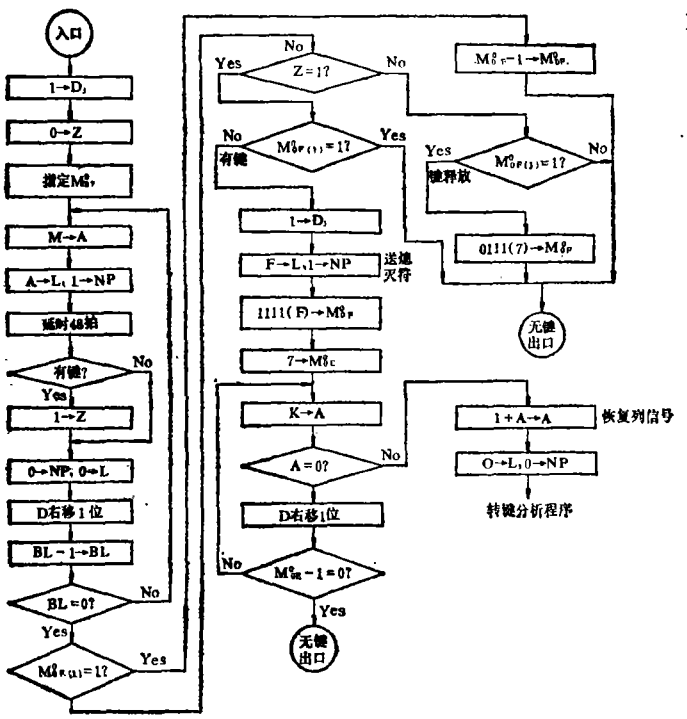


图5-11　显示/测键流程图

```
        (R = 1, W = 0)
        CALL  CLRD
        SHD1          ;
        SHD0          ; 指定D3
        SHD0          ;
        RZ            ;0→Z
        LBS   0
              LB    7,0     ;指定L0 7
DIS:    LDA   0       ;显示内容送A
        SNP           ;A→L, 1→NP: 送显示
        LAM   F
DELAY:  NOP           ;延时48拍
        ADX   F
        JMP   DELAY
        KTA           ;输入键盘列信号
        ADX   F
        SZ            ;有键把Z置位
        RNP           ;关显示
        SHD0          ;D移位
        DECB          ;BL-1, 指向下一位
        JMP   DIS     ;未完，继续显示
        LB    F,0
        SKMP  2       ;M0F(2) = 1?
        JMP   KA      ;否
DECT:   LAM   F       ;是, M0F - 1
        ADD
        EXC   0
        JMP   END
KA:     SKZ           ;Z = 1?
```

```
    JMP KA2
KA1:SKMP 3      ;M(0,F(3)) = 1?
    JMP KEYIN
    JMP END
KA2:SKMP 3      ;M(0,F(3)) = 1?
    JMP END
    LAM 7       ;0111→M(0,F)(同时送F2及T)
    EXC 0
END:……         ;无键出口
KEYIN:CALL CLRD
    SHD1
    SHD0
    SHD0
    LAM F       ;送熄灭符
    SNP
    LB F,0
    EXCD0
    LAM 7       ;行扫描码初值送M(0,E)
KEYIN1:EXC 0
    KTA         ;输入键信号
    ADX F       ;有键(K≠0)?
    JMP KEYIN2  ;是, 转出
    SHD0        ;否, D移位
    LDA 0       ;M(0,E)-1
    ADX F
    JMP KEYIN1  ;继续测键
    JMP END     ;无键出口
KEYIN2:ADX 1    ;恢复键盘列信号
    NOP         ;
    RNP         ;
    ……          ;转键分析程序
```

七、键盘分析程序及状态矩阵(状态表)法

用四位机构成的应用系统大多通过键盘接受各种操作命令和数据。按照机器接收一条操作命令所需的按键次数来说，操作命令分为单键命令和多键命令，按键也可分为单义键和多义(复合)键。单键命令和单义键的明显例子如"启动"、"中止"、"复位"等机器动作控制命令键。而多义复合键的典型例子可见诸于各种高档的计算器。如某键在单独使用时为求平方值X^2，而按了[2nd](Second)键后再按该键变成求平方根$\sqrt{X}$。

