《从业将近十年！手把手教你单片机程序框架》

前言

本书是小武根据鸿哥在网上分享的连载-《从业将近十年！手把手教你单片机程序框架》整理而成，版权归鸿哥所有。此学习资料配合朱兆祺51单片机学习效果更佳，另外欢迎加入朱兆祺51单片机QQ群 110291944，共享交流互助学习。

本书乃鸿哥毕生精华所著，独一无二，没有任何一位单片机工程师愿意拿自己近十年的宝贵经验分享给大家，而鸿哥却做到了。他开启的不仅是一个时代，更是一扇千万单片机新人的黄金大门！

本书更多的是编程的思想，框架的构造，以及详细的注释，对未来从事单片机工作有莫大影响。此书可称绝世经典-书中的神奇编程技巧，你绝对没有见过。没有思想的芦苇是可怕，不会编程的电子工程师是可悲的！
！

本书里面的编程思想可谓书写的淋淋尽致，有触类旁通，举一反三之功效，多品味，多滋味！

另本书简洁明了，开门见山直奔主题，让鸿哥带你畅游单片机世界，领略另一个你不曾来过的仙境！

-鸿哥为单片机而生。

目录

第三十五节：带数码管显示的象棋比赛专用计时器。
第三十六节：带数码管显示的加法简易计算器。
第三十七节：数码管作为仪表盘显示跑马灯的方向，速度和运行状态。
第三十八节：判断数据尾来接收一串数据的串口通用程序框架。
第三十九节：判断数据头来接收一串数据的串口通用程序框架。
第四十节：常用的自定义串口通讯协议。
第四十一节：在串口接收中断里即时解析数据头的特殊程序框架。
第四十二节：通过串口用delay延时方式发送一串数据。
第四十三节：通过串口用计数延时方式发送一串数据。
第四十四节：从机的串口收发综合程序框架
第四十五节：主机的串口收发综合程序框架
第四十六节：利用AT24C02进行掉电后的数据保存。
第四十七节：操作AT24C02时，利用“一气呵成的定时器延时”改善数码管的闪烁现象。
第四十八节：利用DS1302做一个实时时钟　。
第四十九节：利用DS18B20做一个温控器　。
第五十节：利用ADC0832采集电压信号，用平均法和区间法进行软件滤波处理。
第五十一节：利用ADC0832采集电压信号，用连续N次一致性的方法进行滤波处理。
第五十二节：程序后续升级修改的利器，return语句鲜为人知的用法。
第五十三节：指针的第一大好处，让一个函数可以封装多个相当于return语句返回的参数。
第五十四节：指针的第二大好处，指针作为数组在函数中的输入接口。
第五十五节：指针的第三大好处，指针作为数组在函数中的输出接口。
第五十六节：指针的第四大好处，指针作为数组在函数中的输入输出接口。
第五十七节：为指针加上紧箍咒const，避免意外修改了只做输入接口的数据。
第五十八节：指针的第五大好处，指针在众多数组中的中转站作用。
第五十九节：串口程序第40,44,45节中存在一个bug，特此紧急公告。
第六十节：用关中断和互斥量来保护多线程共享的全局变量。
第六十一节：组合BCD码，非组合BCD码，以及数值三者之间的相互转换和关系。
第六十二节：大数据的加法运算。
第六十三节：大数据的减法运算。
第六十四节：大数据的乘法运算。
第六十五节：大数据的除法运算。
第六十六节：单片机外部中断的基础。
第六十七节：利用外部中断实现模拟串口数据的收发。
第六十八节：单片机C语言的多文件编程技巧。
第六十九节：使用static关键字可以减少全局变量的使用
第七十节：深入讲解液晶屏的构字过程。
第七十一节：液晶屏的字符，16点阵，24点阵和32点阵的显示程序。
第七十二节：在液晶屏中把字体顺时针旋转90度显示的算法程序。
第七十三节：在液晶屏中把字体镜像显示的算法程序。
第七十四节：在液晶屏中让字体可以跨区域无缝对接显示的算法程序。
第七十五节：在12864液晶屏中让字体以1个点阵为单位进行移动显示的算法程序。
第七十六节：如何把一个任意数值的变量显示在液晶屏上。
第七十七节：在1个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。
第七十八节：在多个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。
第七十九节：通过主菜单移动光标来进入子菜单窗口的液晶屏程序。
第八十节：调用液晶屏内部字库来显示汉字或字符的坐标体系和本质。
第八十一节：液晶屏显示串口发送过来的任意汉字和字符。
第八十二节：如何通过调用液晶屏内部字库把一个任意数值的变量显示出来。
第八十三节：矩阵键盘输入任意数字或小数点的液晶屏显示程序。
第八十四节：实时同步把键盘输入的BCD码数组转换成数值的液晶屏显示程序。

第八十五节：实时同步把加减按键输入的数值转换成BCD码数组的液晶屏显示程序。
第八十六节：数字键盘与液晶菜单的综合程序。
第八十七节：郑文显捐赠的工控项目源代码。
第八十八节：电子称连续不断从串口对外发送数据，单片机靠关键字快速截取有效数据串。
第八十九节：用单片机内部定时器做一个时钟。

第一节：吴坚鸿谈初学单片机的误区。

（1）很难记住繁杂的寄存器？寄存器不用死记硬背，鸿哥我行走江湖多年，连一个寄存器都记不住。需要配置寄存器的时候，直接在网上或者书本上参考别人现成的配置程序是上策，查找芯片数据手册是中策，死记硬背寄存器是最最下策。
（2）很难记住繁杂的汇编语言指令？除非是在校学生要应付考试或者少数工作中绕不开汇编，否则学汇编就是浪费时间。鸿哥我行走江湖多年，从来就没有用汇编帮客户做过一个项目。
（3）C语言很难学？你不用学指针，你不用学带形参的函数，你不用学结构体，你不用学宏定义，你不用学文件操作，你也不用死记繁琐的数据类型。你只要会：

5条指令语句switch语句，if else语句，while语句，for语句，=赋值语句。
7个运算符+,-,*,/,|,&,!。
4个逻辑关系符||,&&,!=,==.
3个数据类型unsigned char， unsigned int， unsigned long。
3个进制相互转化，二进制，十六进制，十进制。
1个void函数。
1个一维数组code(或const) unsigned char array[]。
那么世界上任何一种逻辑功能的单片机软件你都能做出来。

鸿哥我当年刚毕业出来工作的时候才知道可以用C语言开发单片机，一开始只用if语句就把项目做出来了，没有用指针，没有用带形参的函数等复杂的功能。再到后来才慢慢开始用C语言其他的高级功能，但是我发现C语言其他的高级功能，本质上都是用我前面列举出来的最基本功能集合而成，只是书写更加简单方便了一点，编译后的机器码都大同小异。所以不会指针等高级功能你不用自卑，恰恰相反，当你会最简单的几个语句，就把这些高级功能的程序都做出来了，你才发现你对底层了解得更加透切，再学那些高级功能轻而易举。当你裸机跑的程序都能够协调得很好的时候，你才发现所谓高深的操作系统也不过如此，只要给你时间和金钱你也可以写个操作系统来玩玩。
（4）很难记住精确时间的计算公式？经常看到时间公式等于晶振，时钟周期，执行指令次数他们之间的乘除关系式。鸿哥我认为这些都是浮云，不用纠结也不用去记，大概了解一下就可以了。不管你对公式掌握得有多精确，你都不可能做出非常精确的时间。想用单片机做一个非常精确的时间这种想法一开始就是错的，不可能的。真想做一个比较精确的时间，应该用外围时钟芯片或者FPGA和CPLD，而不是单片机。
（5）很难记住繁杂的各种通信协议？什么IIC,SPI,232串口通讯，CAN,USB等等。这些都是浮云，你不用记那么多，你只要理解两种通讯方式就够了，那就是串行通讯方式和并行通讯方式。不管世界上有多少种通讯协议，物理世界上只有这两种通讯方式，其他各种名称的通讯协议都基于此两种方式演变而来。
（6）很难写短小精悍的程序？初学者不要纠结于此。做项目开发，程序容量不是刻意追求的目标，程序多一点少一点没关系，现在大容量的单片机品种非常多，容量不会是寸土寸金的事情，我们更加要关注程序的运行效率，可读性和可修改性。
（未完待续，下节更精彩，不要走开哦）

第二节：delay()延时实现LED灯的闪烁。

开场白：

上一节鸿哥列出了初学者七大误区，到底什么才是初学者关注的核心？那就是裸机奔跑的程序结构。一个好的程序结构，本身就是一个微型的多任务操作系统。鸿哥教给大家的就是如何编写这个简单的操作系统。在main函数循环中用switch语句实现多任务并行处理的任务切换，再外加一个定时器中断，这两者的结合就是鸿哥多年来所有实战项目的核心。鸿哥的程序结构看似简单，实际上就是那么简单。大家不用着急，本篇连载文章现在才正式开始，这一节我要教会大家两个知识点：
第一点：鸿哥首次提出的“三区一线”理论。此理论把程序代码分成三个区，一个延时分割线。
第二点：delay()延时的用途。
（1）硬件平台：基于朱兆祺51单片机学习板。
（2）实现功能：让一个LED闪烁。
（3）源代码讲解如下：

```
#include "REG52.H"
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayshort);
void delay_long(unsigned int uiDelaylong);
void led_flicker();
/* 注释一:
* 吴坚鸿个人的命名风格:凡是输出后缀都是_dr,凡是输入后缀都是_sr。
* dr代表drive驱动，sr代表sensor感应器
*/
sbit led_dr=P3^5;
void main()  //学习要点: 深刻理解鸿哥首次提出的三区一线理论
  {
/* 注释二:
* initial_myself()函数属于鸿哥三区一线理论的第一区，
* 专门用来初始化单片机自己的寄存器以及个别外围要求响应速度快的输出设备，
* 防止刚上电之后，由于输出IO口电平状态不确定而导致外围设备误动作，
* 比如继电器的误动作等等。
*/
   initial_myself();
/* 注释三:
* 此处的delay_long()延时函数属于第一区与第二区的分割线，
* 延时时间一般是0.3秒到2秒之间，等待外围芯片和模块上电稳定。
* 比如液晶模块，AT24C02存储芯片，DS1302时钟芯片，
* 这类芯片有个特点，一般都是跟单片机进行串口或并口通讯的，
* 并且不要求上电立即处理的。
*/
   delay_long(100);
/* 注释四:
* initial_peripheral()函数属于鸿哥三区一线理论的第二区，
* 专门用来初始化不要求上电立即处理的外围芯片和模块.
* 比如液晶模块，AT24C02存储芯片，DS1302时钟芯片。
* 本程序基于朱兆祺51单片机学习板。
*/
   initial_peripheral();
/* 注释五:
* while(1){}主函数循环区属于鸿哥三区一线理论的第三区，
* 专门用来编写被循环扫描到的非中断应用程序
*/
   while(1)
   {
      led_flicker();   //LED闪烁应用程序
   }
}
void led_flicker() //LED闪烁应用程序
{
  led_dr=1;  //LED亮
  delay_short(50000);  //延时50000个空指令的时间
/* 注释六:
* delay_long(100)延时50000个空指令的时间，因为内嵌了一个500次的for循环
```

```
*/
  led_dr=0;  //LED灭
  delay_long(100);     //延时50000个空指令的时间
}
/* 注释七:
* delay_short(unsigned int uiDelayShort)是小延时函数,
* 专门用在时序驱动的小延时, 一般uiDelayShort的数值取10左右,
* 最大一般也不超过100.本例为了解释此函数的特点, 取值范围超过100。
* 此函数的特点是时间的细分度高, 延时时间不宜过长。uiDelayShort数值
* 的大小就代表里面执行了多少条空指令的时间。数值越大, 延时越长。
* 时间精度不要刻意去计算, 感觉差不多就行。
*/
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
/* 注释八:
* delay_long(unsigned int uiDelayLong)是大延时函数,
* 专门用在上电初始化的大延时,
* 此函数的特点是能实现比较长时间的延时, 细分度取决于内嵌for循环的次数,
* uiDelayLong的数值的大小就代表里面执行了多少次500条空指令的时间。
* 数值越大, 延时越长。时间精度不要刻意去计算, 感觉差不多就行。
*/
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //初始化单片机
{
  led_dr=0;  //LED灭
}
void initial_peripheral() //初始化外围
{
  ;   //本例为空
}
```

总结陈词:

鸿哥首次提出的“三区一线”理论概况了各种项目程序的基本分区。我后续的程序就按此分区编写。

Delay()函数的长延时适用在上电初始化。

Delay()函数的短延时适用在驱动时序的脉冲延时，此时的时间不能太长，本例中暂时没有列出这方面的例子，在后面的章节中会提到。
在本例源代码中，在led_flicker()闪烁应用程序里用到的两个延时delay，它们的延时时间都太长了，在实战项目中肯定不能用这种延时，因为消耗的时间太长了，其它任务根本没有机会执行。那怎么办呢？我们应该如何改善？欲知详情，请听下回分解-----累计主循环次数使LED灯闪烁。
（未完待续，下节更精彩，不要走开哦）

第三节：累计主循环次数使LED灯闪烁。

开场白:
上一节鸿哥提到delay()延时函数消耗的时间太长了，其它任务根本没有机会执行，我们该怎么改善？本节教大家利用累计主循环次数的方法来解决这个问题。这一节要教会大家两个知识点:
第一点：利用累计主循环次数的方法实现时间延时
第二点：switch核心语句之初体验。 鸿哥所有的实战项目都是基于switch语句实现多任务并行处理。
（1）硬件平台：基于朱兆祺51单片机学习板。
（2）实现功能：让一个LED闪烁。
（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* const_time_level是统计循环次数的设定上限，数值越大，LED延时的时间越久
*/
#define const_time_level 10000
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void led_flicker();
sbit led_dr=P3^5;
/* 注释二:
* 吴坚鸿个人的命名风格:凡是switch语句里面的步骤变量后缀都是Step.
* 前缀带uc,ui,ul分别表示此变量是unsigned char,unsigned int,unsigned long.
*/
unsigned char ucLedStep=0; //步骤变量
unsigned int  uiTimeCnt=0; //统计循环次数的延时计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker();
   }
}
void led_flicker() ////第三区 LED闪烁应用程序
{

  switch(ucLedStep)
  {
     case 0:
/* 注释三:
* uiTimeCnt累加循环次数，只有当它的次数大于或等于设定上限const_time_level时，
* 才会去改变LED灯的状态，否则CPU退出led_flicker()任务，继续快速扫描其他的任务，
```

```
* 这样的程序结构就可以达到多任务并行处理的目的。
* 本程序基于朱兆祺51单片机学习板
*/
          uiTimeCnt++;  //累加循环次数，
                  if(uiTimeCnt>=const_time_level) //时间到
                  {
                     uiTimeCnt=0; //时间计数器清零
              led_dr=1;    //让LED亮
                          ucLedStep=1; //切换到下一个步骤
                  }
               break;
     case 1:
          uiTimeCnt++;  //累加循环次数，
                  if(uiTimeCnt>=const_time_level) //时间到
                  {
                     uiTimeCnt=0; //时间计数器清零
              led_dr=0;    //让LED灭
                          ucLedStep=0; //返回到上一个步骤
                  }
               break;

