第 12 章 智能仪器设计

使用单片机来制作智能仪器是单片机的一种重要用途。本章通过一台智能仪器的设计，学习单片机对浮点数的处理方法，学习使用 LED 数码管显示小数的方法。

12．1 设计任务分析

在智能仪器的设计中，常有这样一种要求：使用按键面板来设置一个系数，通过 A/D 转换或计数等方式得到的一个数值乘以该系数得到一个值，将这个值在 LED 数码管上显示出来。

在这样的设计要求中，所设置的系数往往是一个小数，这样做完乘法后，所得到的结果也会是一个小数，如果输入的数变动范围比较大，那么所得到的结果也会有比较大的变动范围。设有如下的要求：

系数范围：0.001~9.999

计数输入：1~4294967295

显示器：6 位 LED 数码管

要求将计数值乘以系数后用数码管显示出来，并且显示尽可能多的小数位数，通过计算不难得到，所要显示的值的范围达在系数为 0.001 时最小，当计数值为 0.001 时，最小计数值为 0.001，而计数值很大或者系数较大时，显示值最多可能达到 999999。数的范围变化如此之大，必须采用浮点数才能够满足要求，为此，下面首先对浮点数的相关知识作一个介绍，然后编程来实现这一要求。

12．2 浮点数

通常人们书写时常用诸如：

17.3＝1.73E1=1.73× 10^{1}

-2345.67=-2.34567E3=-2.34567× 10^{3}

-0.00012345=-1.2345E-4=-1.2345× 10^{-4}

这种方式其实质就是浮点记数法，和定点数相比，浮点数在运算、存储等方面要复杂得多，下面学习有关浮点数基本知识。

12．2．1 浮点数的基本知识

浮点记数方法将一个数用三部分来表示，第一部分为数符，表示正、负数；第二部分表示这个数的有效数字，与相对精度有关；第三部分表示这个数的数量级。在计算机中，数符通常用字节中的一位来表示，规定正数用 0 表示，负数用 1 表示；有效数字部分又称为尾数，用若干个字节的纯小数表示；数量级部分又称阶码，它本身也有正负，用二进制补码整数表示。

同样的数值可以有多种浮点数表达方式，比如 17.3 可以表达为 17.3×10^{1}，0.173×10^{2} 或者 0.0173×10^{3}。因为这种多样性，所以有必要对其加以规范化以达到统一表达的目标。对

于 10 进制浮点数来说，规范的浮点数表达方式具有如下形式：

$\pm d.dd...d \times 10^e$

其中 d.dd...d 即尾数，10 为基数，e 为指数。尾数中数字的每个数字 d 介于 0 和基数之间，包括 0。小数点左侧的数字不为 0。

12．2．2　C51 中的浮点数

前面小节讨论的是十进制浮点数的表示方法，单片机内部的数值表达是基于二进制的，二进制数同样可以有小数点。

1．二进制浮点数的表示方法

二进制浮点数的表法方法类似于十进制，只是基 等于 2，而每个数字 d 只能在 0 和 1 之间取值。如：二进制数 10101.101 相当于

$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$。

写成通用表达式为：

$\pm d.dd...d \times 2^e$

其中**d**在**0**和**1**之间取值，**e**通过一个字节来表示，有符号数，取值范围为**-127~+128**。尾数是用**3**字节的（**24bit**）来表示，由于每个尾数的最高位（即小数点前的**1**位）总是**1**，所以不必存储，第**1**位作为符号位。这样，在计算机内部数据表示的通用格式为：

SEEE EEEE EMMM MMMM MMMM MMMM MMMM MMMM

在这里：

S：尾数的符号位，**0**表示正，**1**表示负。

E：指数的值，实际存储的是相对于**127**的偏移量

M：24位的尾数 （实际存储**23**位）。

0是一个特殊的值，其指数和尾数都是0。

以下以举几个例子来说明。

（1）-12.5

这个数在C51中的存储方式为**0xC1480000**，为何会是这样的一个数呢？这个数用二进制来表示就是：

11000001 01001000 00000000 00000000B

其中第**1**位是尾数的符号位，这里是**1**，说明这是一个负数。

指数的值是其后的**8**位即第**1**个字节的低**7**位和第**2**个字节的第**1**位，组合起来就是：**10000010B**，相当于十进制的**130**。

尾数是其后的**23**位，即 **10010000000000000000000B**，实际尾数是用**24**位来表示的，它的最高位总是**1**，因此，这个数实际上是这样的：1.10010000000000000000000B，注意其中的小数点位置。

接着，要根据指数值来调整尾数的小数点了，如果指数大于**127**，那么小数点将右移，而如果这个值小于**127**，那么小数点将左移，移动的是指数相对于**127**的偏移量，刚才我们已计算出指数值是**130**，那么偏移量是**130**－**127**＝**3**。这样调整后的数变为：

1100.10000000000000000000B

这个就是最终的数。计算一下：

$2^3+2^2+2^{-1}=12.5$

加上最前面的符号为表示负，因此结果就是**-12.5**

（2）0.125

这个数在 C51 中的存储方式是 0x3E000000，写成二进制数就是：

00111110 00000000 00000000 00000000B

其中第 1 位是尾数的符号位，这里是 0，说明这是一个正数。

指数是其后的 8 位，即第一个字节的低 7 位和第二个字节的高一位即：

01111100B，就是十进制数的 124。

而尾数是其后的 23 位，即 0000000 00000000 00000000B，但实际的尾数是 24 位，第一位总是 1，所以真实的尾数是：1.0000000 00000000 00000000B

接着根据指数来调整小数点的位置，由于 124 小于 127，因此，小数点要左移，左移的位数是 127－124＝3 位，这样，最终得到的数是：

0.00 10000000 00000000 00000000B

计算一下：$1\times2^{-3}=0.125$

2．关于浮点数精度的讨论

（1）基于习惯性思维，人们会认为，使用了小数点，可以使得所处理的数据获得很高的精度。其实这是不对的，使用浮点数原因是它能够表达的数的范围很大，而并非精度很高。对于单精度数，由于只有 24 位二进制位的尾数（其中一位隐藏），所以可以表达的最大尾数为 2^{24} - 1 = 16，777，215。也就是说，单精度的浮点数可以表达的十进制数值中，真正有效的数字不高于 8 位。

（2）十进制小数转换为二进制

我们知道，十进制小数转化为二进制小数方法是不断地乘 2 并取整数部分，但是这种变换本身就会产生误差。例如一个十进制小数 0.337 转化为二进制小数

0.337×2＝0.674　0

0.674×2＝1.348　1

0.348×2＝0.696　0

0.696×2＝1.392　1

0.392×2＝0.784　0

0.784×2＝1.568　1

.....