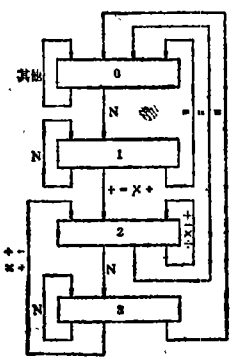
对于多键命令和多义键，各个键之间存在着某种制约关系，同一组键在按照不同的次序使用时其含义可以完全不同。例如设想设计一个能进行整数四则运算的最简单的计算器，共有15个按键：[1]~[9]、[+]、[-]、[×]、[÷]、[=]。让我们看一个操作实例：3×4×5=60。在按第一次[×]键时，不启动运算，而当按第二次[×]键时，要启动乘法程序得出中间结果12并予以显示，然后等待接收新数，和第一次的作用不同。再看该例的错误操作情况：在按第一个运算键时如错按成[÷]键，那么再按一下正确的[×]键，仍然恢复正确的操作(称为"纠错"功能)。这时虽已按了二次运算键，但由于只送了一个操作数，所以并不启动乘法程序，和前面的情况又不一样。

从上例可以清楚地看出，当机器处于不同状态时，同一按键的操作不尽相同。设计这些功能比较复杂的键盘分析程序时，虽也可采用通常的直接分析法，但较好的办法是"状态矩阵法"(即状态表法)。它把整个键盘分析程序看作为一个"系统"。"系统"可处于各种不同的"状态"下，确认后的按键信号是该"系统"的输入条件，决定执行何种操作并转移到哪一个新的状态(或不转移)。

在用状态矩阵法设计分析程序时，首先要画出状态流向图。图中的方框称为"状态方框"，表示程序中各种不同的状态。带箭头的状态转移线表示状态转移的方向，其相应的输入条件称为"表语"，可标在转移线的边上或中间断开处。对于计算器例，我们可画出状态流向图如图5-12。

表5-2 计算器键盘分析状态表

按键	当前状态	下一状态	执行操作
数字键0~9	0	1	送新数
	1	1	继续送数
	2	3	送新数
	3	3	继续送数
运算键+ − × ÷	0	0	
	1	2	存数，存运算符
	2	2	改变运算符
	3	2	运算，存运算符
等号键=	0	0	
	1	0	
	2	0	运算
	3	0	运算



N指数字键

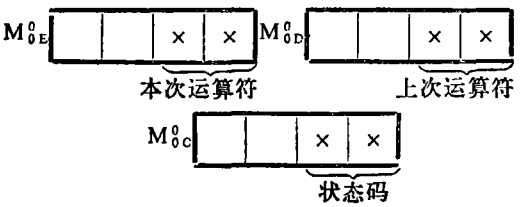
图5-12 计算器状态流向图

它一共有4个状态。"0态"为系统的初始态，在任何状态下按[=]键都回到"0态"。按任一个数字键后，系统从"0态"进入"1态"。这时可继续按数字键以送完所需的数字，输入操作系统将一直停留在"1态"。按任一个运算键(+、−、×、÷等)后，系统将进入"2态"，同时把第一个操作数送至另一个操作数缓冲

区保存起来，以准备接收第2个操作数。当然此时仍然显示第一个操作数。在第"2态"时如再按运算键，则状态不变，但是运算符标志被改变，这就是上述"纠错"功能的实现。再按数字键将进入"3套"。在该状态下按数字键状不变，和"1态"时相同。如按功能键则会启动相应的运算，并显示运算结果。但运算键和[=]键的实际操作不完全一样，前者将回到第"2态"，同时把运算结果送缓冲区保存，因此可立即作新的运算。而后者运算后回到初始态。

根据状态流向图可整理出相应的状态表（见表5-2）。在状态表中第一栏是按键名称，功能相同的键（如数字键、运算键）可并在同一行。第二栏是该键按下时系统所处的状态。第三栏说明按键处理后系统应转移到哪个状态。第四栏则列出了该种情况下应进行的处理和运算。