  }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=0;  //LED灭
}
void initial_peripheral() //第二区 初始化外围
{
  ;   //本例为空
}
```

总结陈词:

在实际项目中，用累计主循环次数实现时间延时是一个不错的选择。这种方法能胜任多任务处理的程序框架，但是它本身也有一个小小的不足。随着主函数里任务量的增加，我们为了保证延时时间的准确性，要不断修正设定上限const_time_level 。我们该怎么解决这个问题呢？欲知详情，请听下回分解-----累计定时中断次数使LED灯闪烁。（未完待续，下节更精彩，不要走开哦）

第四节：累计定时中断次数使LED灯闪烁。

开场白:

上一节提到在累计主循环次数来实现计时，随着主函数里任务量的增加，为了保证延时时间的准确性，要不断修正设定上限阀值const_time_level 。我们该怎么解决这个问题呢？本节教大家利用累计定时中断次数的方法来解决这个问题。这一节要教会大家四个知识点:
第一点: 利用累计定时中断次数的方法实现时间延时
第二点: 展现鸿哥最完整的实战程序框架。在主函数循环里用switch语句实现状态机的切换，在定时中断里累计中断次数，这两个的结合就是我写代码最本质的框架思想。
第三点: 提醒大家C语言中的int ,long变量是由几个字节构成的数据，凡是在main函数和中断函数里有可能同时改变的变量，这个变量应该在主函数中被更改之前，先关闭相应的中断，更改完了此变量，再打开中断，否则会留下不宜察觉的漏洞。当然在大部分的项目中可以不用这么操作，但是在一些要求非常高的项目中，有一些核心变量必须这么做。
第四点: 定时中断的初始值该怎么设置。不用严格按公式来计算时间，一般取个经验值是最大初始值减去1000就可以了。
具体内容，请看源代码讲解。
（1）硬件平台: 基于朱兆祺51单片机学习板。
（2）实现功能: 让一个LED闪烁。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level 200
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void led_flicker();
void T0_time();  //定时中断函数
sbit led_dr=P3^5;
unsigned char ucLedStep=0; //步骤变量
unsigned int  uiTimeCnt=0; //统计定时中断次数的延时计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker();
   }
}
void led_flicker() ////第三区 LED闪烁应用程序
{

  switch(ucLedStep)
  {
     case 0:
/* 注释一:
* uiTimeCnt累加定时中断的次数，每一次定时中断它都会在中断函数里自加一。
* 只有当它的次数大于或等于设定上限const_time_level时，
* 才会去改变LED灯的状态，否则CPU退出led_flicker()任务，继续快速扫描其他的任务，
* 这样的程序结构就可以达到多任务并行处理的目的。这就是鸿哥在所有开发项目中的核心框架。
*/
                  if(uiTimeCnt>=const_time_level) //时间到
                  {
/* 注释二:
```

```
* ET0=0;uiTimeCnt=0;ET0=1;----在清零uiTimeCnt之前，为什么要先禁止定时中断?
* 因为uiTimeCnt是unsigned int类型，本质上是由两个字节组成。
* 在C语言中uiTimeCnt=0看似一条指令，实际上经过编译之后它不只一条汇编指令。
* 由于定时中断函数里也对这个变量进行累加操作，如果不禁止定时中断，
* 那么uiTimeCnt这个变量在main()函数中还没被完全清零的时候，如果这个时候
* 突然来一个定时中断，并且在中断里又更改了此变量，这种情况在某些要求高的
* 项目上会是一个不容易察觉的漏洞，为项目带来隐患。当然，大部分的普通项目，
* 都可以不用那么严格，可以不用禁止定时中断。在这里只是提醒各位初学者有这种情况。
*/
            ET0=0;  //禁止定时中断
                    uiTimeCnt=0; //时间计数器清零
            ET0=1; //开启定时中断
            led_dr=1;    //让LED亮
                        ucLedStep=1; //切换到下一个步骤
                }
             break;
    case 1:
                if(uiTimeCnt>=const_time_level) //时间到
                {
            ET0=0;  //禁止定时中断
                    uiTimeCnt=0; //时间计数器清零
            ET0=1;   //开启定时中断
            led_dr=0;    //让LED灭
                        ucLedStep=0; //返回到上一个步骤
                }
             break;

  }
}
/* 注释三:
* C51的中断函数格式如下:
* void 函数名() interrupt 中断号
* {
*    中断程序内容
* }
* 函数名可以随便取，只要不是编译器已经征用的关键字。
* 这里最关键的是中断号，不同的中断号代表不同类型的中断。
* 定时中断的中断号是 1.至于其它中断的中断号，大家可以查找
* 相关书籍和资料。大家进入中断时，必须先清除中断标志，并且
* 关闭中断，然后再写代码，最后出来时，记得重装初始值，并且
* 打开中断。
*/
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
     uiTimeCnt++;  //累加定时中断的次数，
  }
```

```
TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
TL0=0x2f;
TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释四:
* 单片机有几个定时器，每个定时器又有几种工作方式，
* 那么多种变化，我们记不了那么多，怎么办?
* 大家记住鸿哥的话,无论一个单片机有多少内置资源，
* 我们做系统框架的，只需要一个定时器，一种工作方式。
* 开定时器越多这个系统越不好。需要哪种定时工作方式呢?
* 就需要响应定时中断后重装一下初始值继续跑那种。
* 在51单片机中就是工作方式1。其它的工作方式很少项目能用到。
*/
  TMOD=0x01;  //设置定时器0为工作方式1
  /* 注释五:
* 装定时器的初始值，就像一个水桶里装的水。如果这个桶是空桶，那么想
* 把这个桶灌满水的时间就很长，如果是里面已经装了大半的水，那么想
* 把这个桶灌满水的时间就相对比较短。也就是定时器初始值越小，产生一次
* 定时中断的时间就越长。如果初始值太小了，每次产生定时中断
* 的时间分辨率太粗，如果初始值太大了，虽然每次产生定时中断的时间分辨率很细，
* 但是太频繁的产生中断，不但会影响主函数main()的执行效率，而且累记中断次数
* 的时间误差也会很大。凭鸿哥多年的江湖经验，
* 我觉得最大初始值减去2000是比较好的经验值。当然，大一点小一点没关系。不要走
* 两个极端就行。
*/
TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
TL0=0x2f;
  led_dr=0;  //LED灭
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

本节程序麻雀虽小五脏俱全。在本节中已经展示了我最完整的实战程序框架。
本节程序只有一个LED灯闪烁的单任务，如果要多增加一个任务来并行处理，该怎么办?
欲知详情，请听下回分解-----蜂鸣器的驱动程序。
（未完待续，下节更精彩，不要走开哦）

第五节：蜂鸣器的驱动程序。

开场白:
上一节讲了利用累计定时中断次数实现LED灯闪烁，这个例子同时也第一次展示了我最完整的实战程序框架：用switch语句实现状态机，外加定时中断。这个框架看似简单，实际上就是那么简单。我做的所有开发项目都是基于这个简单框架，但是非常好用。上一节只有一个单任务的LED灯在闪烁，这节开始，我们多增加一个蜂鸣器报警的任务，要教会大家四个知识点:
第一点：蜂鸣器的驱动程序框架编写。
第二点：多任务处理的程序框架。
第三点：如何控制蜂鸣器声音的长叫和短叫。
第四点：如何知道1秒钟需要多少个定时中断，也就是如何按比例修正时间精度。
具体内容，请看源代码讲解。
（1）硬件平台：基于朱兆祺51单片机学习板。
（2）实现功能：同时跑两个任务，第一个任务让一个LED灯1秒钟闪烁一次。第二个任务让蜂鸣器在前面3秒发生一次短叫报警，在后面6秒发生一次长叫报警，反复循环。
（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 如何知道1秒钟需要多少个定时中断?
* 这个需要编写一段小程序测试，得到测试的结果后再按比例修正。
* 步骤:
* 第一步：在程序代码上先写入1秒钟大概需要200个定时中断。
* 第二步：基于以上1秒钟的基准，编写一个60秒的简单测试程序(如果编写超过
* 60秒的时间，这个精度还会更高)。比如，编写一个用蜂鸣器的声音来识别计时的
* 起始和终止的测试程序。
* 第三步：把程序烧录进单片机后，上电开始测试，手上同步打开手机里的秒表。
*         如果单片机仅仅跑了27秒。
* 第四步：那么最终得出1秒钟需要的定时中断次数是:const_time_1s=(200*60)/27=444
*/
#define const_time_05s 222   //0.5秒钟的时间需要的定时中断次数
#define const_time_1s 444   //1秒钟的时间需要的定时中断次数
#define const_time_3s 1332   //3秒钟的时间需要的定时中断次数
#define const_time_6s 2664   //6秒钟的时间需要的定时中断次数
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_voice_long   200  //蜂鸣器长叫的持续时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void led_flicker();
void alarm_run();
void T0_time();  //定时中断函数
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //LED灯的驱动IO口
unsigned char ucLedStep=0; //LED灯的步骤变量
unsigned int  uiTimeLedCnt=0; //LED灯统计定时中断次数的延时计数器
unsigned char ucAlarmStep=0; //报警的步骤变量
unsigned int  uiTimeAlarmCnt=0; //报警统计定时中断次数的延时计数器
```

```
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker();  //第一个任务LED灯闪烁
          alarm_run();    //第二个任务报警器定时报警
   }
}
void led_flicker() //第三区 LED闪烁应用程序
{

  switch(ucLedStep)
  {
     case 0:
           if(uiTimeLedCnt>=const_time_05s) //时间到
           {
             uiTimeLedCnt=0; //时间计数器清零
             led_dr=1;    //让LED亮
             ucLedStep=1; //切换到下一个步骤
           }
           break;
     case 1:
           if(uiTimeLedCnt>=const_time_05s) //时间到
           {
              uiTimeLedCnt=0; //时间计数器清零
              led_dr=0;    //让LED灭
              ucLedStep=0; //返回到上一个步骤
           }
           break;
  }
}
void alarm_run() //第三区 报警器的应用程序
{

  switch(ucAlarmStep)
  {
     case 0:
           if(uiTimeAlarmCnt>=const_time_3s) //时间到
           {
             uiTimeAlarmCnt=0; //时间计数器清零
/* 注释二:
* 只要变量uiVoiceCnt不为0，蜂鸣器就会在定时中断函数里启动鸣叫，并且自减uiVoiceCnt
* 直到uiVoiceCnt为0时才停止鸣叫。因此控制uiVoiceCnt变量的大小就是控制声音的长短。
*/
             uiVoiceCnt=const_voice_short;  //蜂鸣器短叫
             ucAlarmStep=1; //切换到下一个步骤
```

```
            }
            break;
      case 1:
            if(uiTimeAlarmCnt>=const_time_6s) //时间到
            {
               uiTimeAlarmCnt=0; //时间计数器清零
               uiVoiceCnt=const_voice_long;  //蜂鸣器长叫
               ucAlarmStep=0; //返回到上一个步骤
            }
            break;
   }
}
void T0_time() interrupt 1
{
   TF0=0;  //清除中断标志
   TR0=0; //关中断
   if(uiTimeLedCnt<0xffff)  //设定这个条件，防止uiTimeLedCnt超范围。
   {
       uiTimeLedCnt++;  //LED灯的时间计数器，累加定时中断的次数，
   }
   if(uiTimeAlarmCnt<0xffff)  //设定这个条件，防止uiTimeAlarmCnt超范围。
   {
       uiTimeAlarmCnt++;  //报警的时间计数器，累加定时中断的次数，
   }
/* 注释三:
* 为什么不把驱动蜂鸣器这段代码放到main函数的循环里去?
* 因为放在定时中断里，能保证蜂鸣器的声音长度是一致的，
* 如果放在main循环里，声音的长度就有可能受到某些必须
* 一气呵成的任务干扰，得不到及时响应，影响声音长度的一致性。
*/
   if(uiVoiceCnt!=0)
   {
      uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
          beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
   }
   else
   {
      ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
            beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
   }
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
   TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
```

```
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  led_dr=0;  //LED灭
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

本节程序已经展示了一个多任务处理的基本思路，假如要实现一个独立按键检测，能不能也按照这种思路来处理呢？欲知详情，请听下回分解-----在主函数中利用累计主循环次数来实现独立按键的检测。

（未完待续，下节更精彩，不要走开哦）

第六节: 在主函数中利用累计主循环次数来实现独立按键的检测。

开场白:

上一节讲了多任务中蜂鸣器驱动程序的框架，这节继续利用多任务处理的方式，在主函数中利用累计主循环次数来实现独立按键的检测。要教会大家四个知识点:

第一点: 独立按键的驱动程序框架。

第二点: 用累计主循环次数来实现去抖动的延时。

第三点: 灵活运用防止按键不松手后一直触发的按键自锁标志。

第四点: 在按键去抖动延时计时中，添加一个抗干扰的软件监控判断。一旦发现瞬间杂波干扰，马上把延时计数器清零。这种方法是我在复杂的工控项目中总结出来的。以后凡是用到开关感应器的地方，都可以用类似的方法实现软件上的抗干扰处理。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能: 有两个独立按键，每按一个独立按键，蜂鸣器发出“滴”的一声后就停。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
* 调整抖动时间阀值的大小，可以更改按键的触发灵敏度。
* 去抖动的时间本质上等于累计主循环次数的时间。
*/
#define const_key_time1  500    //按键去抖动延时的时间
#define const_key_time2  500    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
```

```
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_scan(); //按键扫描函数
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数
{
/* 注释二:
* 独立按键扫描的详细过程:
* 第一步: 平时没有按键被触发时，按键的自锁标志和去抖动延时计数器一直被清零。
* 第二步: 一旦有按键被按下，去抖动延时计数器开始累加，在还没累加到
*         阀值const_key_time1时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO口突然瞬间触发成高电平，这个时候马上又把延时计数器uiKeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了阀值const_key_time1，则触发按键，把编号ucKeySec赋值。
*         同时，马上把自锁标志ucKeyLock1置位，防止按住按键不松手后一直触发。
* 第四步: 等按键松开后，自锁标志ucKeyLock1及时清零，为下一次自锁做准备。
* 第五步: 以上整个过程，就是识别按键IO口下降沿触发的过程。
*/
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
         uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     ++uiKeyTimeCnt1;  //延时计数器
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
```

```
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
      }
  }
  if(key_sr2==1)
  {
     ucKeyLock2=0;
         uiKeyTimeCnt2=0;
  }
  else if(ucKeyLock2==0)
  {
     ++uiKeyTimeCnt2;
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;
        ucKeySec=2;     //触发2号键
     }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S5键
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
```

```
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

本节程序已经展示了在主函数中，利用累计主循环次数来实现独立按键的检测。这种方法我经常在实战用应用，但是它也有一个小小的不足，随着在主函数循环中任务量的增加，为了保证去抖动延时的时间一致性，要适当调整一下去抖动的阀值const_key_time1。如何解决这个问题呢？欲知详情，请听下回分解-----在主函数中利用累计定时中断的次数来实现独立按键的检测。

（未完待续，下节更精彩，不要走开哦）

第七节：在主函数中利用累计定时中断的次数来实现独立按键的检测。

开场白:

上一节讲了在主函数中利用累计主循环次数来实现独立按键的检测，但是它也有一个小小的不足，随着在主函数中任务量的增加，为了保证去抖动延时的时间一致性，要适当调整一下去抖动的时间阀值const_key_time1。如何解决这个问题呢？这一节教大家在主函数中利用累计定时中断的次数来实现独立按键的检测，可以有效地避免这个问题。要教会大家一个知识点：如何在上一节的基础上，略作修改，就可以在主函数中，利用累计定时中断的次数来实现去抖动的延时。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能：有两个独立按键，每按一个独立按键，蜂鸣器发出“滴”的一声后就停。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
* 调整抖动时间阀值的大小，可以更改按键的触发灵敏度。
* 去抖动的时间本质上等于累计定时中断次数的时间。
*/
#define const_key_time1  30    //按键去抖动延时的时间
#define const_key_time2  30    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned char ucKeyStartFlag1=0; //启动定时中断计数的开关
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned char ucKeyStartFlag2=0; //启动定时中断计数的开关
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_scan(); //按键扫描函数
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数
{
/* 注释二:
* 独立按键扫描的详细过程:
* 第一步: 平时没有按键被触发时，按键的自锁标志,计时器开关和去抖动延时计数器一直被清零。
* 第二步: 一旦有按键被按下，启动计时器，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值const_key_time1时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO口突然瞬间触发成高电平，这个时候马上停止计时，并且把延时计数器uiKeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了阀值const_key_time1，则触发按键，把编号ucKeySec赋值。
*         同时，马上把自锁标志ucKeyLock1置位，防止按住按键不松手后一直触发。
```

```
* 第四步: 等按键松开后，自锁标志ucKeyLock1及时清零，为下一次自锁做准备。
* 第五步: 以上整个过程，就是识别按键IO口下降沿触发的过程。
*/
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
         ucKeyStartFlag1=0; //停止计数器
         uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
         ucKeyStartFlag1=1; //启动计数器
     if(uiKeyTimeCnt1>const_key_time1)
     {
                 ucKeyStartFlag1=0; //停止计数器
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)
  {
     ucKeyLock2=0;
         ucKeyStartFlag2=0; //停止计数器
         uiKeyTimeCnt2=0;
  }
  else if(ucKeyLock2==0)
  {
         ucKeyStartFlag2=1; //启动计数器
     if(uiKeyTimeCnt2>const_key_time2)
     {
            ucKeyStartFlag2=0; //停止计数器
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;
        ucKeySec=2;     //触发2号键
     }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
               uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
               ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S5键
               uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
               ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
```

```
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(ucKeyStartFlag1==1)//启动计数器
  {
     if(uiKeyTimeCnt1<0xffff)  //防止计数器超范围
         {
            uiKeyTimeCnt1++;
         }
  }
   if(ucKeyStartFlag2==1)//启动计数器
  {
     if(uiKeyTimeCnt2<0xffff) //防止计数器超范围
         {
            uiKeyTimeCnt2++;
         }
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
```

```
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

本节程序已经展示了在主函数中，利用累计定时中断次数来实现独立按键的检测。这种方法我也经常在实战用应用，但是如果在某些项目中，需要在主函数里间歇性地执行一些一气呵成的耗时任务，这种方法就不是很实用，因为当主函数正在处理一气呵成的耗时任务时，这个时候如果有按键按下来，就有可能没有及时被响应到而遗漏了。那有什么方法可以解决这类项目中遇到的问题?欲知详情，请听下回分解-----在定时中断函数里执行独立按键的扫描程序。

（未完待续，下节更精彩，不要走开哦）

第八节: 在定时中断函数里执行独立按键的扫描程序。

开场白:

上一节讲了在主函数中利用累计定时中断的次数来实现独立按键的检测，但是如果在某些项目中，需要在主函数里间歇性地执行一些一气呵成的耗时任务，当主函数正在处理一气呵成的耗时任务时（前提是没有关闭定时器中断），这个时候如果有按键按下来，就有可能没有及时被响应到而遗漏了。在定时中断函数里处理独立按键的扫描程序，可以避免这个问题。要教会大家一个知识点: 如何在上一节的基础上，略作修改，就可以在定时中断函数里处理独立按键的扫描程序。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能: 有两个独立按键，每按一个独立按键，蜂鸣器发出“滴”的一声后就停。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
* 调整抖动时间阀值的大小，可以更改按键的触发灵敏度。
* 去抖动的时间本质上等于累计定时中断次数的时间。
*/
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
```

```
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeySec=0;    //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释二:
* 独立按键扫描的详细过程:
* 第一步: 平时没有按键被触发时，按键的自锁标志,去抖动延时计数器一直被清零。
* 第二步: 一旦有按键被按下，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值const_key_time1时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO口突然瞬间触发成高电平，这个时候马上把延时计数器uiKeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了阀值const_key_time1，则触发按键，把编号ucKeySec赋值。
*         同时，马上把自锁标志ucKeyLock1置位，防止按住按键不松手后一直触发。
* 第四步: 等按键松开后，自锁标志ucKeyLock1及时清零，为下一次自锁做准备。
* 第五步: 以上整个过程，就是识别按键IO口下降沿触发的过程。
*/
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
         uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)
  {
     ucKeyLock2=0;
```

```
        uiKeyTimeCnt2=0;
  }
  else if(ucKeyLock2==0)
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;
        ucKeySec=2;      //触发2号键
     }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S5键
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
```

```
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

本节程序已经展示了在定时中断函数里执行独立按键的扫描程序。这节和前面两节所讲的扫描方式，我都在项目上用过，具体跟项目的侧重点不同来选择不同的方式，我本人用得最多的就是当前这种方式。假如要独立按键实现类似鼠标的双击功能，我们改怎么写程序？欲知详情，请听下回分解-----独立按键的双击按键触发。

（未完待续，下节更精彩，不要走开哦）

第九节: 独立按键的双击按键触发。

开场白:

上一节讲了在定时中断函数里处理独立按键的扫描程序，这种结构的程序我用在了很多项目上。这一节教大家如何实现按键双击触发的功能，这种功能类似鼠标的双击。要教会大家一个知识点: 如何在上一节的基础上，略作修改，就可以实现按键的双击功能。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能: 有两个独立按键，每双击一个独立按键，蜂鸣器发出“滴”的一声后就停。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
* 调整抖动时间阀值的大小，可以更改按键的触发灵敏度。
* 去抖动的时间本质上等于累计定时中断次数的时间。
*/
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
```

```
/* 注释二:
* 有效时间差，是指连续两次按键触发的最大有效间隔时间。
* 如果双击的两个按键按下的时间间隔太长，则视为无效双击。
*/
#define const_interval_time1  200     //连续两次按键之间的有效时间差
#define const_interval_time2  200     //连续两次按键之间的有效时间差
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned char ucKeyTouchCnt1=0; //按键按下的次数记录
unsigned int  uiKeyIntervalCnt1=0; //按键间隔的时间计数器
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned char ucKeyTouchCnt2=0; //按键按下的次数记录
unsigned int  uiKeyIntervalCnt2=0; //按键间隔的时间计数器
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释三:
* 独立双击按键扫描的详细过程:
* 第一步: 平时没有按键被触发时，按键的自锁标志,去抖动延时计数器一直被清零。
*         如果之前已经有按键触发过一次，那么启动时间间隔计数器uiKeyIntervalCnt1,
*         在这个允许的时间差范围内，如果一直没有第二次按键触发，则把累加按键触发的
*         次数ucKeyTouchCnt1也清零。
* 第二步: 一旦有按键被按下，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值const_key_time1时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO口突然瞬间触发成高电平，这个时候马上把延时计数器uiKeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了阀值const_key_time1，马上把自锁标志ucKeyLock1置位，
```

```
*          防止按住按键不松手后一直触发。与此同时，累加一次按键次数，如果按键次数累加有两次以上，
*          则认为触发双击按键，并把编号ucKeySec赋值。
* 第四步：等按键松开后，自锁标志ucKeyLock1及时清零，为下一次自锁做准备。并且累加间隔时间，
*          防止两次按键的间隔时间太长。
* 第五步：以上整个过程，就是识别按键IO口下降沿触发的过程。
*/
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
         ucKeyLock1=0; //按键自锁标志清零
         uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
                 if(ucKeyTouchCnt1>0) //之前已经有按键触发过一次，再来一次就构成双击
                 {
                     uiKeyIntervalCnt1++; //按键间隔的时间计数器累加
                         if(uiKeyIntervalCnt1>const_interval_time1) //超过最大允许的间隔时间
                         {
                            uiKeyIntervalCnt1=0; //时间计数器清零
                            ucKeyTouchCnt1=0; //清零按键的按下的次数
                         }
                 }
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
                uiKeyIntervalCnt1=0; //按键有效间隔的时间计数器清零
                ucKeyTouchCnt1++;
                if(ucKeyTouchCnt1>1)  //连续被按了两次以上
                {
                    ucKeyTouchCnt1=0;  //统计按键次数清零
                    ucKeySec=1;     //触发1号键
                }
     }
  }
  if(key_sr2==1)
  {
         ucKeyLock2=0;
         uiKeyTimeCnt2=0;
                  if(ucKeyTouchCnt2>0)
                 {
                     uiKeyIntervalCnt2++; //按键间隔的时间计数器累加
                         if(uiKeyIntervalCnt2>const_interval_time2) //超过最大允许的间隔时间
                         {
                            uiKeyIntervalCnt2=0; //时间计数器清零
                            ucKeyTouchCnt2=0; //清零按键的按下的次数
                         }
                 }
  }
```

```
  else if(ucKeyLock2==0)
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;
                uiKeyIntervalCnt2=0; //按键有效间隔的时间计数器清零
                ucKeyTouchCnt2++;
                if(ucKeyTouchCnt2>1)  //连续被按了两次以上
                {
                    ucKeyTouchCnt2=0;  //统计按键次数清零
                    ucKeySec=2;     //触发2号键
                }
     }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 双击  对应朱兆祺学习板的S1键
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 双击  对应朱兆祺学习板的S5键
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
```

```
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释四:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

假如要两个独立按键实现组合按键的功能，我们该怎么写程序？欲知详情，请听下回分解-----独立按键的组合按键触发。

（未完待续，下节更精彩，不要走开哦）

第十节: 两个独立按键的组合按键触发。

开场白:

上一节讲了按键双击触发功能的程序，这一节讲类似电脑键盘组合按键触发的功能，要教会大家一个知识点: 如何在上一节的基础上，略作修改，就可以实现两个独立按键的组合按键触发功能。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能: 有两个独立按键，当把两个独立按键都按下后，蜂鸣器发出“滴”的一声后就停。直到松开任一个按键后，才能重新进行下一次的组合按键触发。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
* 调整抖动时间阀值的大小，可以更改按键的触发灵敏度。
* 去抖动的时间本质上等于累计定时中断次数的时间。
```

```
*/
#define const_key_time12  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt12=0; //按键去抖动延时计数器
unsigned char ucKeyLock12=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释二:
* 独立组合按键扫描的详细过程:
* 第一步: 平时只要两个按键中有一个没有被按下时，按键的自锁标志,去抖动延时计数器一直被清零。
* 第二步: 一旦两个按键都被按下，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值const_key_time12时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO口突然瞬间触发成高电平，这个时候马上把延时计数器uiKeyTimeCnt12
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了阀值const_key_time12，马上把自锁标志ucKeyLock12置位，
*         防止按住按键不松手后一直触发。并把编号ucKeySec赋值。 组合按键触发
* 第四步: 等按键松开后，自锁标志ucKeyLock12及时清零，为下一次自锁做准备。
* 第五步: 以上整个过程，就是识别按键IO口下降沿触发的过程。
*/
  if(key_sr1==1||key_sr2==1)//IO是高电平，说明两个按键没有全部被按下，这时要及时清零一些标志位
  {
         ucKeyLock12=0; //按键自锁标志清零
         uiKeyTimeCnt12=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock12==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt12++; //累加定时中断次数
     if(uiKeyTimeCnt12>const_key_time12)
```

```
      {
         uiKeyTimeCnt12=0;
         ucKeyLock12=1;  //自锁按键置位,避免一直触发
         ucKeySec=1;     //触发1号键

      }
   }
}
void key_service() //第三区 按键服务的应用程序
{
   switch(ucKeySec) //按键服务状态切换
   {
     case 1:// 1号键 组合按键  对应朱兆祺学习板的S1键和S5键
               uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
               ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
           break;

   }
}
void T0_time() interrupt 1
{
   TF0=0;  //清除中断标志
   TR0=0; //关中断
   key_scan(); //按键扫描函数
   if(uiVoiceCnt!=0)
   {
      uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
          beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
   }
   else
   {
      ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
            beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
   }
   TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
   TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
```

```
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

以前寻呼机流行的时候，寻呼机往往只有一个设置按键，它要求用一个按键来设置不同的参数，这个时候就要用到同一个按键来实现短按和长按的区别触发功能。要现实这种功能，我们该怎么写程序？欲知详情，请听下回分解-----同一个按键短按与长按的区别触发。

（未完待续，下节更精彩，不要走开哦）

第十一节：同一个按键短按与长按的区别触发。

开场白：

上一节讲了类似电脑键盘组合按键触发的功能，这节要教会大家一个知识点：如何在上一节的基础上，略作修改，就可以实现同一个按键短按与长按的区别触发。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能：两个独立按键S1和S5，按住其中一个按键，在短时间内松手，则认为是短按，触发蜂鸣器短鸣一声。如果一直按住这个按键不松手，那么超过规定的长时间内，则认为是长按，触发蜂鸣器长鸣一声。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  20   //蜂鸣器短叫的持续时间
#define const_voice_long   140   //蜂鸣器长叫的持续时间
/* 注释一:
* 调整抖动时间阀值的大小，可以更改按键的触发灵敏度。
* 去抖动的时间本质上等于累计定时中断次数的时间。
*/
#define const_key_time_short1  20    //短按的按键去抖动延时的时间
#define const_key_time_long1   400     //长按的按键去抖动延时的时间
#define const_key_time_short2  20    //短按的按键去抖动延时的时间
#define const_key_time_long2   400     //长按的按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned char ucShortTouchFlag1=0; //短按的触发标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
```

```
unsigned char ucShortTouchFlag2=0; //短按的触发标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释二:
* 长按与短按的按键扫描的详细过程:
* 第一步: 平时只要按键没有被按下时，按键的自锁标志,去抖动延时计数器一直被清零。
* 第二步: 一旦两个按键都被按下，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值const_key_time_short1或者const_key_time_long1时，如果在这期间由于受外界干扰或者按键抖动
, 而使
*         IO口突然瞬间触发成高电平，这个时候马上把延时计数器uiKeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了短按阀值const_key_time_short1，则马上把短按标志ucShortTouchFlag1=1;
*         如果还没有松手，一旦发现按下的时间超过长按阀值const_key_time_long1时，
*         先把短按标志ucShortTouchFlag1清零，然后触发长按。在这段程序里，把自锁标志ucKeyLock1置位,
*         是为了防止按住按键不松手后一直触发。
* 第四步: 等按键松开后，自锁标志ucKeyLock12及时清零，为下一次自锁做准备。如果发现ucShortTouchFlag1等于
1,
*         说明短按有效，这时触发一次短按。
* 第五步: 以上整个过程，就是识别按键IO口下降沿触发的过程。
*/
  if(key_sr1==1)//IO是高电平，说明两个按键没有全部被按下，这时要及时清零一些标志位
  {
      ucKeyLock1=0; //按键自锁标志清零
      uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
            if(ucShortTouchFlag1==1)  //短按触发标志
          {
             ucShortTouchFlag1=0;
                 ucKeySec=1;    //触发一号键的短按
          }
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time_short1)
     {
            ucShortTouchFlag1=1;   //激活按键短按的有效标志
     }
     if(uiKeyTimeCnt1>const_key_time_long1)
```

```
    {
          ucShortTouchFlag1=0;  //清除按键短按的有效标志
       uiKeyTimeCnt1=0;
       ucKeyLock1=1;  //自锁按键置位,避免一直触发
       ucKeySec=2;    //触发1号键的长按

    }
  }
  if(key_sr2==1)//IO是高电平，说明两个按键没有全部被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
          if(ucShortTouchFlag2==1)  //短按触发标志
         {
            ucShortTouchFlag2=0;
                ucKeySec=3;    //触发2号键的短按
         }
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time_short2)
     {
          ucShortTouchFlag2=1;   //激活按键短按的有效标志
     }
     if(uiKeyTimeCnt2>const_key_time_long2)
     {
          ucShortTouchFlag2=0;  //清除按键短按的有效标志
       uiKeyTimeCnt2=0;
       ucKeyLock2=1;  //自锁按键置位,避免一直触发
       ucKeySec=4;    //触发2号键的长按

     }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键的短按  对应朱兆祺学习板的S1键
          uiVoiceCnt=const_voice_short; //按键声音的短触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 1号键的长按  对应朱兆祺学习板的S1键
          uiVoiceCnt=const_voice_long; //按键声音的长触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 2号键的短按  对应朱兆祺学习板的S5键
          uiVoiceCnt=const_voice_short; //按键声音的短触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
```

```
            break;
    case 4:// 2号键的长按  对应朱兆祺学习板的S5键
            uiVoiceCnt=const_voice_long; //按键声音的长触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
          beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
```

```
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

在很多需要人机交互的项目中，需要用按键来快速加减某个数值，这个时候如果按住一个按键不松手，这个数值要有节奏地快速往上加或者快速往下减。要现实这种功能，我们该怎么写程序？欲知详情，请听下回分解-----按住一个独立按键不松手的连续步进触发。

（未完待续，下节更精彩，不要走开哦）

第十二节：按住一个独立按键不松手的连续步进触发。

开场白:

上一节讲了同一个按键短按与长按的区别触发功能，这节要教会大家两个知识点:

第一个知识点：如何在上一节的基础上，略作修改，就可以实现按住一个独立按键不松手的连续步进触发。

第二个知识点：在单片机的C语言编译器中，当无符号数据0减去1时，就会溢出，变成这个类型数据的最大值。比如是unsigned int类型的0减去1就等于65535(0xffff),unsigned char类型的0减去1就等于255(0xff)。这个常识经常要用在判断数据临界点的地方。比如一个数最大值是20，最小值是0。这个数据一直往下减，当我们发现它突然大于20的时候，就知道它溢出了，这个时候要及时把它赋值成0就达到我们的目的。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能：两个独立按键S1和S5，S1键作为加键。S5键做为减键。每按一次S1键则被设置参数uiSetNumber自加1。如果按住S1键不松手超过1秒钟，被设置参数uiSetNumber以每0.25秒的时间间隔往上自加1，一直加到20为止。每按一次S5键则被设置参数uiSetNumber自减1。如果按住S5键不松手超过1秒钟，被设置参数uiSetNumber以每0.25秒的时间间隔往下自减1，一直减到0为止。当被设置参数uiSetNumber小于10的时候，LED灯灭；当大于或者等于10的时候，LED灯亮。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_time_0_25s  111   //0.25秒钟的时间需要的定时中断次数
#define const_time_1s     444   //1秒钟的时间需要的定时中断次数
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
void led_run();  //led灯的应用程序
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //LED的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned int  uiKeyCtntyCnt1=0;  //按键连续触发的间隔延时计数器
```

```
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned int  uiKeyCtntyCnt2=0;  //按键连续触发的间隔延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned int  uiSetNumber=0; //设置的数据
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
           led_run();  //led灯的应用程序
   }
}
void led_run()  //led灯的应用程序
{
   if(uiSetNumber<10)  //如果被设置的参数uiSetNumber小于10，LED灯则灭。否则亮。
   {
      led_dr=0;  //灭
   }
   else
   {
      led_dr=1;  //亮
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释一:
* 独立按键扫描的详细过程:
* 第一步: 平时没有按键被触发时，按键的自锁标志,去抖动延时计数器，以及时间间隔延时计数器一直被清零。
* 第二步: 一旦有按键被按下，去抖动延时计数器开始在定时中断函数里累加，在还没累加到
*         阀值const_key_time1时，如果在这期间由于受外界干扰或者按键抖动，而使
*         IO口突然瞬间触发成高电平，这个时候马上把延时计数器uiKeyTimeCnt1
*         清零了,这个过程非常巧妙，非常有效地去除瞬间的杂波干扰。这是我实战中摸索出来的。
*         以后凡是用到开关感应器的时候，都可以用类似这样的方法去干扰。
* 第三步: 如果按键按下的时间超过了阀值const_key_time1，则触发按键，把编号ucKeySec赋值。
*         同时，马上把自锁标志ucKeyLock1置位，防止按住按键不松手后一直触发。
* 第四步: 如果此时触发了一次按键后，一直不松手，去抖动延时计时器继续累加，直到超过了1秒钟。进入连续触发
模式的程序
* 第五步: 在连续触发模式的程序中，连续累加延时计数器开始累加，每0.25秒就触发一次。
* 第六步: 等按键松开后，自锁标志ucKeyLock1和两个延时计时器及时清零，为下一次自锁做准备。
*/
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
     uiKeyCtntyCnt1=0; //连续累加的时间间隔延时计数器清零
```

```
}
else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
{
   uiKeyTimeCnt1++; //累加定时中断次数
   if(uiKeyTimeCnt1>const_key_time1)
   {
      uiKeyTimeCnt1=0;
      ucKeyLock1=1;  //自锁按键置位,避免一直触发
      ucKeySec=1;    //触发1号键
   }
}
else if(uiKeyTimeCnt1<const_time_1s) //按住累加到1秒
{
   uiKeyTimeCnt1++;
}
else  //按住累加到1秒后仍然不放手，这个时候进入有节奏的连续触发
{
   uiKeyCtntyCnt1++; //连续触发延时计数器累加
       if(uiKeyCtntyCnt1>const_time_0_25s)  //按住没松手，每0.25秒就触发一次
       {
           uiKeyCtntyCnt1=0; //
       ucKeySec=1;    //触发1号键
       }

}
if(key_sr2==1)
{
   ucKeyLock2=0;
   uiKeyTimeCnt2=0;
       uiKeyCtntyCnt2=0;
}
else if(ucKeyLock2==0)
{
   uiKeyTimeCnt2++; //累加定时中断次数
   if(uiKeyTimeCnt2>const_key_time2)
   {
      uiKeyTimeCnt2=0;
      ucKeyLock2=1;
      ucKeySec=2;     //触发2号键
   }
}
else if(uiKeyTimeCnt2<const_time_1s)
{
    uiKeyTimeCnt2++;
}
else
{
    uiKeyCtntyCnt2++;
        if(uiKeyCtntyCnt2>const_time_0_25s)
        {
```

```
            uiKeyCtntyCnt2=0;
                ucKeySec=2;      //触发2号键
          }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 连续加键  对应朱兆祺学习板的S1键
              uiSetNumber++; //被设置的参数连续往上加
                          if(uiSetNumber>20) //最大是20
                          {
                            uiSetNumber=20;
                          }
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 连续减键  对应朱兆祺学习板的S5键
/* 注释二:
* 在单片机的C语言编译器中，当无符号数据0减去1时，就会溢出，变成这个类型数据的最大值。
* 比如是unsigned int的0减去1就等于65535(0xffff),unsigned char的0减去1就等于255(0xff)
*/
              uiSetNumber--; //被设置的参数连续往下减
                          if(uiSetNumber>20) //最小是0.为什么这里用20?因为0减去1就是溢出变成了
65535(0xffff)
                          {
                            uiSetNumber=0;
                          }
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
```

```
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  led_dr=0;  //LED灯灭
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

本程序可以有节奏地快速往上加或者快速往下减。假如被设置数据的范围不是20，而是1000。如果按0.25秒的节奏往上加，那不是累死人了?如果直接把0.25秒的节奏调快到0.01秒，那么到达999的时候，还来不及松手就很容易超过头，不好微调。有没有完整的方案解决这个问题？当然有。欲知详情，请听下回分解-----按住一个独立按键不松手的加速匀速触发。

（未完待续，下节更精彩，不要走开哦）

第十三节：按住一个独立按键不松手的加速匀速触发。

开场白:

上一节讲了按住一个独立按键不松手的连续步进触发功能，这节要教会大家如何在上一节的基础上，略作修改，就可以实现按键的加速匀速触发。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能：两个独立按键S1和S5，S1键作为加键。S5键做为减键。每按一次S1键则被设置参数uiSetNumber自加1。如果按住S1键不松手超过1秒钟，被设置参数uiSetNumber以不断变快的时间间隔往上自加1，这个称为加速触发的功

能，直到到达极限值，则以固定的速度加1，这个过程叫匀速。S5作为减法按键，每触发一次，uiSetNumber就减1，其加速和匀速触发功能跟S1按键一样。当被设置参数uiSetNumber小于500的时候，LED灯灭；当大于或者等于500的时候，LED灯亮。需要注意的是：

第一步：每次按下去触发一次单击按键，如果按下去到松手的时间不超过1秒，则不会进入连续加速触发模式。

第二步：如果按下去不松手的时间超过1秒，则进入连续加速触发模式。按键触发节奏不断加快，蜂鸣器鸣叫的节奏也不断加快。直到它们都到达一个极限值，然后以此极限值间隔匀速触发。在刚开始加速的时候，按键触发与蜂鸣器触发的步骤是一致的，等它们任意一个达到极限值的时候，急促的声音跟按键的触发不一致，并不是蜂鸣器每叫一次，按键就触发一次。实际上加速到最后，按键触发的速度远远比蜂鸣器的触发速度快。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_time_1s     444   //1秒钟的时间需要的定时中断次数
#define const_initial_set 160  //连续触发模式的时候，按键刚开始的间隔触发时间
#define const_min_level  30    //连续触发模式的时候，按键经过加速后，如果一旦发现小于这个值，则直接变到最后的间隔触发时间
#define const_sub_dt  10       //按键的"加速度"，相当于按键间隔时间每次的变化量
#define const_last_min_set 5    //连续触发模式的时候，按键经过加速后，最后的间隔触发时间
#define const_syn_min_level  45 //产生同步声音的最小阀值 这个时间必须要比蜂鸣器的时间略长一点。
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
void led_run();  //led灯的应用程序
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //LED的驱动IO口
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned int  uiKeyCtntyCnt1=0;  //按键连续触发的间隔延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiSynCtntyCnt1=0;   //产生按键同步声音的计数器
unsigned int  uiCtntyTimeSet1=const_initial_set; //按键每次触发的时间间隔，这数值不断变小，导致速度不断加快
unsigned int  uiCtntySynSet1=const_initial_set;//同步声音的时间间隔，这数值不断变小，导致速度不断加快
unsigned char ucCtntyFlag1=0;  //是否处于连续加速触发模式的标志位
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned int  uiKeyCtntyCnt2=0;  //按键连续触发的间隔延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiSynCtntyCnt2=0; //产生按键同步声音的计数器
unsigned int  uiCtntyTimeSet2=const_initial_set; //按键每次触发的时间间隔，这数值不断变小，导致速度不断加快
unsigned int  uiCtntySynSet2=const_initial_set; //同步声音的时间间隔，这数值不断变小，导致速度不断加快
unsigned char ucCtntyFlag2=0; //是否处于连续加速触发模式的标志位
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
```

```
unsigned int  uiSetNumber=0; //设置的数据
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
       led_run();  //led灯的应用程序
   }
}
void led_run()  //led灯的应用程序
{
   if(uiSetNumber<500)  //如果被设置的参数uiSetNumber小于500，LED灯则灭。否则亮。
   {
      led_dr=0;  //灭
   }
   else
   {
      led_dr=1;  //亮
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释一:
* 独立按键连续加速扫描的过程:
* 第一步: 每次按下去触发一次单击按键，如果按下去到松手的时间不超过1秒，则不会进入连续加速触发模式。
* 第二步: 如果按下去不松手的时间超过1秒，则进入连续加速触发模式。按键触发节奏不断加快，蜂鸣器鸣叫的节奏
*         也不断加快。直到它们都到达一个极限值，然后以此极限值间隔匀速触发。在刚开始加速的时候，按键触发
与
*         蜂鸣器触发的步骤是一致的，等它们任意一个达到极限值的时候，急促的声音跟按键的触发不一致，并不是
*         蜂鸣器每叫一次，按键就触发一次。实际上加速到最后，按键触发的速度远远比蜂鸣器的触发速度快。
*/
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
     uiKeyCtntyCnt1=0; //按键连续加速的时间间隔延时计数器清零
         uiSynCtntyCnt1=0;  //蜂鸣器连续加速的时间间隔延时计数器清零
     uiCtntyTimeSet1=const_initial_set; //按键每次触发的时间间隔初始值，这数值不断变小，导致速度不断加快
     uiCtntySynSet1=const_initial_set; //同步声音的时间间隔初始值，这数值不断变小，导致鸣叫的节奏不断加
快
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
```

```
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
                ucCtntyFlag1=0; //连续加速触发模式标志位 0代表单击  1代表连续加速触发
        ucKeySec=1;     //触发1号键
     }
  }
  else if(uiKeyTimeCnt1<const_time_1s) //按住累加到1秒
  {
     uiKeyTimeCnt1++;
  }
  else  //按住累加到1秒后仍然不放手，这个时候进入有节奏的连续加速触发
  {
         uiKeyCtntyCnt1++; //按键连续触发延时计数器累加
//按住没松手，每隔一段uiCtntyTimeSet1时间按键就触发一次，而且uiCtntyTimeSet1不断减小，速度就越来越快
         if(uiKeyCtntyCnt1>uiCtntyTimeSet1)
         {
                    if(uiCtntyTimeSet1>const_min_level)
                        {
                           uiCtntyTimeSet1=uiCtntyTimeSet1-const_sub_dt; //uiCtntyTimeSet1不断减小，速
度就越来越快
                        }
                        else
                        {
                            uiCtntyTimeSet1=const_last_min_set; //uiCtntyTimeSet1不断减小，到达一个极限
值
                        }
             uiKeyCtntyCnt1=0;
                         ucCtntyFlag1=1;  //进入连续加速触发模式
             ucKeySec=1;     //触发1号键
         }
                uiSynCtntyCnt1++; //蜂鸣器连续触发延时计数器累加
//按住没松手，每隔一段uiCtntySynSet1时间蜂鸣器就触发一次，而且uiCtntySynSet1不断减小，鸣叫的节奏就越来越
快
                if(uiSynCtntyCnt1>uiCtntySynSet1)
                {
               uiCtntySynSet1=uiCtntySynSet1-const_sub_dt; //uiCtntySynSet1不断减小，鸣叫的节奏就越来越
快
                        if(uiCtntySynSet1<const_syn_min_level)
                        {
                            uiCtntySynSet1=const_syn_min_level; //uiCtntySynSet1不断减小，达到一个极限
值
                        }
                        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                        uiSynCtntyCnt1=0;

                }

  }
  if(key_sr2==1)
  {
     ucKeyLock2=0;
```

```
   uiKeyTimeCnt2=0;
   uiKeyCtntyCnt2=0;
       uiSynCtntyCnt2=0;
   uiCtntyTimeSet2=const_initial_set;
   uiCtntySynSet2=const_initial_set;
}
else if(ucKeyLock2==0)
{
   uiKeyTimeCnt2++;
   if(uiKeyTimeCnt2>const_key_time2)
   {
      uiKeyTimeCnt2=0;
      ucKeyLock2=1;
              ucCtntyFlag2=0;
      ucKeySec=2;
   }
}
else if(uiKeyTimeCnt2<const_time_1s)
{
   uiKeyTimeCnt2++;
}
else
{
       uiKeyCtntyCnt2++;
       if(uiKeyCtntyCnt2>uiCtntyTimeSet2)
       {
                   if(uiCtntyTimeSet2>const_min_level)
                       {
                          uiCtntyTimeSet2=uiCtntyTimeSet2-const_sub_dt;
                       }
                       else
                       {
                           uiCtntyTimeSet2=const_last_min_set;
                       }
           uiKeyCtntyCnt2=0;
                       ucCtntyFlag2=1;
           ucKeySec=2;
       }
                uiSynCtntyCnt2++;
                if(uiSynCtntyCnt2>uiCtntySynSet2)
                {
               uiCtntySynSet2=uiCtntySynSet2-const_sub_dt;
                       if(uiCtntySynSet2<const_syn_min_level)
                       {
                            uiCtntySynSet2=const_syn_min_level;
                       }
                       uiVoiceCnt=const_voice_short;
                       uiSynCtntyCnt2=0;

               }
```

```
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 连续加键  对应朱兆祺学习板的S1键
              uiSetNumber++; //被设置的参数连续往上加
              if(uiSetNumber>1000) //最大是1000
              {
                   uiSetNumber=1000;
              }
                        if(ucCtntyFlag1==0) //如果是在单击按键的情况下，则蜂鸣器鸣叫，否则蜂鸣器在按键
扫描key_scan里鸣叫
                        {
                    uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                        }
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 连续减键  对应朱兆祺学习板的S5键
/* 注释二:
* 在单片机的C语言编译器中，当无符号数据0减去1时，就会溢出，变成这个类型数据的最大值。
* 比如是unsigned int的0减去1就等于65535(0xffff),unsigned char的0减去1就等于255(0xff)
*/
              uiSetNumber--; //被设置的参数连续往下减
              if(uiSetNumber>1000) //最小是0.为什么这里用1000?因为0减去1就是溢出变成了65535(0xffff)
              {
                  uiSetNumber=0;
              }
                        if(ucCtntyFlag2==0)  //如果是在单击按键的情况下，则蜂鸣器鸣叫，否则蜂鸣器在按
键扫描key_scan里鸣叫
                        {
                    uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                        }
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
```

```
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
          beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  led_dr=0;  //LED灯灭
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

到目前为止，前面一共花了8节内容仔细讲解了独立按键的扫描程序，如果是矩阵键盘，我们该怎么写程序？欲知详情，请听下回分解-----矩阵键盘的单个触发。

（未完待续，下节更精彩，不要走开哦）

第十四节：矩阵键盘的单个触发。

开场白:

上一节讲了按键的加速匀速触发。这节开始讲矩阵键盘的单个触发。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。。

（2）实现功能：16个按键中，每按一个按键都能触发一次蜂鸣器发出“滴”的一声。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释一:
*  矩阵按键扫描的详细过程:
*  先输出某一列低电平，其它三列输出高电平，这个时候再分别判断输入的四行，
*  如果发现哪一行是低电平，就说明对应的某个按键被触发。依次分别输出另外三列
*  中的某一列为低电平，再分别判断输入的四行，就可以检测完16个按键。内部详细的
*  去抖动处理方法跟我前面讲的独立按键去抖动方法是一样的。
*/
  switch(ucKeyStep)
  {
     case 1:   //按键扫描输出第一列低电平
          key_dr1=0;
          key_dr2=1;
          key_dr3=1;
          key_dr4=1;
```

```
        uiKeyTimeCnt=0;  //延时计数器清零
        ucKeyStep++;     //切换到下一个运行步骤
            break;
    case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
        uiKeyTimeCnt++;
                if(uiKeyTimeCnt>1)
                {
                   uiKeyTimeCnt=0;
           ucKeyStep++;     //切换到下一个运行步骤
                }
            break;
    case 3:
        if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
        {
           ucKeyStep++;  //如果没有按键按下，切换到下一个运行步骤
           ucKeyLock=0;  //按键自锁标志清零
           uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙
        }
                else if(ucKeyLock==0)  //有按键按下，且是第一次触发
                {
                   if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                       {
                          uiKeyTimeCnt++;  //去抖动延时计数器
                              if(uiKeyTimeCnt>const_key_time)
                              {
                                 uiKeyTimeCnt=0;
                                 ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                 ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                              }

                       }
                   else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                       {
                          uiKeyTimeCnt++;  //去抖动延时计数器
                              if(uiKeyTimeCnt>const_key_time)
                              {
                                 uiKeyTimeCnt=0;
                                 ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                 ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
                              }

                       }
                   else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
                       {
                          uiKeyTimeCnt++;  //去抖动延时计数器
                              if(uiKeyTimeCnt>const_key_time)
                              {
                                 uiKeyTimeCnt=0;
```

```
                                ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
                            }

                        }
                    else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                        {
                            uiKeyTimeCnt++;  //去抖动延时计数器
                                if(uiKeyTimeCnt>const_key_time)
                                {
                                    uiKeyTimeCnt=0;
                                    ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                    ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                                }

                        }

                }
            break;
    case 4:   //按键扫描输出第二列低电平
        key_dr1=1;
        key_dr2=0;
        key_dr3=1;
        key_dr4=1;
        uiKeyTimeCnt=0;  //延时计数器清零
        ucKeyStep++;     //切换到下一个运行步骤
            break;
    case 5:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
        uiKeyTimeCnt++;
                if(uiKeyTimeCnt>1)
                {
                    uiKeyTimeCnt=0;
           ucKeyStep++;     //切换到下一个运行步骤
                }
            break;
    case 6:
        if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
        {
           ucKeyStep++;  //如果没有按键按下，切换到下一个运行步骤
           ucKeyLock=0;  //按键自锁标志清零
           uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙
        }
                else if(ucKeyLock==0)  //有按键按下，且是第一次触发
                {
                    if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                        {
                            uiKeyTimeCnt++;  //去抖动延时计数器
                                if(uiKeyTimeCnt>const_key_time)
```

```
                        {
                            uiKeyTimeCnt=0;
                            ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                            ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                        }

                    }
                else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                    {
                        uiKeyTimeCnt++;  //去抖动延时计数器
                            if(uiKeyTimeCnt>const_key_time)
                            {
                                uiKeyTimeCnt=0;
                                ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
                            }

                    }
                else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
                    {
                        uiKeyTimeCnt++;  //去抖动延时计数器
                            if(uiKeyTimeCnt>const_key_time)
                            {
                                uiKeyTimeCnt=0;
                                ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S9键
                            }

                    }
                else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                    {
                        uiKeyTimeCnt++;  //去抖动延时计数器
                            if(uiKeyTimeCnt>const_key_time)
                            {
                                uiKeyTimeCnt=0;
                                ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S13键
                            }

                    }

            }
        break;
    case 7:   //按键扫描输出第三列低电平
        key_dr1=1;
        key_dr2=1;
```

```
        key_dr3=0;
        key_dr4=1;
        uiKeyTimeCnt=0;  //延时计数器清零
        ucKeyStep++;     //切换到下一个运行步骤
            break;
    case 8:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
        uiKeyTimeCnt++;
                if(uiKeyTimeCnt>1)
                {
                   uiKeyTimeCnt=0;
           ucKeyStep++;     //切换到下一个运行步骤
                }
            break;
    case 9:
        if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
        {
           ucKeyStep++;  //如果没有按键按下，切换到下一个运行步骤
           ucKeyLock=0;  //按键自锁标志清零
           uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙
        }
                else if(ucKeyLock==0)  //有按键按下，且是第一次触发
                {
                   if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                  ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                               }

                        }
                   else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                                  ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