这种计算有可能是无穷无尽的，但实际运算中不能无限地算下去，只能取有限的二进制位数。如果计算到这里作为结束，那么就认为 0.337 的二进制小数是 0.010101B。下面再将这个二进制转化为十进制：

$1\times2^{-2}+1\times2^{-4}+1\times2^{-6}=0.25+0.0625+0.015625=0.328125$

这个数与 0.337 相差了：0.008875。仔细分析，不难发现，只有尾数是 5 且位数有限的十进制小数才有可能做到精确转换。可见，对于绝大多数的十进制小数来说，转化为二进制小数本身就有误差。综合第 1 条所讨论的精度情况，对于单精度浮点数来说，其有效的精度一般不会超过 7 位。

（3）Keil C51 中没有双精度浮点数

在 keil C51 中可以使用 double 来定义双精度符号数，但 C51 处理 double 的方法和处理 float 型数据的方法是相同的，因此实际运算时并不会比 float 型数据得到更高的精度。

例：
```
double x=10.002;
double y;
y=x*12.3456;
```

得到的结果是 y=123.4807（理论上 y=123.4806912）。

由此可知，浮点数远比整数复杂得多，这提醒我们，在进行浮点运算中，特别要小心。例如，在整数中判断两个变量 a、b 是否相等，可以这么样来写：

```
int a，b；
.....对 a，b 进行计算等操作
if（a==b）
.....
else
……
```

但是，对于浮点数这样来写可能就要出问题。

例：

```
float a，b；
.....经过计算，理论上 a 和 b 都应等 0.0123
if（a==b）
.....
```

这样写很有可能会出问题，因为 a 和 b 也许永远也无法做到真正的完全相等，该条件永完无法实现，但如果写成：

```
if（a>=b+p）
    ……
```

其中 p 是一个非常小的数，如 0.00001 等，就比较合理，不容易出错。

3. 浮点数中的特殊值

除了正常的浮点数以外，还有一些浮点数错误值。这些错误值作为 IEEE 的标准而存在，在自己编程时，可以使用函数检查，以确认是否有这种错误产生。IEEE 中定义的浮点错误值为：

NaN **0xFFFFFFF** 不是一个数
+INF **0x7F80000** 正无穷大 （正数溢出）
-INF **0xFF80000** 负无穷大 （负数溢出）

当出现诸如对一个负数开根、零除以零等操作时，将会产生**NaN**错误。如果有两个很大的数相乘，而其结果已超过了**C51**中浮点数所能表达的范围，此时产生一个正数溢出错误。这可以让程序员有可以通过编程来发现这样的错误，从而做出适当的处理，因此产生正数溢出比简单地让这个数等于浮点数所能表达的最大数更有价值。对于负数来说，也会有这样的一个溢出问题，所以在**C51**中分别定义了正无穷大和负无穷大两个特殊值。

C51提供了**_chkfloat_**函数以便快速地检查是否有浮点数错误。

```
#include <intrins.h>
unsigned char _chkfloat_ （float val）;        /*检测错误*/
```

描述： _chkfloat_ 函数用于检测浮点数的状态

返回值：返回的数值表示浮点数的状态

0：标准的浮点数
1：浮点数0
2：+INF （正溢出）
3： -INF （负溢出）
4：NaN （非数值）

例：

```
#include <intrins.h>
```

```
#include <stdio.h>
char _chkfloat_ （float）;
float f1， f2， f3;
void tst_chkfloat （void）
 {
    f1 = f2 * f3;
    switch （_chkfloat_ （f1））
       {
          case 0:
             printf （"result is a number\n"）;  break;
          case 1:
             printf （"result is zero\n"）;  break;
          case 2:
             printf （"result is +INF\n"）;  break;
          case 3:
             printf （"result is -INF\n"）;  break;
          case 4:
             printf （"result is NaN\n"）;  break;
       }
}
```

4.浮点数运算注意事项

虽然浮点数能够表达的数的范围很大，但实际运算中仍要注意一些原则：

- 避免两个极大数相乘
- 避免两个相近数相减
- 避免用极小数做分母

通常这些运算并不会在运算的最终结果中出现，但如果不注意的话，它们可能会在运算的中间步骤中出现，从而造成运算的误差增加或者不能获得正确的运算结果。

例：有3个浮点数，$a=1.3*10^{20}$，$b=2.4*10^{20}$，$c=3.4*10^{15}$

要求a*b/c的值。如果采用下面的方法：

```
float a=1.3e20;
float b=2.4e20;
float c=3.4e15;
float x ； //中间结果
x=a*b;
x=x/c;
```

就不合适，因为a*b的结果大于了浮点数所能表达的范围。结果如图12-1所示。

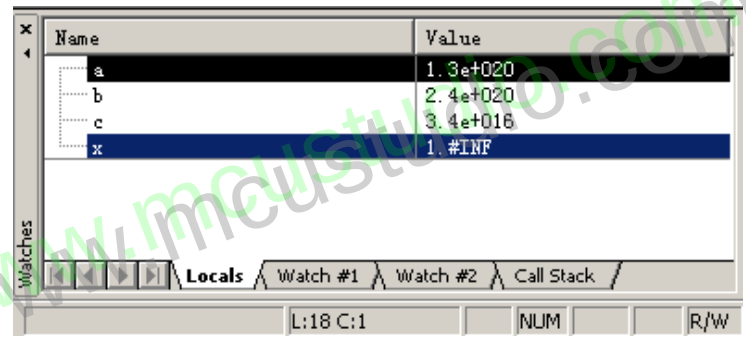

图12-1 运算结果出现INF错误

而采用

```
x=a/c;
x=x*b;
```

就能够正确地进行运算，并获得正确的结果。

12．2．3 浮点数转化为整型数

如果仅需将浮点数的整数部分转化为整型数，则只要直接将 float 型的数据赋给整型变量即可。

```
例：float x=3.4567;
    int a=0;
    ......
    a=（int）x;
```