有了状态表就能编出分析程序。首先需设立状态标志单元以存储状态。由于一共只有4个状态，2bit就够了，假定用$M^0_{C(1,0)}$这2位。运算功能键按下时并不立即起作用，而要等下一次按运算键或等号键时才进行运算，所以运算符也要存储起来。特别是在作连续运算时，需存储2次运算符。现用M^0_E和M^0_D这二个单元，分别存本次的运算符和前次的运算符。（由于它们都只占用2位，所以用一个单元也可以，但程序要麻烦一些。）按键的编码则存在M^0_F中。



M^0_F 值	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
对应按键	0	1	2	3	4	5	6	7	8	9	+	−	×	÷	=

图5-13 键盘分析程序状态标志及按键编码

键盘分析程序状态标志及按键编码见图5-13，计算器键盘分析流程图见图5-14。

分析程序首先将按键编码取至A中，用ADX指令判别出是哪一类按键。根据按键内容及当前状态（对$M^0_{C(1,0)}$用位操作指令SKMP来判别）分别进行修改状态、送数、运算等相应的操作。为了提高效率，相同的操作可合并在一起，但先后次序要注意。送数、运算等操作应放在最后，因为它们是一个独立而复杂的程序模块，作为子程序调用往往不太合适，应从分析程序末尾转去执行。

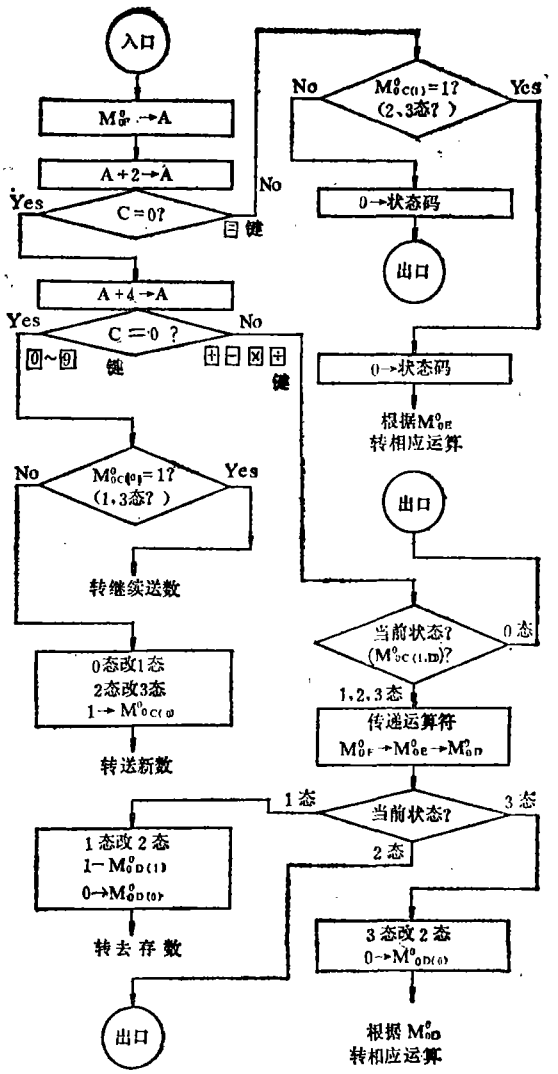


图5-14 计算器键盘分析流程图

存运算符和改变运算符的工作由同一般程序"传递运算符"来做。运算符的编码分别为A(1010)、B(1011)、C(1100)、D(1101)，其末2位各不相同，可用作为判别的标志。每次按过有效的运算符后，把上次的运算符从M^0_E中送至M^0_D，再把本次运算符从M^0_F中送至M^0_E。连续运算时从M^0_D中取出上次运算符判断运算的种类进行相应运算。但在按等号键时，由于只按了一次运算键，所以是从M^0_E中取出判别。

最后说明一点：这种状态分析的办法不但在键盘分析中是一种有效的工具，而且还可引伸延用于模拟信号的分析处理工作。例如，要判断出模拟信号曲线中的谷点或峰点，就必需对一连串前后相关的采样值进行状态分析。对此状态表法能发挥强有力的作用。笔者就曾用此法介决了光密度扫描仪的曲线分析工作、效果颇佳。（待续）