                               }

                        }
                   else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
```

```
                        {
                            uiKeyTimeCnt=0;
                            ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                            ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
                        }

                    }
                else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                    {
                       uiKeyTimeCnt++;  //去抖动延时计数器
                           if(uiKeyTimeCnt>const_key_time)
                           {
                            uiKeyTimeCnt=0;
                            ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                            ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                        }

                    }

            }
        break;
case 10:   //按键扫描输出第四列低电平
    key_dr1=1;
    key_dr2=1;
    key_dr3=1;
    key_dr4=0;
    uiKeyTimeCnt=0;  //延时计数器清零
    ucKeyStep++;     //切换到下一个运行步骤
        break;
case 11:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
    uiKeyTimeCnt++;
            if(uiKeyTimeCnt>1)
            {
               uiKeyTimeCnt=0;
      ucKeyStep++;     //切换到下一个运行步骤
            }
         break;
case 12:
    if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
    {
       ucKeyStep=1;  //如果没有按键按下，返回到第一步，重新开始扫描
       ucKeyLock=0;  //按键自锁标志清零
       uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙
    }
            else if(ucKeyLock==0)  //有按键按下，且是第一次触发
            {
               if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                   {
```

```
                    uiKeyTimeCnt++;  //去抖动延时计数器
                        if(uiKeyTimeCnt>const_key_time)
                        {
                           uiKeyTimeCnt=0;
                           ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                           ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                        }

                    }
                else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                    {
                       uiKeyTimeCnt++;  //去抖动延时计数器
                        if(uiKeyTimeCnt>const_key_time)
                        {
                           uiKeyTimeCnt=0;
                           ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                           ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
                        }

                    }
                else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
                    {
                       uiKeyTimeCnt++;  //去抖动延时计数器
                        if(uiKeyTimeCnt>const_key_time)
                        {
                           uiKeyTimeCnt=0;
                           ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                           ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
                        }

                    }
                else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                    {
                       uiKeyTimeCnt++;  //去抖动延时计数器
                        if(uiKeyTimeCnt>const_key_time)
                        {
                           uiKeyTimeCnt=0;
                           ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                           ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                        }

                    }

            }
        break;
```

```
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S2键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 3号键 对应朱兆祺学习板的S3键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 4:// 4号键 对应朱兆祺学习板的S4键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 5:// 5号键 对应朱兆祺学习板的S5键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 6:// 6号键 对应朱兆祺学习板的S6键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 7:// 7号键 对应朱兆祺学习板的S7键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 8:// 8号键 对应朱兆祺学习板的S8键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 9:// 9号键 对应朱兆祺学习板的S9键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 10:// 10号键 对应朱兆祺学习板的S10键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 11:// 11号键 对应朱兆祺学习板的S11键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
```

```
    case 12:// 12号键 对应朱兆祺学习板的S12键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 13:// 13号键 对应朱兆祺学习板的S13键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 14:// 14号键 对应朱兆祺学习板的S14键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 15:// 15号键 对应朱兆祺学习板的S15键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 16:// 16号键 对应朱兆祺学习板的S16键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
```

```
        }
    }
}
void initial_myself()  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

在这一节中，有的人咋看我的按键扫描代码，会觉得代码太多了。我一直认为，只要单片机容量够，代码多一点少一点并不重要，只要不影响运行效率就行。而且有时候，代码写多一点，可读性非常强，修改起来也非常方便。如果一味的追求压缩代码，就会刻意用很多循环，数组等元素，代码虽然紧凑了，但是可分离性，可改性，可读性就没那么强。我说那么多并不是因为我技术有限而不懂压缩，就找个借口敷衍大家，不信？我下一节把这节的代码压缩一下分享给大家。凡是相似度高的那部分代码都可以压缩，具体怎么压缩？欲知详情，请听下回分解-----矩阵键盘单个触发的压缩代码编程。

（未完待续，下节更精彩，不要走开哦）

第十五节：矩阵键盘单个触发的压缩代码编程。

开场白:

上一节讲了矩阵键盘的单个触发。这节要教会大家在不改变其它任何性能的情况下，把上一节的按键扫描程序压缩一下容量。经过压缩后，把原来1558个字节压缩到860个字节的程序容量。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。。

（2）实现功能：16个按键中，每按一个按键都能触发一次蜂鸣器发出“滴”的一声。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
```

```
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释一:
*  矩阵按键扫描的详细过程:
*  先输出某一列低电平，其它三列输出高电平，这个时候再分别判断输入的四行，
*  如果发现哪一行是低电平，就说明对应的某个按键被触发。依次分别输出另外三列
*  中的某一列为低电平，再分别判断输入的四行，就可以检测完16个按键。内部详细的
*  去抖动处理方法跟我前面讲的独立按键去抖动方法是一样的。
*/
  switch(ucKeyStep)
  {
     case 1:   //按键扫描输出第ucRowRecord列低电平
               if(ucRowRecord==1)  //第一列输出低电平
                   {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==2)  //第二列输出低电平
                   {
              key_dr1=1;
              key_dr2=0;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==3)  //第三列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=0;
              key_dr4=1;
                   }
               else   //第四列输出低电平
```

```
            {
        key_dr1=1;
        key_dr2=1;
        key_dr3=1;
        key_dr4=0;
            }
    uiKeyTimeCnt=0;  //延时计数器清零
    ucKeyStep++;     //切换到下一个运行步骤
        break;
case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
    uiKeyTimeCnt++;
            if(uiKeyTimeCnt>1)
            {
               uiKeyTimeCnt=0;
        ucKeyStep++;     //切换到下一个运行步骤
            }
        break;
case 3:
    if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
    {
        ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
        ucKeyLock=0;  //按键自锁标志清零
        uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                ucRowRecord++;  //输出下一列
                if(ucRowRecord>4)
                {
                   ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                }
    }
            else if(ucKeyLock==0)  //有按键按下，且是第一次触发
            {
               if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                   {
                      uiKeyTimeCnt++;  //去抖动延时计数器
                          if(uiKeyTimeCnt>const_key_time)
                          {
                             uiKeyTimeCnt=0;
                             ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                  if(ucRowRecord==1)  //第一列输出低电平
                      {
                                 ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                      }
                  else if(ucRowRecord==2)  //第二列输出低电平
                      {
                                 ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                      }
                  else if(ucRowRecord==3)  //第三列输出低电平
                      {
```

```
                    ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                }
            else   //第四列输出低电平
                {
                    ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                }
                    }

            }
        else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
            {
                uiKeyTimeCnt++;  //去抖动延时计数器
                    if(uiKeyTimeCnt>const_key_time)
                    {
                        uiKeyTimeCnt=0;
                        ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
            if(ucRowRecord==1)  //第一列输出低电平
                {
                    ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
                }
            else if(ucRowRecord==2)  //第二列输出低电平
                {
                    ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
                }
            else if(ucRowRecord==3)  //第三列输出低电平
                {
                    ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
                }
            else   //第四列输出低电平
                {
                    ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
                }
                    }

            }
        else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
            {
                uiKeyTimeCnt++;  //去抖动延时计数器
                    if(uiKeyTimeCnt>const_key_time)
                    {
                        uiKeyTimeCnt=0;
                        ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
            if(ucRowRecord==1)  //第一列输出低电平
                {
                    ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
                }
            else if(ucRowRecord==2)  //第二列输出低电平
                {
```

```
                                ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S10键
                            }
                        else if(ucRowRecord==3)  //第三列输出低电平
                            {
                                ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
                            }
                        else   //第四列输出低电平
                            {
                                ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
                            }
                                }

                      }
                    else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                        if(ucRowRecord==1)  //第一列输出低电平
                            {
                                ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                            }
                        else if(ucRowRecord==2)  //第二列输出低电平
                            {
                                ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                            }
                        else if(ucRowRecord==3)  //第三列输出低电平
                            {
                                ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                            }
                        else   //第四列输出低电平
                            {
                                ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                            }
                                }

                      }

                }
            break;
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
```

```
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 2:// 2号键 对应朱兆祺学习板的S2键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 3号键 对应朱兆祺学习板的S3键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 4号键 对应朱兆祺学习板的S4键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 5:// 5号键 对应朱兆祺学习板的S5键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 6:// 6号键 对应朱兆祺学习板的S6键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 7:// 7号键 对应朱兆祺学习板的S7键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 8:// 8号键 对应朱兆祺学习板的S8键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 9:// 9号键 对应朱兆祺学习板的S9键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 10:// 10号键 对应朱兆祺学习板的S10键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 11:// 11号键 对应朱兆祺学习板的S11键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 12:// 12号键 对应朱兆祺学习板的S12键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 13:// 13号键 对应朱兆祺学习板的S13键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
```

```
            break;
    case 14:// 14号键 对应朱兆祺学习板的S14键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 15:// 15号键 对应朱兆祺学习板的S15键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 16:// 16号键 对应朱兆祺学习板的S16键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
```

```
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

已经花了两节讲矩阵键盘的单个触发程序。那么，矩阵键盘可不可以实现类似独立按键的组合按键功能？当然可以，但是也有一些附加限制条件。欲知详情，请听下回分解-----矩阵键盘的组合按键触发。

（未完待续，下节更精彩，不要走开哦）

第十六节: 矩阵键盘的组合按键触发。

开场白:

上一节讲了矩阵键盘单个触发的压缩代码编程。这节讲矩阵键盘的组合按键触发。要教会大家三个知识点:

第一点: 如何把矩阵键盘翻译成独立按盘的处理方式。然后按独立按键的方式来实现组合按键的功能。

第二点: 要提醒大家在设计矩阵键盘时，很容易犯的一个错误。任意两个组合按键不能处于同一行，否则触发性能大打折扣。在做产品的时候，硬件电路设计中，除了四路行输入的要加上拉电阻，四路列输出也应该串入一个470欧左右的限流电阻，否则当同一行的两个按键同时按下时，很容易烧坏单片机IO口。为什么？大家仔细想想原因。因为如果没有限流电阻，同一行的两个按键同时按下时，在某一瞬间，输出的两路高低电平将会直接短接在一起，引起短路。在朱兆祺的学习板中，S1至S4是同一行，S5至S8是同一行，S9至S12是同一行，S13至S16是同一行。

第三点: 在鸿哥矩阵键盘的组合按键处理程序中，组合按键的去抖动延时const_key_time_comb千万不能等于单击按键的去抖动延时const_key_time，否则组合按键会覆盖单击按键的触发。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。

（2）实现功能: 16个按键中，每按一个按键都能触发一次蜂鸣器发出“滴”的一声。在同时按下S1和S16按键时，将会点亮一个LED灯。在同时按下S4和S13按键时，将会熄灭一个LED灯。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
*  注意: 组合按键的去抖动延时const_key_time_comb千万不能等于单击按键
*  的去抖动延时const_key_time，否则组合按键会覆盖单击按键的触发。
*/
#define const_key_time  12    //按键去抖动延时的时间
#define const_key_time_comb  14    //组合按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
/* 注释二:
*  注意: 任意两个组合按键不能处于同一行，否则触发性能大打折扣。
*  在做产品的时候，硬件电路设计中，除了四路行输入的要加上拉电阻，
*  四路列输出也应该串入一个470欧左右的限流电阻，否则当同一行的两个
*  按键同时按下时，很容易烧坏单片机IO口。为什么？大家仔细想想原因。
*  因为如果没有限流电阻，同一行的两个按键同时按下时，在某一瞬间，
*  输出的两路高低电平将会直接短接在一起，引起短路。
```

```
*  在朱兆祺的学习板中，S1至S4是同一行，S5至S8是同一行，S9至S12是同一行，S13至S16是同一行。
*/
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit led_dr=P3^5;  //LED灯的输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt[16]=0; //16个按键去抖动延时计数器
unsigned char ucKeyLock[16]=0; //16个按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt_01_16=0; //S1和S16组合按键去抖动延时计数器
unsigned char ucKeyLock_01_16=0; //S1和S16组合按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt_04_13=0; //S4和S13组合按键去抖动延时计数器
unsigned char ucKeyLock_04_13=0; //S4和S13组合按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned int  uiKeyStatus=0xffff;  //此变量每一位代表一个按键的状态，共16个按键。1代表没有被按下，0代表
被按下。
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
/* 注释三:
*  第一步: 先把16个按键翻译成独立按键。
*  第二步: 再按独立按键的去抖动方式进行按键识别。
*  第三步: 本程序把矩阵键盘翻译成独立按键的处理方式后，大家可以按独立按键的方式
*          来实现组合按键，双击，长按和短按，按住连续触发等功能。
*          我本人不再详细介绍这方面的内容。有兴趣的朋友，可以参考一下我前面章节讲的独立按键。
*/
  switch(ucKeyStep)
  {
     case 1:   //把16个按键的状态快速记录在uiKeyStatus变量的每一位中,相当于把矩阵键盘翻译成独立按键。
               for(ucRowRecord=1;ucRowRecord<5;ucRowRecord++)
                     {
                  if(ucRowRecord==1)  //第一列输出低电平
                      {
```

```
                key_dr1=0;
                key_dr2=1;
                key_dr3=1;
                key_dr4=1;
                 //如果是薄膜按键或者走线比较长的按键，此处应该加几个空延时，等待列输出信号稳定再判断输入
的状态
                       if(key_sr1==0)
                           {
                              uiKeyStatus=uiKeyStatus&0xfffe; //对应朱兆祺学习板的S1键被按下
                           }
                       if(key_sr2==0)
                           {
                              uiKeyStatus=uiKeyStatus&0xffef; //对应朱兆祺学习板的S5键被按下
                           }
                       if(key_sr3==0)
                           {
                              uiKeyStatus=uiKeyStatus&0xfeff; //对应朱兆祺学习板的S9键被按下
                           }
                       if(key_sr4==0)
                           {
                              uiKeyStatus=uiKeyStatus&0xefff; //对应朱兆祺学习板的S13键被按下
                           }
                    }
                 else if(ucRowRecord==2)  //第二列输出低电平
                     {
                key_dr1=1;
                key_dr2=0;
                key_dr3=1;
                key_dr4=1;
                //如果是薄膜按键或者走线比较长的按键，此处应该加几个空延时，等待列输出信号稳定再判断输入
的状态
                        if(key_sr1==0)
                              {
                              uiKeyStatus=uiKeyStatus&0xfffd; //对应朱兆祺学习板的S2键被按下
                             }
                        if(key_sr2==0)
                            {
                              uiKeyStatus=uiKeyStatus&0xffdf; //对应朱兆祺学习板的S6键被按下
                            }
                        if(key_sr3==0)
                            {
                               uiKeyStatus=uiKeyStatus&0xfdff; //对应朱兆祺学习板的S10键被按下
                            }
                        if(key_sr4==0)
                            {
                               uiKeyStatus=uiKeyStatus&0xdfff; //对应朱兆祺学习板的S14键被按下
                             }
                    }
                 else if(ucRowRecord==3)  //第三列输出低电平
                     {
```

```
            key_dr1=1;
            key_dr2=1;
            key_dr3=0;
            key_dr4=1;
             //如果是薄膜按键或者走线比较长的按键，此处应该加几个空延时，等待列输出信号稳定再判断输
入的状态
                    if(key_sr1==0)
                        {
                           uiKeyStatus=uiKeyStatus&0xfffb; //对应朱兆祺学习板的S3键被按下
                        }
                    if(key_sr2==0)
                        {
                           uiKeyStatus=uiKeyStatus&0xffbf; //对应朱兆祺学习板的S7键被按下
                        }
                    if(key_sr3==0)
                        {
                           uiKeyStatus=uiKeyStatus&0xfbff; //对应朱兆祺学习板的S11键被按下
                         }
                    if(key_sr4==0)
                        {
                           uiKeyStatus=uiKeyStatus&0xbfff; //对应朱兆祺学习板的S15键被按下
                        }
                }
             else   //第四列输出低电平
                 {
            key_dr1=1;
            key_dr2=1;
            key_dr3=1;
            key_dr4=0;
             //如果是薄膜按键或者走线比较长的按键，此处应该加几个空延时，等待列输出信号稳定再判断输
入的状态
                    if(key_sr1==0)
                         {
                           uiKeyStatus=uiKeyStatus&0xfff7; //对应朱兆祺学习板的S4键被按下
                        }
                    if(key_sr2==0)
                        {
                           uiKeyStatus=uiKeyStatus&0xff7f; //对应朱兆祺学习板的S8键被按下
                        }
                    if(key_sr3==0)
                        {
                           uiKeyStatus=uiKeyStatus&0xf7ff; //对应朱兆祺学习板的S12键被按下
                        }
                    if(key_sr4==0)
                        {
                           uiKeyStatus=uiKeyStatus&0x7fff; //对应朱兆祺学习板的S16键被按下
                        }
                }
        }
        ucKeyStep=2;      //切换到下一个运行步骤
```

```
            break;
    case 2:  //像独立按键一样进行去抖动和翻译。以下代码相似度很高，大家有兴趣的话还可以加for循环来压缩代码
            if((uiKeyStatus&0x0001)==0x0001)  //说明1号键没有被按下来
                {
            uiKeyTimeCnt[0]=0;
            ucKeyLock[0]=0;
                }
                else if(ucKeyLock[0]==0)
                {
                   uiKeyTimeCnt[0]++;
                       if(uiKeyTimeCnt[0]>const_key_time)
                       {
                          uiKeyTimeCnt[0]=0;
                          ucKeyLock[0]=1; //自锁按键，防止不断触发
                          ucKeySec=1;   //被触发1号键
                       }
                }
            if((uiKeyStatus&0x0002)==0x0002)  //说明2号键没有被按下来
                {
            uiKeyTimeCnt[1]=0;
            ucKeyLock[1]=0;
                }
                else if(ucKeyLock[1]==0)
                {
                   uiKeyTimeCnt[1]++;
                       if(uiKeyTimeCnt[1]>const_key_time)
                       {
                          uiKeyTimeCnt[1]=0;
                          ucKeyLock[1]=1; //自锁按键，防止不断触发
                          ucKeySec=2;   //被触发2号键
                       }
                }
            if((uiKeyStatus&0x0004)==0x0004)  //说明3号键没有被按下来
                {
            uiKeyTimeCnt[2]=0;
            ucKeyLock[2]=0;
                }
                else if(ucKeyLock[2]==0)
                {
                   uiKeyTimeCnt[2]++;
                       if(uiKeyTimeCnt[2]>const_key_time)
                       {
                          uiKeyTimeCnt[2]=0;
                          ucKeyLock[2]=1; //自锁按键，防止不断触发
                          ucKeySec=3;   //被触发3号键
                       }
                }
            if((uiKeyStatus&0x0008)==0x0008)  //说明4号键没有被按下来
                {
```

```
uiKeyTimeCnt[3]=0;
ucKeyLock[3]=0;
     }
     else if(ucKeyLock[3]==0)
     {
        uiKeyTimeCnt[3]++;
            if(uiKeyTimeCnt[3]>const_key_time)
            {
               uiKeyTimeCnt[3]=0;
               ucKeyLock[3]=1; //自锁按键，防止不断触发
               ucKeySec=4;    //被触发4号键
            }
     }
 if((uiKeyStatus&0x0010)==0x0010)  //说明5号键没有被按下来
     {
uiKeyTimeCnt[4]=0;
ucKeyLock[4]=0;
     }
     else if(ucKeyLock[4]==0)
     {
        uiKeyTimeCnt[4]++;
            if(uiKeyTimeCnt[4]>const_key_time)
            {
               uiKeyTimeCnt[4]=0;
               ucKeyLock[4]=1; //自锁按键，防止不断触发
               ucKeySec=5;    //被触发5号键
            }
     }
 if((uiKeyStatus&0x0020)==0x0020)  //说明6号键没有被按下来
     {
uiKeyTimeCnt[5]=0;
ucKeyLock[5]=0;
     }
     else if(ucKeyLock[5]==0)