即可完成整数部分的转化。不要小看这么一行程序，实际运作中，C51 需要调用一个函数来完成这项工作，并且这项工作需要消耗掉五百多个机器周期的时间。这就是通常所说的，单片机程序中加入浮点运算后速度会明显变慢的原因之一。将浮点数转化为整数后，就可以使用传统的方法对整数处理以得到待显示的各位数值了。但是如果需要将小数点后的数值也显示出来，就不能这样做了。

通常，要处理浮点数的小数部分都是在计算的最后，要将数值显示出来，就需要将浮点数转化成为十进制 BCD 码。

一种思路是先将小数扩大一定的倍数，然后取整数部分，剩下的小数部分舍去，在显示的时候，根据扩大的倍数在相应的数码管上点亮小数点，这样就能将小数部分显示出来。

例如某数 12.34508 要在数码管上显示出来，实际的硬件中，数码管的位数总是有限的，假设仪器上用到的数码管是 4 位，那么应该显示 12.34，而其后不管有多少位，都不能被显示出来。这样，只要将这个待显示数乘以 100，然后对其取整，得到一个整数 1234，然后将此数显示出来，在编写显示程序时，在显示倒数第 2 位数码管时，点亮小数点，那么人们实际看到的就是显示 12.34。这种方式请参考第 15 章全数字信号发生器中的例子。

这种方式可以显示小数点，但其局限性也很明显，即能够显示的数的范围有限，如果要求显示的数据范围很大，如从 0.001 显示到 9999，那么这种方法编程将会变得很复杂，并且效率很低。而当所要显示的位数更多时，这种方法几乎不可实现。这时，需要使用其他方法来实现，详见例 12-1 中的介绍。

12．3 智能仪器设计

设计与制作本台仪器的目的是为了演示单片机中浮点数的用法。

【例 12-1】设计一个智能仪器，能使用键盘设置系数，系数的值从 0.001~9.999 变化，显示值等于系数×计数值，其值可以在 0.001~999999 变化，采用 6 位数码管来显示数据。

这个例子采用多模块编程，如图 12-2 所示。

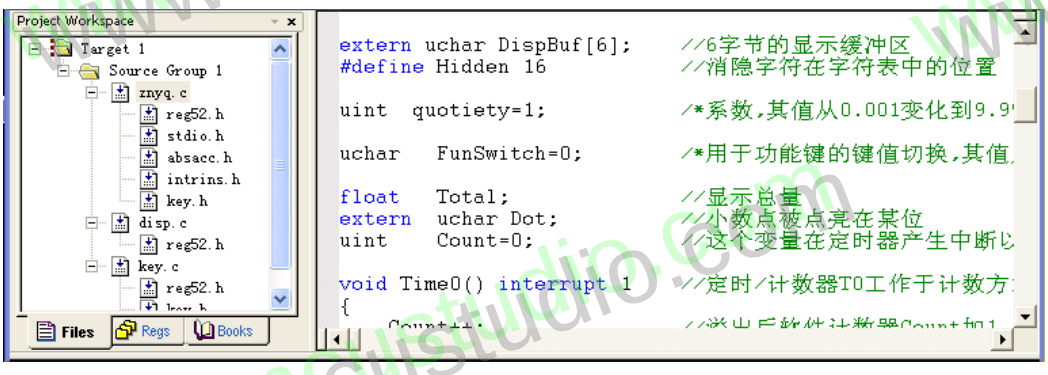

图 12-2 智能仪器多模块编程

从图中可以看到，一共有 3 个程序：znyq.c、disp.c、key.c，分别为主程序模块、显示程序模块和按键处理模块。下面对各段程序分别加以说明。

主程序如下：

```
#include <reg52.h>
#include <stdio.h>
#include<absacc.h>
#include <intrins.h>
#include <key.h>

/*有关全局变量的定义*/
extern uchar DispBuf[6];      //6 字节的显示缓冲区，这个变量在 disp.c 中定义
#define Hidden 16             //消隐字符在字符表中的位置
uint   quotiety=1;            /*系数，其值从 0.001 变化到 9.999*/
uchar    FunSwitch=0;
/*用于功能键的键值切换，其值为 0，1，分别表示显示总量和显示系数*/

float Total;                  //显示总量
extern   uchar Dot;           //小数点被点亮在某位，这个变量在 disp.c 中定义
uint  Count=0;
//软件计数器，这个变量在定时器产生中断以后加 1，与定时器一起构成 32 位的计数器
/*************************************************************************
函数功能：定时/计数器 T0 中断程序
参      数：无
返      回：无
备      注：定时/计数器 T0 工作于计数方式
*************************************************************************/
```

```
void Time0（） interrupt 1        //
{    Count++;                  //溢出后软件计数器 Count 加 1
}

void main（）
{
        unsigned long lTemp=0;
        uint iTmp=0;
        uchar    cTemp=0;
        uchar    cBuf[12];
        uchar    i;
        P2=0xff;
        P0=0x1f;                  //数码管全部熄灭

        TH1=-（2500/256）;
        TL1=-（2500%256）;  //12M 晶振时为 4ms 一次中断
        TMOD=0x15;              //T/C1 工作于定时器方式 1，T0 工作于计数方式 1
        ET1=1;
        EA=1;
        ET0=1;
        TR0=1;
        TR1=1;
/*以上初始化部份*/
        for（;;）
        {
          if（cTemp=Key（））      //调用键盘扫描程序，获得返回值
              KeyValue（cTemp）;  //如有返回值不为零，说明有键按下，作键值处理
          switch（FunSwitch）
          {
              case 0:                   //显示总量
              {
                  lTemp=Count;      //取得软件计数值
                  iTmp=TH0;         //读取 T0 计数器的高 8 位
                  cTemp=TL0;        //读取 T0 计数器的低 8 位
                  if（cTemp==0）    //如果读到的是 0
                      iTmp=TH0;     //可能在读此数时 TH0 已发生变化，因此重读一次
                  lTemp*=65536l;    //软件计数器的值扩大 65536 倍
                  lTemp+=iTmp*256+cTemp;   //加上 T0 中的计数值
                  Total=（float）lTemp;   //准备计算总量，先将计数值转为浮点数
                  Total*=quotiety;        //计数值乘以系数
                  Total/=1000;
//结果除以 1000，因为 quotiety 的值在 1~9999 之间，但表示 0.001~9.999
                  sprintf（cBuf,  "%.5lf",  Total）;    //将浮点数转化为字符数组
                  for（i=0；i<12；i++）
```

```
//这段代码是查找浮点数组的末尾位置，即包括整数和小数部分的总长度值
                    {    cTemp=cBuf[i];
                         if（cTemp==0）
                         break;
                    }
                    Dot=i-6;              //小数点应该显示在最末一位整数位上
                    for（i=0；i<6；i++）  //准备将字符数组送去显示
                         DispBuf[i]=cBuf[i]-0x30；//将 ASCII 字符转批为数字
                    break;
                } /*case 0 end*/
                case  1：          //显示系数
                {
                    Dot=2；        //在显示系数时，小数点固定显示在第 3 位数码管上
                    DispBuf[0]=10；
//第一位数码管显示字符 A，表示正在进行系数的设置
                    DispBuf[1]=Hidden；  //第二位数码管不显示
                    iTmp=quotiety；
//将系数变量 quotiety 的值送给无符号整型变量 iTmp
                    DispBuf[5]=iTmp%10；
//iTmp 的值除 10 取余即得到最末位的数值
                    iTmp/=10；
//将 iTmp 除以 10，由于其为整型变量，因此，最后一位被舍去
                    DispBuf[4]=iTmp%10；//再次除 10 取余，得到次低位的值
                    iTmp/=10；                //iTmp 除以 10，再次舍去最低位
                    DispBuf[3]=iTmp%10；//iTmp 除 10 取余得末位数
                    DispBuf[2]=iTmp/10；  //iTmp 除 10 得前面的数
                    break；
                } /*case end*/
            }/*switch end */
        }
}
```