     {
        uiKeyTimeCnt[5]++;
            if(uiKeyTimeCnt[5]>const_key_time)
            {
               uiKeyTimeCnt[5]=0;
               ucKeyLock[5]=1; //自锁按键，防止不断触发
               ucKeySec=6;    //被触发6号键
            }
     }
 if((uiKeyStatus&0x0040)==0x0040)  //说明7号键没有被按下来
     {
uiKeyTimeCnt[6]=0;
ucKeyLock[6]=0;
     }
     else if(ucKeyLock[6]==0)
     {
```

```
        uiKeyTimeCnt[6]++;
            if(uiKeyTimeCnt[6]>const_key_time)
            {
               uiKeyTimeCnt[6]=0;
               ucKeyLock[6]=1; //自锁按键，防止不断触发
               ucKeySec=7;   //被触发7号键
            }
     }
 if((uiKeyStatus&0x0080)==0x0080)  //说明8号键没有被按下来
     {
uiKeyTimeCnt[7]=0;
ucKeyLock[7]=0;
     }
     else if(ucKeyLock[7]==0)
     {
        uiKeyTimeCnt[7]++;
            if(uiKeyTimeCnt[7]>const_key_time)
            {
               uiKeyTimeCnt[7]=0;
               ucKeyLock[7]=1; //自锁按键，防止不断触发
               ucKeySec=8;   //被触发8号键
            }
     }
 if((uiKeyStatus&0x0100)==0x0100)  //说明9号键没有被按下来
     {
uiKeyTimeCnt[8]=0;
ucKeyLock[8]=0;
     }
     else if(ucKeyLock[8]==0)
     {
        uiKeyTimeCnt[8]++;
            if(uiKeyTimeCnt[8]>const_key_time)
            {
               uiKeyTimeCnt[8]=0;
               ucKeyLock[8]=1; //自锁按键，防止不断触发
               ucKeySec=9;   //被触发9号键
            }
     }
 if((uiKeyStatus&0x0200)==0x0200)  //说明10号键没有被按下来
     {
uiKeyTimeCnt[9]=0;
ucKeyLock[9]=0;
     }
     else if(ucKeyLock[9]==0)
     {
        uiKeyTimeCnt[9]++;
            if(uiKeyTimeCnt[9]>const_key_time)
            {
               uiKeyTimeCnt[9]=0;
               ucKeyLock[9]=1; //自锁按键，防止不断触发
```

```
                ucKeySec=10;    //被触发10号键
            }
        }
 if((uiKeyStatus&0x0400)==0x0400)  //说明11号键没有被按下来
     {
uiKeyTimeCnt[10]=0;
ucKeyLock[10]=0;
     }
     else if(ucKeyLock[10]==0)
     {
        uiKeyTimeCnt[10]++;
            if(uiKeyTimeCnt[10]>const_key_time)
            {
                uiKeyTimeCnt[10]=0;
                ucKeyLock[10]=1; //自锁按键，防止不断触发
                ucKeySec=11;    //被触发11号键
            }
     }
 if((uiKeyStatus&0x0800)==0x0800)  //说明12号键没有被按下来
     {
uiKeyTimeCnt[11]=0;
ucKeyLock[11]=0;
     }
     else if(ucKeyLock[11]==0)
     {
        uiKeyTimeCnt[11]++;
            if(uiKeyTimeCnt[11]>const_key_time)
            {
                uiKeyTimeCnt[11]=0;
                ucKeyLock[11]=1; //自锁按键，防止不断触发
                ucKeySec=12;    //被触发12号键
            }
     }
 if((uiKeyStatus&0x0800)==0x0800)  //说明12号键没有被按下来
     {
uiKeyTimeCnt[11]=0;
ucKeyLock[11]=0;
     }
     else if(ucKeyLock[11]==0)
     {
        uiKeyTimeCnt[11]++;
            if(uiKeyTimeCnt[11]>const_key_time)
            {
                uiKeyTimeCnt[11]=0;
                ucKeyLock[11]=1; //自锁按键，防止不断触发
                ucKeySec=12;    //被触发12号键
            }
     }
 if((uiKeyStatus&0x1000)==0x1000)  //说明13号键没有被按下来
     {
```

```
uiKeyTimeCnt[12]=0;
ucKeyLock[12]=0;
     }
     else if(ucKeyLock[12]==0)
     {
        uiKeyTimeCnt[12]++;
            if(uiKeyTimeCnt[12]>const_key_time)
            {
               uiKeyTimeCnt[12]=0;
               ucKeyLock[12]=1; //自锁按键，防止不断触发
               ucKeySec=13;    //被触发13号键
            }
     }
 if((uiKeyStatus&0x2000)==0x2000)  //说明14号键没有被按下来
     {
uiKeyTimeCnt[13]=0;
ucKeyLock[13]=0;
     }
     else if(ucKeyLock[13]==0)
     {
        uiKeyTimeCnt[13]++;
            if(uiKeyTimeCnt[13]>const_key_time)
            {
               uiKeyTimeCnt[13]=0;
               ucKeyLock[13]=1; //自锁按键，防止不断触发
               ucKeySec=14;    //被触发14号键
            }
     }
 if((uiKeyStatus&0x4000)==0x4000)  //说明15号键没有被按下来
     {
uiKeyTimeCnt[14]=0;
ucKeyLock[14]=0;
     }
     else if(ucKeyLock[14]==0)
     {
        uiKeyTimeCnt[14]++;
            if(uiKeyTimeCnt[14]>const_key_time)
            {
               uiKeyTimeCnt[14]=0;
               ucKeyLock[14]=1; //自锁按键，防止不断触发
               ucKeySec=15;    //被触发15号键
            }
     }
 if((uiKeyStatus&0x8000)==0x8000)  //说明16号键没有被按下来
     {
uiKeyTimeCnt[15]=0;
ucKeyLock[15]=0;
     }
     else if(ucKeyLock[15]==0)
     {
```

```
                uiKeyTimeCnt[15]++;
                    if(uiKeyTimeCnt[15]>const_key_time)
                    {
                       uiKeyTimeCnt[15]=0;
                       ucKeyLock[15]=1; //自锁按键，防止不断触发
                       ucKeySec=16;    //被触发16号键
                    }
                }
             if((uiKeyStatus&0x8001)==0x0000)  //S1和S16的组合键盘被按下。
                {
            if(ucKeyLock_01_16==0)
                    {
                        uiKeyTimeCnt_01_16++;
                            if(uiKeyTimeCnt_01_16>const_key_time_comb)
                            {
                               uiKeyTimeCnt_01_16=0;
                                   ucKeyLock_01_16=1;
                                   ucKeySec=17;    //被触发17号组合键
                            }

                    }
                }
                else
                {
            uiKeyTimeCnt_01_16=0; //S1和S16组合按键去抖动延时计数器
            ucKeyLock_01_16=0; //S1和S16组合按键触发后自锁的变量标志
                }
             if((uiKeyStatus&0x1008)==0x0000)  //S4和S13的组合键盘被按下。
                {
            if(ucKeyLock_04_13==0)
                    {
                        uiKeyTimeCnt_04_13++;
                            if(uiKeyTimeCnt_04_13>const_key_time_comb)
                            {
                               uiKeyTimeCnt_04_13=0;
                                   ucKeyLock_04_13=1;
                                   ucKeySec=18;    //被触发18号组合键
                            }

                    }
                }
                else
                {
            uiKeyTimeCnt_04_13=0; //S4和S13组合按键去抖动延时计数器
            ucKeyLock_04_13=0; //S4和S13组合按键触发后自锁的变量标志
                }
        uiKeyStatus=0xffff;    //及时恢复状态，方便下一次扫描
                ucKeyStep=1;  //返回到第一个运行步骤重新开始扫描
             break;
}
```

```
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S2键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 3号键 对应朱兆祺学习板的S3键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 4:// 4号键 对应朱兆祺学习板的S4键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 5:// 5号键 对应朱兆祺学习板的S5键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 6:// 6号键 对应朱兆祺学习板的S6键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 7:// 7号键 对应朱兆祺学习板的S7键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 8:// 8号键 对应朱兆祺学习板的S8键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 9:// 9号键 对应朱兆祺学习板的S9键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 10:// 10号键 对应朱兆祺学习板的S10键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 11:// 11号键 对应朱兆祺学习板的S11键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 12:// 12号键 对应朱兆祺学习板的S12键
```

```
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 13:// 13号键 对应朱兆祺学习板的S13键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 14:// 14号键 对应朱兆祺学习板的S14键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 15:// 15号键 对应朱兆祺学习板的S15键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 16:// 16号键 对应朱兆祺学习板的S16键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 17:// 17号组合键 对应朱兆祺学习板的S1和S16键的组合按键
            led_dr=1; //LED灯亮
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 18:// 18号组合键 对应朱兆祺学习板的S4和S13键的组合按键
            led_dr=0; //LED灯灭
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
```

```
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=0; //LED灯灭
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

这节讲了如何把矩阵键盘翻译成独立按键的处理方式，然后像独立按键一样实现组合按键的功能，关于矩阵按键的双击，长按和短按，按键连续触发等功能我不再详细介绍，有兴趣的朋友可以参考我前面章节讲的独立按键。在实际的项目中，按键可以控制很多外设。为了以后进一步讲按键控制外设等功能，接下来我会讲哪些新内容呢？欲知详情，请听下回分解-----两片联级74HC595驱动16个LED灯的基本驱动程序。

（未完待续，下节更精彩，不要走开哦）

第十七节：两片联级74HC595驱动16个LED灯的基本驱动程序。

开场白:

上一节讲了如何把矩阵键盘翻译成独立按键的处理方式。这节讲74HC595的驱动程序。要教会大家两个知识点:

第一点：朱兆祺的学习板是用74HC595控制LED，因此可以直接把595的OE引脚接地。如果在工控中，用来控制继电器，那么此芯片的片选脚OE不要为了省一个IO口而直接接地，否则会引起上电瞬间继电器莫名其妙地动作。为了解决这个问题，OE脚应该用一个IO口单独驱动，并且千万要记住，此IO必须接一个15K左右的上拉电阻，然后在程序刚上电运行时，先把OE置高，并且尽快把所有的74HC595输出口置低,然后再把OE置低.当然还有另外一种解决办法，就是用一个10uF的电解电容跟一个100K的下拉电阻，组成跟51单片机外围复位电路原理一样的电路，连接到OE口，这样确保上电瞬间OE口有一小段时间是处于高电平状态,在此期间，尽快通过软件把74hc595的所有输出口置低。

第二点：两个联级74HC595的工作过程：每个74HC595内部都有一个8位的寄存器，两个联级起来就有两个寄存器。ST引脚就相当于一个刷新信号引脚，当ST引脚产生一个上升沿信号时，就会把寄存器的数值输出到74HC595的输出引脚并且锁存起来，DS是数据引脚，SH是把新数据送入寄存器的时钟信号。也就是说，SH引脚负责把数据送入到寄存器里，ST引脚负责把寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。

（2）实现功能：两片联级的74HC595驱动的16个LED灯交叉闪烁。比如，先是第1,3,5,7,9,11,13,15八个灯亮，其它的灯都灭。然后再反过来，原来亮的就灭，原来灭的就亮。交替闪烁。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level 200
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker();
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
/* 注释一:
* 朱兆祺的学习板是用74HC595控制LED，因此可以直接把595的OE引脚接地。如果在工控中，用来控制继电器，
* 那么此芯片的片选脚OE不要为了省一个IO口而直接接地，否则会引起上电瞬间继电器莫名其妙地动作。
* 为了解决这个问题，OE脚应该用一个IO口单独驱动，并且千万要记住，此IO必须接一个15K左右的
* 上拉电阻，然后在程序刚上电运行时，先把OE置高，并且尽快把所有的74HC595输出口置低,然后再把OE置低.
* 当然还有另外一种解决办法，就是用一个10uF的电解电容跟一个100K的下拉电阻，组成跟51单片机外围复位电路原理
* 一样的电路，连接到OE口，这样确保上电瞬间OE口有一小段时间是处于高电平状态,在此 期间，
* 尽快通过软件把74hc595的所有输出口置低。
*/
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucLedStep=0; //步骤变量
unsigned int  uiTimeCnt=0; //统计定时中断次数的延时计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker();
   }
}
/* 注释二:
* 两个联级74HC595的工作过程:
* 每个74HC595内部都有一个8位的寄存器，两个联级起来就有两个寄存器。ST引脚就相当于一个刷新
* 信号引脚，当ST引脚产生一个上升沿信号时，就会把寄存器的数值输出到74HC595的输出引脚并且锁存起来，
* DS是数据引脚，SH是把新数据送入寄存器的时钟信号。也就是说，SH引脚负责把数据送入到寄存器里，ST引脚
* 负责把寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来。
*/
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
```

```
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(15);
        hc595_sh_dr=1;
        delay_short(15);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(15);
        hc595_sh_dr=1;
        delay_short(15);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void led_flicker() ////第三区 LED闪烁应用程序
{
  switch(ucLedStep)
  {
     case 0:
          if(uiTimeCnt>=const_time_level) //时间到
          {
              uiTimeCnt=0; //时间计数器清零
              hc595_drive(0x55,0x55);
              ucLedStep=1; //切换到下一个步骤
          }
          break;
     case 1:
          if(uiTimeCnt>=const_time_level) //时间到
          {
              uiTimeCnt=0; //时间计数器清零
              hc595_drive(0xaa,0xaa);
              ucLedStep=0; //返回到上一个步骤
          }
          break;

    }
}
void T0_time() interrupt 1
```

```
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt++;  //累加定时中断的次数，
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

这节讲了74HC595的驱动程序，它是一次控制16个LED同时亮灭的，在实际中应用不太方便，如果我们想要像单片机IO口直接控制LED那样方便，我们该怎么编写程序呢？欲知详情，请听下回分解-----把74HC595驱动程序翻译成类似单片机IO口直接驱动的方式。

（未完待续，下节更精彩，不要走开哦）

第十八节：把74HC595驱动程序翻译成类似单片机IO口直接驱动的方式。

开场白:

上一节讲了74HC595的驱动程序。为了更加方便操作74HC595输出的每个IO状态，这节讲如何把74HC595驱动程序翻译成类似单片机IO口直接驱动的方式。要教会大家两个知识点:
第一点: 如何灵活运用与和非的运算符来实现位的操作。
第二点: 如何灵活运用一个更新变量来实现静态刷新输出或者静态刷新显示的功能。
具体内容，请看源代码讲解。
（1）硬件平台: 基于朱兆祺51单片机学习板。
（2）实现功能: 两片联级的74HC595驱动的16个LED灯交叉闪烁。比如，先是第1,3,5,7,9,11,13,15八个灯亮，其它的灯都灭。然后再反过来，原来亮的就灭，原来灭的就亮。交替闪烁。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level 200
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker();
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep=0; //步骤变量
unsigned int  uiTimeCnt=0; //统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
    {
```

```
      led_flicker();
          led_update();  //LED更新函数
   }
}
/* 注释一:
* 把74HC595驱动程序翻译成类似单片机IO口直接驱动方式的过程。
* 每次更新LED输出，记得都要把ucLed_update置1表示更新。
*/
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x02;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfd;
           }
       if(ucLed_dr3==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x04;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfb;
           }
       if(ucLed_dr4==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x08;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xf7;
           }
       if(ucLed_dr5==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x10;
           }
           else
```

```
    {
       ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
```

```
            }
        if(ucLed_dr12==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x08;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xf7;
            }
        if(ucLed_dr13==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x10;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xef;
            }
        if(ucLed_dr14==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x20;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xdf;
            }
        if(ucLed_dr15==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x40;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);   //74HC595底层驱动函数
    }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
```

```
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void led_flicker() ////第三区 LED闪烁应用程序
{
  switch(ucLedStep)
  {
     case 0:
           if(uiTimeCnt>=const_time_level) //时间到
           {
               uiTimeCnt=0; //时间计数器清零
               ucLed_dr1=1;  //每个变量都代表一个LED灯的状态
               ucLed_dr2=0;
               ucLed_dr3=1;
               ucLed_dr4=0;
               ucLed_dr5=1;
               ucLed_dr6=0;
               ucLed_dr7=1;
               ucLed_dr8=0;
               ucLed_dr9=1;
               ucLed_dr10=0;
               ucLed_dr11=1;
               ucLed_dr12=0;
```

```
                ucLed_dr13=1;
                ucLed_dr14=0;
                ucLed_dr15=1;
                ucLed_dr16=0;
                ucLed_update=1;  //更新显示
                ucLedStep=1; //切换到下一个步骤
            }
            break;
      case 1:
            if(uiTimeCnt>=const_time_level) //时间到
            {
                uiTimeCnt=0; //时间计数器清零
                ucLed_dr1=0;  //每个变量都代表一个LED灯的状态
                ucLed_dr2=1;
                ucLed_dr3=0;
                ucLed_dr4=1;
                ucLed_dr5=0;
                ucLed_dr6=1;
                ucLed_dr7=0;
                ucLed_dr8=1;
                ucLed_dr9=0;
                ucLed_dr10=1;
                ucLed_dr11=0;
                ucLed_dr12=1;
                ucLed_dr13=0;
                ucLed_dr14=1;
                ucLed_dr15=0;
                ucLed_dr16=1;
                ucLed_update=1;  //更新显示
                ucLedStep=0; //返回到上一个步骤
            }
            break;

    }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt++;  //累加定时中断的次数，
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
```

```
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

这节讲了把74HC595驱动程序翻译成类似单片机IO口直接驱动的方式，接下来，我们该如何来运用这种驱动方式实现跑马灯的程序？欲知详情，请听下回分解-----依次逐个点亮LED之后，再依次逐个熄灭LED的跑马灯程序。

（未完待续，下节更精彩，不要走开哦）

第十九节: 依次逐个点亮LED之后，再依次逐个熄灭LED的跑马灯程序。

开场白:

上一节讲了把74HC595驱动程序翻译成类似单片机IO口直接驱动的方式。这节在上一节的驱动程序基础上，开始讲跑马灯程序。我的跑马灯程序看似简单而且重复，其实蕴含着鸿哥的大智慧。它是基于鸿哥的switch状态机思想，领略到了它的简单和精髓，以后任何所谓复杂的工程项目，都不再复杂。要教会大家一个知识点: 通过本跑马灯程序，加深理解鸿哥所有实战项目中switch状态机的思想精髓。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。

（2）实现功能: 第1个至第8个LED灯，先依次逐个亮，再依次逐个灭。第9至第16个LED灯一直灭。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level_01_08  200  //第1个至第8个LED跑马灯的速度延时时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker_01_08(); // 第1个至第8个LED的跑马灯程序，逐个亮，逐个灭.
```

```
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_01_08=0; //第1个至第8个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_01_08=0; //第1个至第8个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker_01_08(); // 第1个至第8个LED的跑马灯程序，逐个亮，逐个灭.
          led_update();  //LED更新函数
   }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
```

```
    }
if(ucLed_dr2==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x02;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfd;
    }
if(ucLed_dr3==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x04;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfb;
    }
if(ucLed_dr4==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x08;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
```

```
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
if(ucLed_dr13==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x10;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xef;
    }
if(ucLed_dr14==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x20;
```

```
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xdf;
            }
        if(ucLed_dr15==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x40;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
    }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
```

```
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
/* 注释一:
* 以下程序，看似简单而且重复，其实蕴含着鸿哥的大智慧。
* 它是基于鸿哥的switch状态机思想，领略到了它的简单和精髓，
* 以后任何所谓复杂的工程项目，都不再复杂。
*/
void led_flicker_01_08() //第1个至第8个LED的跑马灯程序，逐个亮，逐个灭.
{
  switch(ucLedStep_01_08)
  {
     case 0:
          if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
          {
              uiTimeCnt_01_08=0; //时间计数器清零
              ucLed_dr1=1;  //第1个亮
              ucLed_update=1;  //更新显示
              ucLedStep_01_08=1; //切换到下一个步骤
          }
          break;
     case 1:
          if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
          {
              uiTimeCnt_01_08=0; //时间计数器清零
              ucLed_dr2=1;  //第2个亮
              ucLed_update=1;  //更新显示
              ucLedStep_01_08=2; //切换到下一个步骤
          }
          break;
     case 2:
          if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
          {
              uiTimeCnt_01_08=0; //时间计数器清零
              ucLed_dr3=1;  //第3个亮
              ucLed_update=1;  //更新显示
              ucLedStep_01_08=3; //切换到下一个步骤
          }
          break;
     case 3:
          if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
          {
              uiTimeCnt_01_08=0; //时间计数器清零
```

```
            ucLed_dr4=1;  //第4个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=4; //切换到下一个步骤
        }
        break;
case 4:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr5=1;  //第5个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=5; //切换到下一个步骤
        }
        break;
case 5:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr6=1;  //第6个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=6; //切换到下一个步骤
        }
        break;
case 6:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr7=1;  //第7个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=7; //切换到下一个步骤
        }
        break;
case 7:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr8=1;  //第8个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=8; //切换到下一个步骤
        }
        break;
case 8:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr8=0;  //第8个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=9; //切换到下一个步骤
        }
        break;
```

```
case 9:
      if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
      {
          uiTimeCnt_01_08=0; //时间计数器清零
          ucLed_dr7=0;  //第7个灭
          ucLed_update=1;  //更新显示
          ucLedStep_01_08=10; //切换到下一个步骤
      }
      break;
case 10:
      if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
      {
          uiTimeCnt_01_08=0; //时间计数器清零
          ucLed_dr6=0;  //第6个灭
          ucLed_update=1;  //更新显示
          ucLedStep_01_08=11; //切换到下一个步骤
      }
      break;
case 11:
      if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
      {
          uiTimeCnt_01_08=0; //时间计数器清零
          ucLed_dr5=0;  //第5个灭
          ucLed_update=1;  //更新显示
          ucLedStep_01_08=12; //切换到下一个步骤
      }
      break;
case 12:
      if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
      {
          uiTimeCnt_01_08=0; //时间计数器清零
          ucLed_dr4=0;  //第4个灭
          ucLed_update=1;  //更新显示
          ucLedStep_01_08=13; //切换到下一个步骤
      }
      break;
case 13:
      if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
      {
          uiTimeCnt_01_08=0; //时间计数器清零
          ucLed_dr3=0;  //第3个灭
          ucLed_update=1;  //更新显示
          ucLedStep_01_08=14; //切换到下一个步骤
      }
      break;
case 14:
      if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
      {
          uiTimeCnt_01_08=0; //时间计数器清零
          ucLed_dr2=0;  //第2个灭
```

```
                ucLed_update=1;  //更新显示
                ucLedStep_01_08=15; //切换到下一个步骤
            }
            break;
      case 15:
            if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
            {
                uiTimeCnt_01_08=0; //时间计数器清零
                ucLed_dr1=0;  //第1个灭
                ucLed_update=1;  //更新显示
                ucLedStep_01_08=0; //返回到最开始处，重新开始新的一次循环。
            }
            break;
   }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_01_08<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt_01_08++;  //累加定时中断的次数，
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  TMOD=0x01;  //设置定时器0为工作方式1
```

```
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

这节讲了在第1个至第8个LED灯中，先依次逐个亮再依次逐个灭的跑马灯程序。下一节我们略作修改，继续做跑马灯的程序，要求在第9个至第16个LED灯中，依次逐个亮灯并且每次只能亮一个灯（其它的都灭），依次循环，我们该如何编写程序？欲知详情，请听下回分解-----依次逐个亮灯并且每次只能亮一个灯的跑马灯程序。

（未完待续，下节更精彩，不要走开哦）

第二十节：依次逐个亮灯并且每次只能亮一个灯的跑马灯程序。

开场白:

上一节讲了先依次逐个亮再依次逐个灭的跑马灯程序。这一节在上一节的基础上，略作修改，继续讲跑马灯程序。我的跑马灯程序看似简单而且重复，其实蕴含着鸿哥的大智慧。它是基于鸿哥的switch状态机思想，领略到了它的简单和精髓，以后任何所谓复杂的工程项目，都不再复杂。要教会大家一个知识点：通过本跑马灯程序，加深理解鸿哥所有实战项目中switch状态机的思想精髓。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。

（2）实现功能：第9个至第16个LED灯，依次逐个亮灯并且每次只能亮一个灯。第1至第8个LED灯一直灭。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level_09_16  300  //第9个至第16个LED跑马灯的速度延时时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker_09_16(); // 第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