程序分析：这是整个程序中的主模块，其流程图如图 12-3 所示。

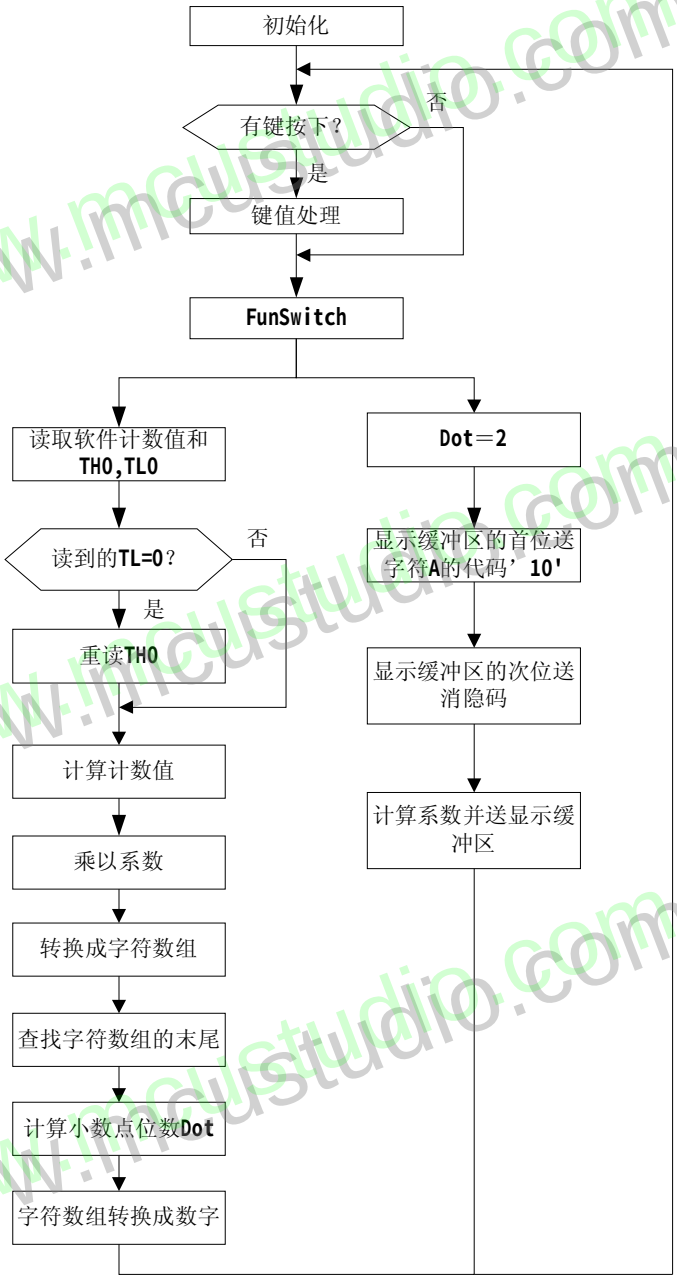

图 12-3 主程序流程图

系统初始化以后，即调用键盘程序，如果有键按下，则对按键进行处理并获得键值。随后程序有两个分支，即显示总量（系数*计数值）或者显示正在设置的系数。通过变量 FunSwitch 值的变化来实现这两个分支，当 FunSwitch=0 时显示总量，而 FunSwitch=1 时显示系数值。系数一共有 4 位，其值从 0.001~9.999，因此其小数点可以固定显示在第 3 位数码管上，程序直接给变量 Dot 赋值 2 来实现这一点。而显示总量时，小数点的位置在不断变化，程序使用了 sprintf 函数来对浮点数进行格式化，下面首先介绍一下 sprintf 的有关知识。

sprintf 函数，其原型为：

```
int sprintf （
char *buffer,          /* 存储缓冲区指针*/
const char *fmtstr     /* 格式化字符串*/
```

[， argument]…）； /* 附加参数*/

该函数使用指定的格式化方式格式化指定对象，并将格式化后的对象存储在存储缓冲区中指针所指的区域中。其中格式化的字符串的含义如下：

格式字符由字符、转义序列、格式说明组成，字符和转义字符将被原样复制到输出结果中，而格式特性字符总是由%作为引导，需要在该符号后面加上若干参数，格式字符串是从左往右读，第一个格式特性字符与第一个输出量匹配，第二个格式特性字符与第二个输出量，以此类推，如果输出量的数量比格式特性字符多，多出来的输出量将被忽略。如果附加参数比格式字符少，将产生不可预知的结果，格式字符具有下列所示格式：

% 标志 宽度 .精度 {b|B|l|L} 类型

格式字符中的每个区域可能是一个单个的字符或具有指定含义的数字。

区域“类型”是一个单个的字符用于说明参数被当成是字符、字串、数字或指针，详细说明见表 **10-1**。

表 **12-1** 格式化字符的含义

字符	参数类型	输出格式
D	Int	有符号十进制数
U	unsigned int	无符号十进制数
O	unsigned int	无符号八进制数
X	unsigned int	无符号十六进制数
F	Float	浮点数，使用格式[-]ddd.dddd
E	Float	浮点数，使用格式[-]d.dddde[-]dd
E	Float	浮点数，使用格式[-]d.ddddE[-]dd
g	Float	浮点数，既可以使用 f 格式也可以使用 e 格式，取决于给点值及精度使用哪种格式更紧凑
G	Float	与 g 格式相同
C	Char	有符号字符型
S	generic *	具有 null 字符的字符串
P	generic *	指针，使用格式 t：aaaa，这里 t 是存储类型（c：代码空间，I：内部 RAM，x：扩展的外部 RAM 空间，p：扩展的外部一页 ROM）。aaaa 是一个十六进制数的地址

直接给出字母 b、B 和 l、L 可以比类型指定符号优先指定。

看一看程序中的写法：

sprintf（cBuf， "%.5lf"， Total）；

其含义为：将 **Total** 这个变量按照 **5** 位小数的要求格式化，整数部分未作要求，有多少位就输出多少位，将结果写入 **cBuf** 指针所指的内存空间中。

由于设计中的采用了 **6** 位数码管，因此实际工作时必须保证数据的整数位不多于 **6** 位。或者换一种说法：当设计者对所需显示的数据进行分析，确认其整数部分不超过 **6** 位时，设计中采用了 **6** 位数码管。

6 位数码管在显示小数时，至少第 **1** 位必须是整数，因此，最多可能是显示 **5** 位小数。当然，本例中系数只有 **4** 位小数，而计数值又是整数，所以实际上最多只会显示 **4** 位小数，不过为了更具有一般性，所以本例中还是作为 **5** 位小数来处理。这样，至多 **6** 位整数加上至多 **5** 位小数，可能会有多至 **11** 位数据储存在 **cBuf** 缓冲区中。换言之，在定义 **cBuf** 时其长度不得小于 **11**。

由于小数位数不能事先确定，因此在将待显示的数送入 **cBuf** 后，需要判断小数点的位

数，并且将此位数赋给一个变量 **Dot**，以便利用这个变量在 **disp.C** 中点亮相应位的小数点。

判断小数点的方法是查找浮点数组的末尾位置，即找到包括整数和小数部分的总长度值。找到之后，减去 **5** 位小数即可得到整数部分的长度。查找数组末尾位置的方法是找到数字“**0**”，找到 **0**，就说明找到了浮点数组的末尾。那么如果所要显示的数据中有“**0**”会不会造成误判断呢？不用担心，要显示的数字在 **cBuf** 中是以 **ASCII** 码来保存的，数字“**0**”被保存为 **0x30**。

例如：**12.5** 这个数，在浮点数组中的形式如图 **12-4** 所示：

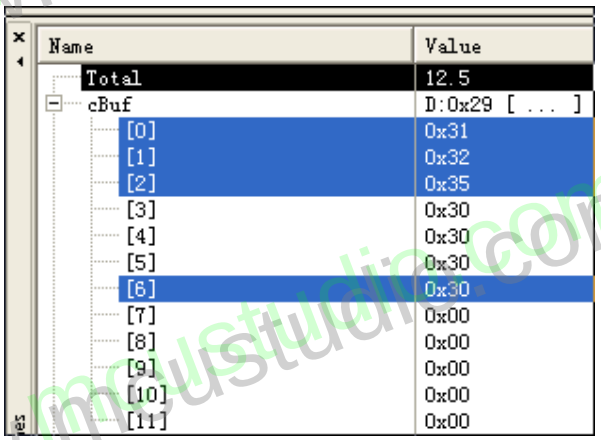

图 **12-4** 浮点数在内存中的存放

可见，虽然看起来这个数只有 **1** 位小数，但 **sprintf** 函数会当成是 **12.50000** 处理，即补足 **5** 位小数。

小数点应该在第 **2** 位数码管上，确定小数点的代码如下：

```
for（i=0；i<12；i++）
{    cTemp=cBuf[i]；
        if（cTemp==0）
        break；
}
Dot=i-6；              //i 的值表示数据的总长度值
```

这样，Dot 表示了要显示的小数点的位置，对于这个例子来说，Dot=1，在 Disp.c 文件中可以看到，这个数将与显示程序配合，在第 2 个数码管上显示出小数点来。

如果要将此数送去数码管显示，则要将待显示的数由 ASCII 码转为数值，方法是将值减去 0x30，如果是送去 LCD 显示，由于其直接显示 ASCII 码，因此就不需要再作变换了。

由于数码管只有 6 位，因此，只能显示前 6 位数字，将 cBuf 的前 6 位经送到显示缓冲区，程序如下：

```
for（i=0；i<6；i++）                  //准备将字符数组送去显示
    DispBuf[i]=cBuf[i]-0x30；     //将 ASCII 字符转批为数字
```