```

```
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_09_16=0; //第9个至第16个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_09_16=0; //第9个至第16个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker_09_16(); // 第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
          led_update();  //LED更新函数
   }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x02;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfd;
           }
       if(ucLed_dr3==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x04;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfb;
           }
       if(ucLed_dr4==1)
           {
```

```
        ucLedStatus08_01=ucLedStatus08_01|0x08;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
```

```
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
if(ucLed_dr13==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x10;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xef;
    }
if(ucLed_dr14==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x20;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xdf;
    }
if(ucLed_dr15==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x40;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xbf;
    }
if(ucLed_dr16==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x80;
    }
    else
    {
```

```
            ucLedStatus16_09=ucLedStatus16_09&0x7f;
          }
       hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
   }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
         delay_short(15);
         hc595_sh_dr=1;
         delay_short(15);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucLedStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
         delay_short(15);
         hc595_sh_dr=1;
         delay_short(15);
         ucTempData=ucTempData<<1;
   }
   hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(15);
   hc595_st_dr=1;
   delay_short(15);
   hc595_sh_dr=0;    //拉低，抗干扰就增强
   hc595_st_dr=0;
   hc595_ds_dr=0;
}
/* 注释一:
* 以下程序，看似简单而且重复，其实蕴含着鸿哥的大智慧。
* 它是基于鸿哥的switch状态机思想，领略到了它的简单和精髓，
* 以后任何所谓复杂的工程项目，都不再复杂。
*/
void led_flicker_09_16() //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
{
  switch(ucLedStep_09_16)
  {
```

```
case 0:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
          ucLed_dr16=0;  //第16个灭
          ucLed_dr9=1;  //第9个亮
          ucLed_update=1;  //更新显示
          ucLedStep_09_16=1; //切换到下一个步骤
      }
      break;
case 1:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
          ucLed_dr9=0;  //第9个灭
          ucLed_dr10=1;  //第10个亮
          ucLed_update=1;  //更新显示
          ucLedStep_09_16=2; //切换到下一个步骤
      }
      break;
case 2:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
          ucLed_dr10=0;  //第10个灭
          ucLed_dr11=1;  //第11个亮
          ucLed_update=1;  //更新显示
          ucLedStep_09_16=3; //切换到下一个步骤
      }
      break;
case 3:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
          ucLed_dr11=0;  //第11个灭
          ucLed_dr12=1;  //第12个亮
          ucLed_update=1;  //更新显示
          ucLedStep_09_16=4; //切换到下一个步骤
      }
      break;
case 4:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
          ucLed_dr12=0;  //第12个灭
          ucLed_dr13=1;  //第13个亮
          ucLed_update=1;  //更新显示
          ucLedStep_09_16=5; //切换到下一个步骤
      }
      break;
```

```
    case 5:
          if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
              ucLed_dr13=0;  //第13个灭
              ucLed_dr14=1;  //第14个亮
              ucLed_update=1;  //更新显示
              ucLedStep_09_16=6; //切换到下一个步骤
          }
          break;
    case 6:
          if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
              ucLed_dr14=0;  //第14个灭
              ucLed_dr15=1;  //第15个亮
              ucLed_update=1;  //更新显示
              ucLedStep_09_16=7; //切换到下一个步骤
          }
          break;
    case 7:
          if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
              ucLed_dr15=0;  //第15个灭
              ucLed_dr16=1;  //第16个亮
              ucLed_update=1;  //更新显示
              ucLedStep_09_16=0; //返回到开始处，重新开始新的一次循环
          }
          break;

  }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_09_16<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt_09_16++;  //累加定时中断的次数，
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
```

```
    ;   //一个分号相当于执行一条空语句
  }
}
void delay_long(unsigned int uiDelayLong)
{
  unsigned int i;
  unsigned int j;
  for(i=0;i<uiDelayLong;i++)
  {
     for(j=0;j<500;j++)  //内嵌循环的空指令数量
         {
            ; //一个分号相当于执行一条空语句
         }
  }
}
void initial_myself()  //第一区 初始化单片机
{
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

上一节和这一节讲了两种不同的跑马灯程序，如果要让这两种不同的跑马灯程序都能各自独立运行，就涉及到多任务并行处理的程序框架。没错，下一节就讲多任务并行处理这方面的知识，欲知详情，请听下回分解-----多任务并行处理两路跑马灯。

（未完待续，下节更精彩，不要走开哦）

第二十一节: 多任务并行处理两路跑马灯。

开场白:

上一节讲了依次逐个亮灯并且每次只能亮一个灯的跑马灯程序。这一节要结合前面两节的内容，实现多任务并行处理两路跑马灯。要教会大家一个知识点: 利用鸿哥的switch状态机思想，实现多任务并行处理的程序。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。

（2）实现功能:

第一路独立运行的任务是: 第1个至第8个LED灯，先依次逐个亮，再依次逐个灭。

第二路独立运行的任务是: 第9个至第16个LED灯，依次逐个亮灯并且每次只能亮一个灯。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level_01_08  200  //第1个至第8个LED跑马灯的速度延时时间
#define const_time_level_09_16  300  //第9个至第16个LED跑马灯的速度延时时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker_01_08(); //第一路独立运行的任务 第1个至第8个LED的跑马灯程序，逐个亮，逐个灭.
```

```
void led_flicker_09_16(); //第二路独立运行的任务 第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_01_08=0; //第1个至第8个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_01_08=0; //第1个至第8个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStep_09_16=0; //第9个至第16个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_09_16=0; //第9个至第16个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker_01_08(); //第一路独立运行的任务 第1个至第8个LED的跑马灯程序，逐个亮，逐个灭.
      led_flicker_09_16(); //第二路独立运行的任务 第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个
.
          led_update();  //LED更新函数
   }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
```

```
        ucLedStatus08_01=ucLedStatus08_01|0x01;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xfe;
    }
if(ucLed_dr2==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x02;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xfd;
    }
if(ucLed_dr3==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x04;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xfb;
    }
if(ucLed_dr4==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x08;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
```

```
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
if(ucLed_dr13==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x10;
    }
    else
    {
```

```
            ucLedStatus16_09=ucLedStatus16_09&0xef;
        }
    if(ucLed_dr14==1)
        {
            ucLedStatus16_09=ucLedStatus16_09|0x20;
        }
        else
        {
            ucLedStatus16_09=ucLedStatus16_09&0xdf;
        }
    if(ucLed_dr15==1)
        {
            ucLedStatus16_09=ucLedStatus16_09|0x40;
        }
        else
        {
            ucLedStatus16_09=ucLedStatus16_09&0xbf;
        }
    if(ucLed_dr16==1)
        {
            ucLedStatus16_09=ucLedStatus16_09|0x80;
        }
        else
        {
            ucLedStatus16_09=ucLedStatus16_09&0x7f;
        }
    hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
  }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
  unsigned char i;
  unsigned char ucTempData;
  hc595_sh_dr=0;
  hc595_st_dr=0;
  ucTempData=ucLedStatusTemp16_09;  //先送高8位
  for(i=0;i<8;i++)
  {
      if(ucTempData>=0x80)hc595_ds_dr=1;
      else hc595_ds_dr=0;
      hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
      delay_short(15);
      hc595_sh_dr=1;
      delay_short(15);
      ucTempData=ucTempData<<1;
  }
  ucTempData=ucLedStatusTemp08_01;  //再先送低8位
  for(i=0;i<8;i++)
  {
      if(ucTempData>=0x80)hc595_ds_dr=1;
```

```
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(15);
        hc595_sh_dr=1;
        delay_short(15);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
/* 注释一:
* 以下程序，看似简单而且重复，其实蕴含着鸿哥的大智慧。
* 它是基于鸿哥的switch状态机思想，领略到了它的简单和精髓，
* 以后任何所谓复杂的工程项目，都不再复杂。
*/
void led_flicker_01_08() //第一路独立运行的任务 第1个至第8个LED的跑马灯程序，逐个亮，逐个灭.
{
  switch(ucLedStep_01_08)
  {
     case 0:
           if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
           {
               uiTimeCnt_01_08=0; //时间计数器清零
               ucLed_dr1=1;  //第1个亮
               ucLed_update=1;  //更新显示
               ucLedStep_01_08=1; //切换到下一个步骤
           }
           break;
     case 1:
           if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
           {
               uiTimeCnt_01_08=0; //时间计数器清零
               ucLed_dr2=1;  //第2个亮
               ucLed_update=1;  //更新显示
               ucLedStep_01_08=2; //切换到下一个步骤
           }
           break;
     case 2:
           if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
           {
               uiTimeCnt_01_08=0; //时间计数器清零
               ucLed_dr3=1;  //第3个亮
               ucLed_update=1;  //更新显示
               ucLedStep_01_08=3; //切换到下一个步骤
           }
```

```
        break;
    case 3:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr4=1;  //第4个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=4; //切换到下一个步骤
        }
        break;
    case 4:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr5=1;  //第5个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=5; //切换到下一个步骤
        }
        break;
    case 5:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr6=1;  //第6个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=6; //切换到下一个步骤
        }
        break;
    case 6:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr7=1;  //第7个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=7; //切换到下一个步骤
        }
        break;
    case 7:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr8=1;  //第8个亮
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=8; //切换到下一个步骤
        }
        break;
    case 8:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
```

```
            ucLed_dr8=0;  //第8个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=9; //切换到下一个步骤
        }
        break;
case 9:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr7=0;  //第7个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=10; //切换到下一个步骤
        }
        break;
case 10:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr6=0;  //第6个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=11; //切换到下一个步骤
        }
        break;
case 11:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr5=0;  //第5个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=12; //切换到下一个步骤
        }
        break;
case 12:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr4=0;  //第4个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=13; //切换到下一个步骤
        }
        break;
case 13:
        if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
        {
            uiTimeCnt_01_08=0; //时间计数器清零
            ucLed_dr3=0;  //第3个灭
            ucLed_update=1;  //更新显示
            ucLedStep_01_08=14; //切换到下一个步骤
        }
        break;
```

```
    case 14:
          if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
          {
              uiTimeCnt_01_08=0; //时间计数器清零
              ucLed_dr2=0;  //第2个灭
              ucLed_update=1;  //更新显示
              ucLedStep_01_08=15; //切换到下一个步骤
          }
          break;
    case 15:
          if(uiTimeCnt_01_08>=const_time_level_01_08) //时间到
          {
              uiTimeCnt_01_08=0; //时间计数器清零
              ucLed_dr1=0;  //第1个灭
              ucLed_update=1;  //更新显示
              ucLedStep_01_08=0; //返回到最开始处，重新开始新的一次循环。
          }
          break;
  }
}
void led_flicker_09_16() //第二路独立运行的任务  第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
{
  switch(ucLedStep_09_16)
  {
    case 0:
          if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
              ucLed_dr16=0;  //第16个灭
              ucLed_dr9=1;  //第9个亮
              ucLed_update=1;  //更新显示
              ucLedStep_09_16=1; //切换到下一个步骤
          }
          break;
    case 1:
          if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
              ucLed_dr9=0;  //第9个灭
              ucLed_dr10=1;  //第10个亮
              ucLed_update=1;  //更新显示
              ucLedStep_09_16=2; //切换到下一个步骤
          }
          break;
    case 2:
          if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
              ucLed_dr10=0;  //第10个灭
              ucLed_dr11=1;  //第11个亮
```

```
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=3; //切换到下一个步骤
        }
        break;
case 3:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
            ucLed_dr11=0;  //第11个灭
            ucLed_dr12=1;  //第12个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=4; //切换到下一个步骤
        }
        break;
case 4:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
            ucLed_dr12=0;  //第12个灭
            ucLed_dr13=1;  //第13个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=5; //切换到下一个步骤
        }
        break;
case 5:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
            ucLed_dr13=0;  //第13个灭
            ucLed_dr14=1;  //第14个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=6; //切换到下一个步骤
        }
        break;
case 6:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
            ucLed_dr14=0;  //第14个灭
            ucLed_dr15=1;  //第15个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=7; //切换到下一个步骤
        }
        break;
case 7:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
            ucLed_dr15=0;  //第15个灭
            ucLed_dr16=1;  //第16个亮
```

```
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=0; //返回到开始处，重新开始新的一次循环
            }
            break;

    }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_01_08<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt_01_08++;  //累加定时中断的次数，
  }
  if(uiTimeCnt_09_16<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt_09_16++;  //累加定时中断的次数，
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
```

```
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

总结陈词:

这一节讲了多任务并行处理两路跑马灯的程序，从下一节开始，将会在跑马灯的基础上，新加入按键这个元素。如何把按键跟跑马灯的任务有效的关联起来，欲知详情，请听下回分解-----独立按键控制跑马灯的方向。

（未完待续，下节更精彩，不要走开哦）

第二十二节：独立按键控制跑马灯的方向。

开场白:

上一节讲了多任务并行处理两路跑马灯的程序。这一节要教会大家一个知识点：如何通过一个中间变量把按键跟跑马灯的任务有效的关联起来。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。用矩阵键盘中的S1键作为改变方向的独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能:

第1个至第8个LED灯一直不亮。在第9个至第16个LED灯，依次逐个亮灯并且每次只能亮一个灯。按一次独立按键S1，将会更改跑马灯的运动方向。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_level_09_16  300  //第9个至第16个LED跑马灯的速度延时时间
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
```

```
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_09_16=0; //第9个至第16个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_09_16=0; //第9个至第16个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedDirFlag=0;   //方向变量，把按键与跑马灯关联起来的核心变量,0代表正方向，1代表反方向
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
          led_update();  //LED更新函数
      key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
```

```
    case 1:// 改变跑马灯方向的按键 对应朱兆祺学习板的S1键
          if(ucLedDirFlag==0) //通过中间变量改变跑马灯的方向
                  {
                     ucLedDirFlag=1;
                  }
                  else
                  {
                           ucLedDirFlag=0;
                  }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

  }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x02;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfd;
           }
       if(ucLed_dr3==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x04;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfb;
           }
       if(ucLed_dr4==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x08;
           }
           else
           {
```

```
        ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
```

```
    if(ucLed_dr11==1)
        {
           ucLedStatus16_09=ucLedStatus16_09|0x04;
        }
        else
        {
           ucLedStatus16_09=ucLedStatus16_09&0xfb;
        }
    if(ucLed_dr12==1)
        {
           ucLedStatus16_09=ucLedStatus16_09|0x08;
        }
        else
        {
           ucLedStatus16_09=ucLedStatus16_09&0xf7;
        }
    if(ucLed_dr13==1)
        {
           ucLedStatus16_09=ucLedStatus16_09|0x10;
        }
        else
        {
           ucLedStatus16_09=ucLedStatus16_09&0xef;
        }
    if(ucLed_dr14==1)
        {
           ucLedStatus16_09=ucLedStatus16_09|0x20;
        }
        else
        {
           ucLedStatus16_09=ucLedStatus16_09&0xdf;
        }
    if(ucLed_dr15==1)
        {
           ucLedStatus16_09=ucLedStatus16_09|0x40;
        }
        else
        {
           ucLedStatus16_09=ucLedStatus16_09&0xbf;
        }
    if(ucLed_dr16==1)
        {
           ucLedStatus16_09=ucLedStatus16_09|0x80;
        }
        else
        {
           ucLedStatus16_09=ucLedStatus16_09&0x7f;
        }
    hc595_drive(ucLedStatus16_09,ucLedStatus08_01);   //74HC595底层驱动函数
}
```

```
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(15);
         hc595_sh_dr=1;
         delay_short(15);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucLedStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(15);
         hc595_sh_dr=1;
         delay_short(15);
         ucTempData=ucTempData<<1;
   }
   hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(15);
   hc595_st_dr=1;
   delay_short(15);
   hc595_sh_dr=0;    //拉低，抗干扰就增强
   hc595_st_dr=0;
   hc595_ds_dr=0;
}
/* 注释一:
* 以下程序，要学会如何通过中间变量，把按键和跑马灯的任务关联起来
*/
void led_flicker_09_16() //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
{
  switch(ucLedStep_09_16)
  {
     case 0:
           if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
```

```
            ucLed_dr16=0;  //第16个灭
            ucLed_dr9=1;  //第9个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=1; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr15=1;  //第15个亮
            ucLed_dr16=0;  //第16个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=7; //返回上一个步骤
                    }
      }
      break;
case 1:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr9=0;  //第9个灭
            ucLed_dr10=1;  //第10个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=2; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr16=1;  //第16个亮
            ucLed_dr9=0;  //第9个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=0; //返回上一个步骤
                    }
      }
      break;
case 2:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr10=0;  //第10个灭
            ucLed_dr11=1;  //第11个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=3; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr9=1;  //第9个亮
            ucLed_dr10=0;  //第10个灭
            ucLed_update=1;  //更新显示
```

```
            ucLedStep_09_16=1; //返回上一个步骤
                    }
        }
        break;
case 3:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
               ucLed_dr11=0;  //第11个灭
               ucLed_dr12=1;  //第12个亮
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=4; //切换到下一个步骤
                    }
                    else  //反方向
                    {
               ucLed_dr10=1;  //第10个亮
               ucLed_dr11=0;  //第11个灭
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=2; //返回上一个步骤
                    }
        }
        break;
case 4:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
               ucLed_dr12=0;  //第12个灭
               ucLed_dr13=1;  //第13个亮
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=5; //切换到下一个步骤
                    }
                    else  //反方向
                    {
               ucLed_dr11=1;  //第11个亮
               ucLed_dr12=0;  //第12个灭
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=3; //返回上一个步骤
                    }
        }
        break;
case 5:
        if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
```

```
            ucLed_dr13=0;  //第13个灭
            ucLed_dr14=1;  //第14个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=6; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr12=1;  //第12个亮
            ucLed_dr13=0;  //第13个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=4; //返回上一个步骤
                    }
      }
      break;
case 6:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr14=0;  //第14个灭
            ucLed_dr15=1;  //第15个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=7; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr13=1;  //第13个亮
            ucLed_dr14=0;  //第14个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=5; //返回上一个步骤
                    }
      }
      break;
case 7:
      if(uiTimeCnt_09_16>=const_time_level_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr15=0;  //第15个灭
            ucLed_dr16=1;  //第16个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=0; //返回到开始处，重新开始新的一次循环
                    }
                    else  //反方向
                    {
            ucLed_dr14=1;  //第14个亮
            ucLed_dr15=0;  //第15个灭
            ucLed_update=1;  //更新显示
```

```
                    ucLedStep_09_16=6; //返回上一个步骤
                            }
            }
            break;

    }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_09_16<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt_09_16++;  //累加定时中断的次数，
  }
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

```
void initial_myself()  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

这一节讲了独立按键控制跑马灯的方向。如果按键要控制跑马灯的速度，我们该怎么编写程序呢？欲知详情，请听下回分解-----独立按键控制跑马灯的速度。

（未完待续，下节更精彩，不要走开哦）

第二十三节：独立按键控制跑马灯的速度。

开场白:

上一节讲了独立按键控制跑马灯的方向。这一节继续要教会大家一个知识点：如何通过一个中间变量把按键跟跑马灯的速度有效关联起来。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。在上一节的基础上，增加一个加速按键和一个减速按键，用矩阵键盘中的S5键作为加速独立按键，用矩阵键盘中的S9键作为减速独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能:

在上一节的基础上，第1个至第8个LED灯一直不亮。在第9个至第16个LED灯，依次逐个亮灯并且每次只能亮一个灯。每按一次独立按键S5，速度都会加快。每按一次独立按键S9，速度都会减慢。跟上一节一样，用S1来改变方向。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
```

```
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
unsigned char ucKeySec=0;    //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucLed_dr1=0;    //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_09_16=0; //第9个至第16个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_09_16=0; //第9个至第16个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;    //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;    //代表底层74HC595输出状态的中间变量
unsigned char ucLedDirFlag=0;    //方向变量，把按键与跑马灯关联起来的核心变量,0代表正方向，1代表反方向
unsigned int  uiSetTimeLevel_09_16=300;  //速度变量，此数值越大速度越慢，此数值越小速度越快。
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
          led_update();  //LED更新函数
      key_service(); //按键服务的应用程序
   }
```

```
}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
}
void key_service() //按键服务的应用程序
```

```
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 改变跑马灯方向的按键 对应朱兆祺学习板的S1键
          if(ucLedDirFlag==0) //通过中间变量改变跑马灯的方向
                  {
                     ucLedDirFlag=1;
                  }
                  else
                  {
                           ucLedDirFlag=0;
                  }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

    case 2:// 加速按键 对应朱兆祺学习板的S5键 uiSetTimeLevel_09_16越小速度越快
          uiSetTimeLevel_09_16=uiSetTimeLevel_09_16-10;
                  if(uiSetTimeLevel_09_16<50)  //最快限定在50
                  {
                      uiSetTimeLevel_09_16=50;
                  }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 减速按键 对应朱兆祺学习板的S9键  uiSetTimeLevel_09_16越大速度越慢
          uiSetTimeLevel_09_16=uiSetTimeLevel_09_16+10;
                  if(uiSetTimeLevel_09_16>550)  //最慢限定在550
                  {
                      uiSetTimeLevel_09_16=550;
                  }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
```

```
    {
       ucLedStatus08_01=ucLedStatus08_01|0x02;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfd;
    }
if(ucLed_dr3==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x04;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfb;
    }
if(ucLed_dr4==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x08;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
```

```
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
if(ucLed_dr13==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x10;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xef;
    }
if(ucLed_dr14==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x20;
    }
    else
```

```
            {
               ucLedStatus16_09=ucLedStatus16_09&0xdf;
            }
        if(ucLed_dr15==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x40;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
    }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
```

```
    hc595_st_dr=0;   //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;     //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
/* 注释一:
* 以下程序，要学会如何通过中间变量，把按键和跑马灯的任务关联起来
*/
void led_flicker_09_16() //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
{
  switch(ucLedStep_09_16)
  {
     case 0:
           if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
                  ucLed_dr16=0;  //第16个灭
                  ucLed_dr9=1;  //第9个亮
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=1; //切换到下一个步骤
                           }
                           else  //反方向
                           {
                  ucLed_dr15=1;  //第15个亮
                  ucLed_dr16=0;  //第16个灭
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=7; //返回上一个步骤
                           }
           }
           break;
     case 1:
           if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
                  ucLed_dr9=0;  //第9个灭
                  ucLed_dr10=1;  //第10个亮
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=2; //切换到下一个步骤
                           }
                           else  //反方向
                           {
                  ucLed_dr16=1;  //第16个亮
                  ucLed_dr9=0;  //第9个灭
```

```
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=0; //返回上一个步骤
                    }
        }
        break;
case 2:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr10=0;  //第10个灭
            ucLed_dr11=1;  //第11个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=3; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr9=1;  //第9个亮
            ucLed_dr10=0;  //第10个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=1; //返回上一个步骤
                    }
        }
        break;
case 3:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr11=0;  //第11个灭
            ucLed_dr12=1;  //第12个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=4; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr10=1;  //第10个亮
            ucLed_dr11=0;  //第11个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=2; //返回上一个步骤
                    }
        }
        break;
case 4:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
```

```
                {
            ucLed_dr12=0;  //第12个灭
            ucLed_dr13=1;  //第13个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=5; //切换到下一个步骤
                }
                else  //反方向
                {
            ucLed_dr11=1;  //第11个亮
            ucLed_dr12=0;  //第12个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=3; //返回上一个步骤
                }
      }
      break;
case 5:
      if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                if(ucLedDirFlag==0)  //正方向
                {
            ucLed_dr13=0;  //第13个灭
            ucLed_dr14=1;  //第14个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=6; //切换到下一个步骤
                }
                else  //反方向
                {
            ucLed_dr12=1;  //第12个亮
            ucLed_dr13=0;  //第13个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=4; //返回上一个步骤
                }
      }
      break;
case 6:
      if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                if(ucLedDirFlag==0)  //正方向
                {
            ucLed_dr14=0;  //第14个灭
            ucLed_dr15=1;  //第15个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=7; //切换到下一个步骤
                }
                else  //反方向
                {
            ucLed_dr13=1;  //第13个亮
            ucLed_dr14=0;  //第14个灭
```

```
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=5; //返回上一个步骤
                        }
          }
          break;
     case 7:
          if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
          {
              uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
                ucLed_dr15=0;  //第15个灭
                ucLed_dr16=1;  //第16个亮
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=0; //返回到开始处，重新开始新的一次循环
                        }
                        else  //反方向
                        {
                ucLed_dr14=1;  //第14个亮
                ucLed_dr15=0;  //第15个灭
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=6; //返回上一个步骤
                        }
          }
          break;

   }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_09_16<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      uiTimeCnt_09_16++;  //累加定时中断的次数，
  }
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
```

```
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

这一节讲了独立按键控制跑马灯的速度。如果按键要控制跑马灯的启动和暂停，我们该怎么编写程序呢？欲知详情，请听下回分解-----独立按键控制跑马灯的启动和暂停。

（未完待续，下节更精彩，不要走开哦）

第二十四节：独立按键控制跑马灯的启动和暂停。

开场白:

上一节讲了独立按键控制跑马灯的速度。这一节继续要教会大家一个知识点：如何通过一个中间变量把按键跟跑马灯的启动和暂停有效关联起来。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。在上一节的基础上，增加一个启动和暂停按键，用矩阵键盘中的S13键作

为启动和暂停独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。
（2）实现功能:
在上一节的基础上，第1个至第8个LED灯一直不亮。在第9个至第16个LED灯，依次逐个亮灯并且每次只能亮一个灯。每按一次独立按键S13键，原来运行的跑马灯会暂停，原来暂停的跑马灯会运行。其它跟上一节一样，用S1来改变方向，用S5和S9来改变速度。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
```

```
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_09_16=0; //第9个至第16个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_09_16=0; //第9个至第16个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedDirFlag=0;   //方向变量，把按键与跑马灯关联起来的核心变量,0代表正方向，1代表反方向
unsigned int  uiSetTimeLevel_09_16=300;  //速度变量，此数值越大速度越慢，此数值越小速度越快。
unsigned char ucLedStartFlag=1;   //启动和暂停的变量，0代表暂停，1代表启动
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
          led_update();  //LED更新函数
      key_service(); //按键服务的应用程序
   }
}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
```

```
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;    //触发4号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 改变跑马灯方向的按键 对应朱兆祺学习板的S1键
          if(ucLedDirFlag==0) //通过中间变量改变跑马灯的方向
                  {
                     ucLedDirFlag=1;
                  }
                  else
```

```
                {
                        ucLedDirFlag=0;
                }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 2:// 加速按键 对应朱兆祺学习板的S5键 uiSetTimeLevel_09_16越小速度越快
        uiSetTimeLevel_09_16=uiSetTimeLevel_09_16-10;
                if(uiSetTimeLevel_09_16<50)  //最快限定在50
                {
                    uiSetTimeLevel_09_16=50;
                }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3:// 减速按键 对应朱兆祺学习板的S9键  uiSetTimeLevel_09_16越大速度越慢
        uiSetTimeLevel_09_16=uiSetTimeLevel_09_16+10;
                if(uiSetTimeLevel_09_16>550)  //最慢限定在550
                {
                    uiSetTimeLevel_09_16=550;
                }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 4:// 启动和暂停按键 对应朱兆祺学习板的S13键  ucLedStartFlag为0时代表暂停，为1时代表启动
            if(ucLedStartFlag==1)  //启动和暂停两种状态循环切换
                {
                   ucLedStartFlag=0;
                }
                else                    //启动和暂停两种状态循环切换
                {
                        ucLedStartFlag=1;
                }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
  }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
```

```
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfe;
    }
if(ucLed_dr2==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x02;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfd;
    }
if(ucLed_dr3==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x04;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfb;
    }
if(ucLed_dr4==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x08;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
```

```
    }
if(ucLed_dr8==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
if(ucLed_dr13==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x10;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xef;
    }
if(ucLed_dr14==1)
```

```
            {
               ucLedStatus16_09=ucLedStatus16_09|0x20;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xdf;
            }
        if(ucLed_dr15==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x40;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
    }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
```

```
        hc595_sh_dr=1;
        delay_short(15);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
/* 注释一:
* 以下程序，要学会如何通过中间变量，把按键和跑马灯的任务关联起来
*/
void led_flicker_09_16() //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
{
  if(ucLedStartFlag==1)  //此变量为1时代表启动
  {
     switch(ucLedStep_09_16)
     {
     case 0:
           if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
                  ucLed_dr16=0;  //第16个灭
                  ucLed_dr9=1;  //第9个亮
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=1; //切换到下一个步骤
                           }
                           else  //反方向
                           {
                  ucLed_dr15=1;  //第15个亮
                  ucLed_dr16=0;  //第16个灭
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=7; //返回上一个步骤
                           }
           }
           break;
     case 1:
           if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
                  ucLed_dr9=0;  //第9个灭
                  ucLed_dr10=1;  //第10个亮
                  ucLed_update=1;  //更新显示
```

```
            ucLedStep_09_16=2; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr16=1;  //第16个亮
            ucLed_dr9=0;  //第9个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=0; //返回上一个步骤
                    }
      }
      break;
case 2:
      if(uiTimeCnt_09_16>=uiSetTimeLeve1_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr10=0;  //第10个灭
            ucLed_dr11=1;  //第11个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=3; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr9=1;  //第9个亮
            ucLed_dr10=0;  //第10个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=1; //返回上一个步骤
                    }
      }
      break;
case 3:
      if(uiTimeCnt_09_16>=uiSetTimeLeve1_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr11=0;  //第11个灭
            ucLed_dr12=1;  //第12个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=4; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr10=1;  //第10个亮
            ucLed_dr11=0;  //第11个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=2; //返回上一个步骤
                    }
      }
```