至此，浮点数转化为 BCD 码的工作完成，在显示缓冲区里已是待显示的数据了，下面的 disp.c 程序要完成的工作是将显示缓冲区中的内容显示出来，并且根据 Dot 的值在相应的数码管上点亮小数点。

```
Disp.c
#include "reg52.h"
typedef unsigned char uchar；
typedef unsigned int    uint；
```

```
    uchar code BitTab[]={0x7F，0xBF，0xDF，0xEF，0xF7，0xFB}；
    uchar code DispTab[]={0xC0，0xF9，0xA4，0xB0，0x99，0x92，0x82，0xF8，0x80，
0x90，0x88，0x83，0xC6，0xA1，0x86，0x8E，0xFF}；

    uchar DispBuf[6]；          //6 字节的显示缓冲区
    uchar Dot；                 //小数点控制，值在 1~5 之间，0 表示没有小数点
    extern uchar FunSwitch；
    extern uchar ShiftKey；
    /*************************************************************************
    函数功能：定时器 T1 中断处理程序，
    参    数：无
    返    回：无
    备    注：定时/计数器 T1 用于显示
    *************************************************************************/
    void Timer1（） interrupt 3
    {   uchar tmp；
        bit       sMark；
        bit       BitLamp；     //控制显示位闪烁的标志
        static uchar tCount；   //计数器，显示程序通过它得知现正显示哪个数码管
        static uchar sCount；   //秒计数器
        TH1=（65536-2500）/256；
        TL1=（65536-2500）%256；//定时时间为 2500 个周期
        sCount++；
        if（sCount>=200）           //2.5*200=500ms
        {   sCount=0；
            sMark=~sMark；
        }
        if（FunSwitch==0）
            BitLamp=1；     //如果是正在显示计数值，则没有显示闪烁的问题
        Else
        {
            if（sMark&&（tCount==ShiftKey））//如果秒标志为 1 并且是当前操作位
                BitLamp=0；             //显示标志为零
            else
                BitLamp=1；             //否则显示标志为 1
        }
        tmp=BitTab[tCount]；            //取位值
        P2=P2|0xfc；                    //P2 与 11111100B 相或
        P2=P2&tmp；                     //P2 与取出的位值相与
        tmp=DispBuf[tCount]；           //取出待显示的数
        tmp=DispTab[tmp]；              //取字形码
        if（（tCount==Dot）&&（Dot!=5））
    /*如果 Dot 不等于 5（Dot 等于 5 说明小数在最末位，此时不应显示小数点），且计数值
等于小数点控制位数*/
```

```
        tmp&=0x7f;                    //点亮小数点
    if（BitLamp）                        //如果 BitLamp 为 1
        P0=tmp;                       //将字形码送出到 P0 口
    Else                              //否则
        P0=0xff;                      //将 0xff 送到 P0 口，停止所有显示
    tCount++;
    if（tCount==6）
        tCount=0;
}
```

程序分析：这是一个定时中断服务程序，进入定时中断后，重置 TH1 和 TL1 的定时初值，由于所用的晶振为 12M，因此 TH1=（65536-2500）/256；TL1=（65536-2500）%256；设置的定时时间为 2500us，即 2.5ms，当然，实际的时间要略长一些，不过，本程序并不涉及精确的定时要求，因此，程序中未作修正。sCount 是一个计数器，每计到 200，就让一个标志 sMark 取反，同时让 sCount 回零。这样的效果就是每 2.5*200＝500ms 使得 sMark 取反一次。

接下来要区分显示总量与设置系数时不同的特性，因为在设置系数时，当前操作位要闪烁显示，以指示操作者，因此，这里判断 FunSwitch 是否为 0？如果是 0，说明正在显示总量，不需考虑显示闪烁问题。否则判断 sMark 是否等于 1，并且当前显示的位是否是当前正在操作的位。当前操作位通过变量 ShiftKey 来实现。

```
if（sMark&&（tCount==ShiftKey））  //如果秒标志为 1 并且是当前操作位
```

这里用到的 FunSwitch 和 ShiftKey 都是全局变量，前者在 znyq.c 文件中定义，而后者在 key.c 文件中定义，因此在 disp.c 文件中这两个变量定义是这样的：

```
extern uchar FunSwitch;
extern uchar ShiftKey;
```

设置好 BitLamp 标志后，接下来就是一些常规操作，根据显示计数器来取字形码，取出字形码后，首先要实现小数点的点亮。根据 Dot 的值及显示计数器的值来决定是否要点亮小数点，如果这两者相等，说明需要点亮小数点，则用 0x7f（0111 1111B）与字形码相与，不论当前正在显示的字形码是什么，总是让其最高位为 0。最高位所对应的正是小数点 h 笔段，将这样的数字送到 P0 口，将会点亮小数点。

显示完小数点以后，接下来要实现当前显示值闪烁提示的工作。程序中根据 BitLamp 的值来决定是否将字形码送到 P0 口，如果该位是 1，那么将字形码送到 P0 口，以显示相应的字符；如果该位是 0，则将 0xff 送到 P0 口，使得该位字符不显示（消隐），这样，就实现了当前操作位每 0.5s 亮/灭一次。有关该段程序的流程如图 12-4 所示。

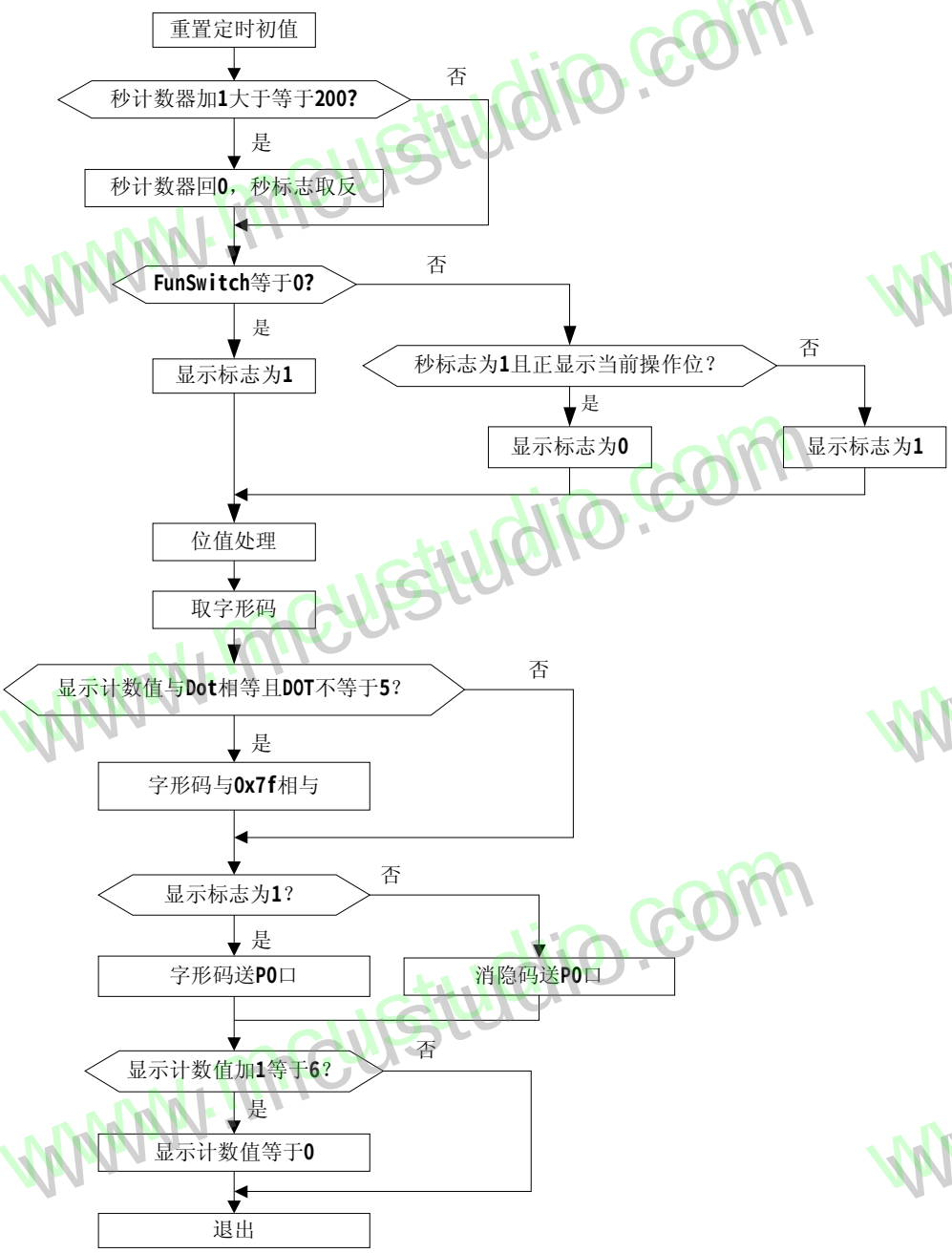

图 12-4 定时中断流程图

第 3 个文件是 Key.c，用以实现按键操作。

```
#include  "reg52.h"
#include  "key.h"
extern uint    quotiety;     //系数，在 znyq 中定义
extern uchar DispBuf[6];  //6 字节的显示缓冲区
uchar ShiftKey=0;       //控制字，控制哪位是当前操作位，用以键盘和显示程序中传递
extern uchar FunSwitch;
//用于功能键的键值切换，其值为 0，1，分别表示显示总量和显示系数
```

```
/*********************************************************************
函数功能：延时程序
参　　数：延时时间，毫秒数
返　　回：无
备　　注：无
*********************************************************************/
void mDelay（uchar j）
{   uchar i=0;
    for（; j>0; j--）
    {   for（i=0; i<125; i++）
        {; }
    }
}
/*********************************************************************
函数功能：键值处理程序
参　　数：键值
返　　回：无
备　　注：键值处理过程描述如下：
1. 切换键，FunSwitch 的值，在 0-1 之间循环
2. shift 键，ShiftKey 的值在 2-5 之间循环
3. 加 1 键，在相应位（显示缓冲区）加 1（控制在 0-9 之间）
4. 减 1 键，在相应位（显示缓冲区）减 1（控制在 0-9 之间）
*********************************************************************/
void KeyValue（uchar KeyVal）
{   bit  kProc=0;
//如果经过了加 1 或减 1 处理，则该值变为 1，在计算了 quotiety 后，该值变为 0
    switch （KeyVal）
    {
        case 0xfb:                  // P3.2 引脚所接按键用于切换功能
        {
            FunSwitch++;
            if（FunSwitch==2）          //FunSwitch 值在 0 和 1 之间变化
                FunSwitch=0;
            ShiftKey=2;             //保证总是从第一位开始操作
            break;
        }
        case 0xf7:                      //P3.3 引脚所接为移位键
        {   if（FunSwitch==0）      //如果未按下设置键，则该键封锁，以下三键相同
                break;
            ShiftKey++;
            if（ShiftKey>=6）
                ShiftKey=2;         //保证 ShiftKey 键值在 2~5 之间变化
            break;
        }
```

```
            case 0xef:                          //P3.4 引脚所接为加 1 键
            {
                if（FunSwitch==0）              //不是在设置系数，直接返回
                    break;
                if（++（*（DispBuf+ShiftKey））==10）
//对显示缓冲区加 1，然后判断是否到 10，如果到了 10，回 0
                    *（DispBuf+ShiftKey）=0;
                kProc=1;
                break;
            }
            case 0xdf:                      //P1.5 所接为减 1 键
            {
                if（FunSwitch==0）              //不是在设置系数，直接返回
                    break;
                if（--（*（DispBuf+ShiftKey））==0xff）
//对显示缓冲区减 1，然后判断是否到了 0xff，如果到了 0xff，即回到 9
                    *（DispBuf+ShiftKey）=9;
                kProc=1;
                break;
            }
        }
        if（kProc）
        {   quotiety=DispBuf[2]*1000+DispBuf[3]*100+DispBuf[4]*10+DispBuf[5];
            //计算系数
            kProc=0;
        }
}
/*KeyValue（） 函数结束*/
/*************************************************************************
函数功能：读取键值并返回
参    数：无
返    回：键值
备    注：无
*************************************************************************/
uchar Key（void）                   /*键盘处理*/
{
    uchar Key=0;
    uchar temp=0;               /*暂存键值*/
    Key=P3;                     /*取 P3 口的值*/
    Key=Key|0xc3;
    if（0xff!=Key）             /*有键按下*/
    {   mDelay（10）;               /*延时 10 毫秒*/
        P3=P3|0x3c;
        Key=P3;
```

```
            Key=Key|0xc3;
            if（Key!=0xff）          /*确实有键按下*/
            {
                temp=Key;
                for（;;）          /*等待按键结束*/
                {     Key=P3;
                      Key=Key|0xc3;
                      mDelay（10）;
                      if（Key==0xff）
                      {
                          break;
                      }
                 }
                 return （temp）;
             }
        }
        return  0;              //无键按下
    }
```

程序分析：key.c 文件中主要包括了 Key 函数和 KeyValue 函数，其中第一个函数是取键值，然后返回，这个函数不作详细介绍。KeyValue 函数用来对键值进行处理，这其中有几点需要说明。

（1）防止误操作 在仪器运行时，某些键只有在进入特定状态后才能允许被处理，否则会导致混乱。程序中作了如下处理：

```
case 0xf7：                 //P3.3 引脚所接为移位键
{   if（FunSwitch==0）      //如果未按下设置键，则该键封锁，以下三键相同
        break;
```

即在 FunSwitch 等于 0 也就是在显示总量时，如果按下了移位键、加 1 键、减 1 键都是无效的，不能对其进行处理。

（2）切换到设置系数时，总是从最高位开始设置。这是按键设计中的一个细节问题，如果不注意，会造成使用者的混乱。程序中的处理方法非常简单，如下面的程序所示。

```
ShiftKey=2;                 //保证总是从第一位开始操作
```

即进入设置状态时，将用于移位控制的 ShiftKey 变量值置为 2 即可。

程序实现：本程序可以使用硬件来进行调试，如果没有硬件，也可以使用作者所开发的 dpj.dll 实验仿真板来演示。关于实验仿真板的详细介绍，请参考文献 3，这里不作详细介绍，仅说明如何使用这一文件。

到 http://www.mcustudio.com 下载实验仿真板的压缩文件，解压后，将其中的 dpj.dll 文件复制到 Keil 文件的安装文件夹下。例如，某机的 keil 软件安装在 D 盘 Keil 文件夹下，那么就将该文件复制到 D：\Keil\Bin 文件夹下即可。

建立名为 znyq 的工程文件，将这三个源程序输入，加入工程。设置工程，使用 dpj.dll 实验仿真板作为调试时使用。点击工具栏上的  按钮，打开 Optins for Target 'Target 1'对话框，选中 Debug 选项卡，在 Parameter：文本框中输入-ddpj，如图 12-5 所示。

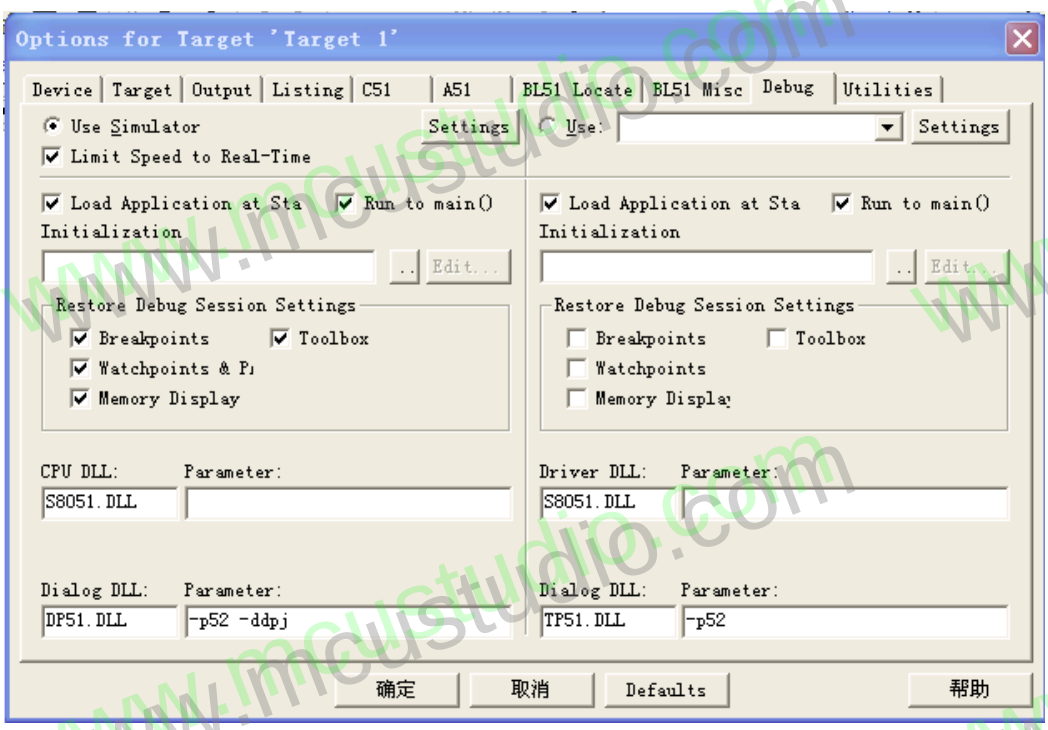

图 12-5 设置工程

编译、链接无误后按 Ctrl+F5 运行程序，单击菜单 Peripherals à 单片机实验仿真板，打开单片机实验仿真板，即可在该板上练习设置的方法。

如图 12-6 所示是设置系数时的状态。

图 12-6 设置系数

系数设置完成后，单击 P3.2 按钮回到计数状态，然后点击 P3.4 实现计数功能，此时所计的数据将会乘以系数，并显示出来，如图 12-7 所示。

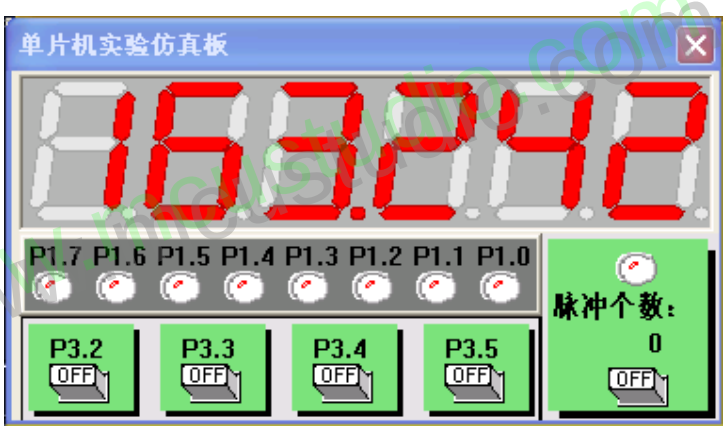

图 12-7 显示数值

思考与实践

在【例 12-1】的基础上增加系数 B 和 C，其中 B 的系数为 0.01~99.99，而系数 C 的值为 0 或者 1，根据系数 C 来决定最终显示的结果是使用系数 A 还是系数 B 来乘以计数值。试编程实现。