```
        break;
    case 4:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
               ucLed_dr12=0;  //第12个灭
               ucLed_dr13=1;  //第13个亮
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=5; //切换到下一个步骤
                        }
                        else  //反方向
                        {
               ucLed_dr11=1;  //第11个亮
               ucLed_dr12=0;  //第12个灭
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=3; //返回上一个步骤
                        }
        }
        break;
    case 5:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
               ucLed_dr13=0;  //第13个灭
               ucLed_dr14=1;  //第14个亮
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=6; //切换到下一个步骤
                        }
                        else  //反方向
                        {
               ucLed_dr12=1;  //第12个亮
               ucLed_dr13=0;  //第13个灭
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=4; //返回上一个步骤
                        }
        }
        break;
    case 6:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
               ucLed_dr14=0;  //第14个灭
               ucLed_dr15=1;  //第15个亮
               ucLed_update=1;  //更新显示
```

```
                ucLedStep_09_16=7; //切换到下一个步骤
                        }
                        else  //反方向
                        {
                ucLed_dr13=1;  //第13个亮
                ucLed_dr14=0;  //第14个灭
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=5; //返回上一个步骤
                        }
            }
            break;
      case 7:
            if(uiTimeCnt_09_16>=uiSetTimeLeve1_09_16) //时间到
            {
                uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
                ucLed_dr15=0;  //第15个灭
                ucLed_dr16=1;  //第16个亮
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=0; //返回到开始处，重新开始新的一次循环
                        }
                        else  //反方向
                        {
                ucLed_dr14=1;  //第14个亮
                ucLed_dr15=0;  //第15个灭
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=6; //返回上一个步骤
                        }
            }
            break;

       }
    }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_09_16<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      if(ucLedStartFlag==1)  //此变量为1时代表启动
          {
         uiTimeCnt_09_16++;  //累加定时中断的次数，
          }
  }
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
```

```
      beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:
这几节循序渐进地讲了独立按键控制跑马灯各种状态的程序。
在很多实际工控项目中，经常会涉及到运动的自动控制，运动的自动控制就必然会涉及到感应器。
下一节我将会讲感应器和运动控制的程序框架，
欲知详情，请听下回分解-----用LED灯和按键来模拟工业自动化设备的运动控制。

第二十五节: 用LED灯和按键来模拟工业自动化设备的运动控制。

开场白:
前面三节讲了独立按键控制跑马灯的各种状态，这一节我们要做一个机械手控制程序，这个机械手可以左右移动，最左边有一个开关感应器，最右边也有一个开关感应器。它也可以上下移动，最下面有一个开关感应器。左右移动是通过一个气缸控制，上下移动也是通过一个气缸控制。而单片机控制气缸，本质上是通过三极管把信号放大，然后控制气缸上的电磁阀。这个系统机械手驱动部分的输出和输入信号如下:

2个输出IO口，分别控制2个气缸。对于左右移动的气缸，当IO口为0时往左边跑，当IO口为1时往右边跑。对于上下移动的气缸，当IO口为0时往上边跑，当IO口为1时往下边跑。

3个输入IO口，分别检测3个开关感应器。感应器没有被触发时，IO口检测为高电平1。被触发时，IO口检测为低电平0。

这一节继续要教会大家两个知识点:
第一点: 如何用软件进行开关感应器的抗干扰处理。
第二点: 如何用Switch语句搭建工业自动控制的程序框架。还是那句话，我们只要以Switch语句为支点，再复杂再繁琐的程序都可以轻松地编写出来。
具体内容，请看源代码讲解。
（1）硬件平台: 基于朱兆祺51单片机学习板。用矩阵键盘中的S1键作为启动独立按键，用S5按键模拟左边的开关感应器，用S9按键模拟右边的开关感应器，用S13按键模拟下边的开关感应器。记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。
（2）实现功能:

开机默认机械手在左上方的原点位置。按下启动按键后，机械手从左边开始往右边移动，当机械手移动到最右边时，机械手马上开始往下移动，最后机械手移动到最右下角的位置时，延时1秒，然后原路返回，一直返回到左上角的原点位置。注意: 启动按键必须等机械手处于左上角原点位置时，启动按键的触发才有效。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_sensor  20   //开关感应器去抖动延时的时间
#define const_1s  500  //1秒钟大概的定时中断次数
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void left_to_right();  //从左边移动到右边
void right_to_left(); //从右边返回到左边
void up_to_dowm();   //从上边移动到下边
void down_to_up();    //从下边返回到上边
void run(); //设备自动控制程序
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_update();  //LED更新函数
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
void sensor_scan(); //开关感应器软件抗干扰处理函数，放在定时中断里。
sbit hc595_sh_dr=P2^3;
sbit hc595_st_dr=P2^4;
```

```
sbit hc595_ds_dr=P2^5;
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit left_sr=P0^1; //左边的开关感应器    对应朱兆祺学习板的S5键
sbit right_sr=P0^2; //右边的开关感应器    有对应朱兆祺学习板的S9键
sbit down_sr=P0^3; //下边的开关感应器    对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned char ucLeftSr=0;  //左边感应器经过软件抗干扰处理后的状态标志
unsigned char ucRightSr=0;  //右边感应器经过软件抗干扰处理后的状态标志
unsigned char ucDownSr=0;  //下边感应器经过软件抗干扰处理后的状态标志
unsigned int  uiLeftCnt1=0;  //左边感应器软件抗干扰所需的计数器变量
unsigned int  uiLeftCnt2=0;
unsigned int  uiRightCnt1=0;  //右边感应器软件抗干扰所需的计数器变量
unsigned int  uiRightCnt2=0;
unsigned int  uiDownCnt1=0;   //下边软件抗干扰所需的计数器变量
unsigned int  uiDownCnt2=0;
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStatus16_09=0;   //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;   //代表底层74HC595输出状态的中间变量
unsigned int  uiRunTimeCnt=0;  //运动中的时间延时计数器变量
unsigned char ucRunStep=0;  //运动控制的步骤变量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      run(); //设备自动控制程序
      led_update();  //LED更新函数
```

```
      key_service(); //按键服务的应用程序
   }
}
/* 注释一:
* 开关感应器的抗干扰处理，本质上类似按键的去抖动处理。唯一的区别是:
* 按键去抖动关注的是IO口的一种状态，而开关感应器关注的是IO口的两种状态。
* 当开关感应器从原来的1状态切换到0状态之前，要进行软件滤波处理过程，一旦成功地
* 切换到0状态了，再想从0状态切换到1状态的时候，又要经过软件滤波处理过程，符合
* 条件后才能切换到1的状态。通俗的话来说，按键的去抖动从1变成0难，从0变成1容易。
* 开关感应器从1变成0难，从0变成1也难。这里所说的"难"是指要经过去抖处理。
*/
void sensor_scan() //开关感应器软件抗干扰处理函数，放在定时中断里。
{
   if(left_sr==1)  //左边感应器是高电平，说明有可能没有被接触     对应朱兆祺学习板的S5键
   {
       uiLeftCnt1=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiLeftCnt2++; //类似独立按键去抖动的软件抗干扰处理
           if(uiLeftCnt2>const_sensor)
           {
              uiLeftCnt2=0;
                  ucLeftSr=1;    //说明感应器确实没有被接触
           }
   }
   else    //左边感应器是低电平，说明有可能被接触到了
   {
       uiLeftCnt2=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiLeftCnt1++;
           if(uiLeftCnt1>const_sensor)
           {
              uiLeftCnt1=0;
                  ucLeftSr=0;    //说明感应器确实被接触到了
           }
   }
   if(right_sr==1)  //右边感应器是高电平，说明有可能没有被接触     对应朱兆祺学习板的S9键
   {
       uiRightCnt1=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiRightCnt2++; //类似独立按键去抖动的软件抗干扰处理
           if(uiRightCnt2>const_sensor)
           {
              uiRightCnt2=0;
                  ucRightSr=1;    //说明感应器确实没有被接触
           }
   }
   else    //右边感应器是低电平，说明有可能被接触到了
   {
       uiRightCnt2=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiRightCnt1++;
           if(uiRightCnt1>const_sensor)
           {
              uiRightCnt1=0;
```

```
                    ucRightSr=0;    //说明感应器确实被接触到了
                }
    }
    if(down_sr==1)  //下边感应器是高电平，说明有可能没有被接触     对应朱兆祺学习板的S13键
    {
        uiDownCnt1=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
        uiDownCnt2++; //类似独立按键去抖动的软件抗干扰处理
            if(uiDownCnt2>const_sensor)
            {
               uiDownCnt2=0;
                    ucDownSr=1;    //说明感应器确实没有被接触
            }
    }
    else    //下边感应器是低电平，说明有可能被接触到了
    {
        uiDownCnt2=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
        uiDownCnt1++;
            if(uiDownCnt1>const_sensor)
            {
               uiDownCnt1=0;
                    ucDownSr=0;    //说明感应器确实被接触到了
            }
    }
}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;   //自锁按键置位,避免一直触发
        ucKeySec=1;     //触发1号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 启动按键    对应朱兆祺学习板的S1键
         if(ucLeftSr==0)  //处于左上角原点位置
         {
            ucRunStep=1; //启动
```

```
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        }

        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

  }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x02;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfd;
           }
       if(ucLed_dr3==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x04;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfb;
           }
       if(ucLed_dr4==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x08;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xf7;
           }
       if(ucLed_dr5==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x10;
           }
           else
```

```
    {
       ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
```

```
            }
        if(ucLed_dr12==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x08;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xf7;
            }
        if(ucLed_dr13==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x10;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xef;
            }
        if(ucLed_dr14==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x20;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xdf;
            }
        if(ucLed_dr15==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x40;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
    }
}
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
```

```
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(15);
          hc595_sh_dr=1;
          delay_short(15);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(15);
    hc595_st_dr=1;
    delay_short(15);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void left_to_right()  //从左边移动到右边
{
    ucLed_dr1=1;   // 1代表左右气缸从左边移动到右边
    ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void right_to_left() //从右边返回到左边
{
    ucLed_dr1=0;   // 0代表左右气缸从右边返回到左边
    ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void up_to_down()   //从上边移动到下边
{
    ucLed_dr2=1;   // 1代表上下气缸从上边移动到下边
    ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void down_to_up()    //从下边返回到上边
{
    ucLed_dr2=0;   // 0代表上下气缸从下边返回到上边
    ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
```

```
void run() //设备自动控制程序
{
switch(ucRunStep)
{
       case 0:     //机械手处于左上角原点的位置，待命状态。此时触发启动按键ucRunStep=1，就触发后续一些列的连续动作。
            break;
       case 1:     //机械手从左边往右边移动
            left_to_right();
            ucRunStep=2;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
       case 2:     //等待机械手移动到最右边，直到触发了最右边的开关感应器。
            if(ucRightSr==0)  //右边感应器被触发
            {
               ucRunStep=3;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
       case 3:     //机械手从右上边往右下边移动，从上往下。
            up_to_down();
            ucRunStep=4;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
       case 4:     //等待机械手从右上边移动到右下边，直到触发了右下边的开关感应器。
            if(ucDownSr==0)  //右下边感应器被触发
            {
               uiRunTimeCnt=0;  //时间计数器清零，为接下来延时1秒钟做准备
               ucRunStep=5;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
       case 5:     //机械手在右下边延时1秒
            if(uiRunTimeCnt>const_1s)  //延时1秒
            {
               ucRunStep=6;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
       case 6:     //原路返回，机械手从右下边往右上边移动。
            down_to_up();
            ucRunStep=7;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
       case 7:     //原路返回，等待机械手移动到最右边的感应开关
            if(ucRightSr==0)
            {
               ucRunStep=8;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
       case 8:     //原路返回，等待机械手从右边往左边移动
            right_to_left();
            ucRunStep=9;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
       case 9:     //原路返回，等待机械手移动到最左边的感应开关，表示返回到了原点
            if(ucLeftSr==0) //返回到左上角的原点位置
```

```
            {
               ucRunStep=0;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
   }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  sensor_scan(); //开关感应器软件抗干扰处理函数
  key_scan(); //按键扫描函数
  if(uiRunTimeCnt<0xffff) //不要超过最大int类型范围
  {
     uiRunTimeCnt++; //延时计数器
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

```
void initial_myself()  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

总结陈词:

前面花了很多节内容在讲按键和跑马灯的关系，但是一直没涉及到人机界面，在大多数的实际项目中，人机界面是必不可少的。人机界面的程序框架该怎么样写？欲知详情，请听下回分解-----在主函数while循环中驱动数码管的动态扫描程序。

（未完待续，下节更精彩，不要走开哦）

第二十六节: 在主函数while循环中驱动数码管的动态扫描程序。

开场白:

上一节通过一个机械手自动控制程序展示了我在工控常用的编程框架，但是一直没涉及到人机界面，在大多数的实际项目中，人机界面是必不可少的，这一节开始讲最常用的人机界面------动态数码管的驱动。

这一节要教会大家两个知识点:

第一点: 数码管的动态驱动原理。

第二点: 如何通过编程，让数码管显示的内容转移到几个变量接口上，方便以后编写更上一层的窗口程序。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。用两片74HC595动态驱动八位共阴数码管。

（2）实现功能:

开机后显示 8765.4321 的内容，注意，其中有一个小数点。

（3）源代码讲解如下:

```
#include "REG52.H"
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
```

```
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucDisplayUpdate=1; //更新显示标志
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //不显示  序号10
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      display_drive();  //显示数码管字模的驱动函数
   }
}
/* 注释一:
* 动态驱动数码管的原理是，在八位数码管中，在任何一个瞬间，每次只显示其中一位数码管，另外的七个数码管
```

```
* 通过设置其公共位com为高电平来关闭显示，只要切换画面的速度足够快，人的视觉就分辨不出来，感觉八个数码管
* 是同时亮的。以下dig_hc595_drive(xx,yy）函数，其中第一个形参xx是驱动数码管段seg的引脚，第二个形参yy是驱
动
* 数码管公共位com的引脚。
*/
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
                   if(ucDigDot4==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xf7);
               break;
      case 5:  //显示第5位
           ucDigShowTemp=dig_table[ucDigShow5];
                   if(ucDigDot5==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xef);
               break;
```

```
        case 6:  //显示第6位
             ucDigShowTemp=dig_table[ucDigShow6];
                    if(ucDigDot6==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
             dig_hc595_drive(ucDigShowTemp,0xdf);
                 break;
        case 7:  //显示第7位
             ucDigShowTemp=dig_table[ucDigShow7];
                    if(ucDigDot7==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
             dig_hc595_drive(ucDigShowTemp,0xbf);
                 break;
        case 8:  //显示第8位
             ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
             dig_hc595_drive(ucDigShowTemp,0x7f);
                 break;
     }
     ucDisplayDriveStep++;
     if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
     {
       ucDisplayDriveStep=1;
     }
/* 注释二:
* 如果直接是单片机的IO口引脚驱动的数码管，由于驱动的速度太快，此处应该适当增加一点delay延时或者
* 用计数延时的方式来延时，目的是在八位数码管中切换到每位数码管显示的时候，都能停留一会再切换到其它
* 位的数码管界面，这样可以增加显示的效果。但是，由于朱兆祺51学习板是间接经过74HC595驱动数码管的，
* 在单片机驱动74HC595的时候，dig_hc595_drive函数本身内部需要执行很多指令，已经相当于delay延时了，
* 因此这里不再需要加delay延时函数或者计数延时。
*/
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   dig_hc595_sh_dr=0;
   dig_hc595_st_dr=0;
   ucTempData=ucDigStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
```

```
/* 注释三:
*  注意，此处的延时delay_short必须尽可能小，否则动态扫描数码管的速度就不够。
*/
        dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        dig_hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
   }
   ucTempData=ucDigStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
        if(ucTempData>=0x80)dig_hc595_ds_dr=1;
        else dig_hc595_ds_dr=0;
        dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        dig_hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
   }
   dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   dig_hc595_st_dr=1;
   delay_short(1);
   dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
   dig_hc595_st_dr=0;
   dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
   }
   ucTempData=ucLedStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
        if(ucTempData>=0x80)hc595_ds_dr=1;
```

```
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
}
void initial_peripheral() //第二区 初始化外围
{
/* 注释四:
* 让数码管显示的内容转移到以下几个变量接口上，方便以后编写更上一层的窗口程序。
* 只要更改以下对应变量的内容，就可以显示你想显示的数字。初学者应该仔细看看display_drive等函数，
* 了解来龙去脉，就可以知道本驱动程序的框架原理了。
*/
    ucDigShow8=8;  //第8位数码管要显示的内容
    ucDigShow7=7;  //第7位数码管要显示的内容
```

```
    ucDigShow6=6;   //第6位数码管要显示的内容
    ucDigShow5=5;   //第5位数码管要显示的内容
    ucDigShow4=4;   //第4位数码管要显示的内容
    ucDigShow3=3;   //第3位数码管要显示的内容
    ucDigShow2=2;   //第2位数码管要显示的内容
    ucDigShow1=1;   //第1位数码管要显示的内容
    ucDigDot8=0;
    ucDigDot7=0;
    ucDigDot6=0;
    ucDigDot5=1;   //显示第5位的小数点
    ucDigDot4=0;
    ucDigDot3=0;
    ucDigDot2=0;
    ucDigDot1=0;
}
```

总结陈词:

把本程序下载到朱兆祺51学习板上，发现显示的效果还是挺不错的。但是，本程序也有一个弱点，在一些项目中 ，主函数循环中的任务越多，就意味着在某一瞬间，每显示一位数码管停留的时间就会越久，一旦超过某个值，会严重影响显示的效果，有没有办法改善它？当然有。欲知详情，请听下回分解-----在定时中断里动态扫描数码管的程序。

（未完待续，下节更精彩，不要走开哦）

第二十七节：在定时中断里动态扫描数码管的程序。

开场白:

上一节讲了在主函数循环中动态扫描数码管的程序，但是该程序有一个隐患，在一些项目中 ，主函数循环中的任务越多，就意味着在某一瞬间，每显示一位数码管停留的时间就会越久，一旦超过某个值，会严重影响显示的效果。这一节要教会大家两个知识点:

第一个：如何把动态扫描数码管的程序放在定时中断里，彻底解决上节的显示隐患。

第二个：在定时中断里的重装初始值不能太大，否则动态扫描数码管的速度就不够。我把原来常用的初始值2000改成了500。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。用两片74HC595动态驱动八位共阴数码管。

（2）实现功能:

开机后显示 8765.4321 的内容，注意，其中有一个小数点。

（3）源代码讲解如下:

```
#include "REG52.H"
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
```

```
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucDisplayUpdate=1; //更新显示标志
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //不显示  序号10
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      ;
   }
}
/* 注释一:
* 动态驱动数码管的原理是，在八位数码管中，在任何一个瞬间，每次只显示其中一位数码管，另外的七个数码管
* 通过设置其公共位com为高电平来关闭显示，只要切换画面的速度足够快，人的视觉就分辨不出来，感觉八个数码管
* 是同时亮的。以下dig_hc595_drive(xx,yy) 函数，其中第一个形参xx是驱动数码管段seg的引脚，第二个形参yy是驱
```

```
动
* 数码管公共位com的引脚。
*/
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
                   if(ucDigDot4==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xf7);
               break;
      case 5:  //显示第5位
           ucDigShowTemp=dig_table[ucDigShow5];
                   if(ucDigDot5==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xef);
               break;
      case 6:  //显示第6位
           ucDigShowTemp=dig_table[ucDigShow6];
```

```
                    if(ucDigDot6==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                    if(ucDigDot7==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
/* 注释二:
* 如果直接是单片机的IO口引脚驱动的数码管，由于驱动的速度太快，此处应该适当增加一点delay延时或者
* 用计数延时的方式来延时，目的是在八位数码管中切换到每位数码管显示的时候，都能停留一会再切换到其它
* 位的数码管界面，这样可以增加显示的效果。但是，由于朱兆祺51学习板是间接经过74HC595驱动数码管的，
* 在单片机驱动74HC595的时候，dig_hc595_drive函数本身内部需要执行很多指令，已经相当于delay延时了，
* 因此这里不再需要加delay延时函数或者计数延时。
*/
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
/* 注释三:
*  注意，此处的延时delay_short必须尽可能小，否则动态扫描数码管的速度就不够。
```

```
*/
        dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        dig_hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)dig_hc595_ds_dr=1;
        else dig_hc595_ds_dr=0;
        dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        dig_hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
```

```
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  display_drive();  //数码管字模的驱动函数
/* 注释四:
*  注意，此处的重装初始值不能太大，否则动态扫描数码管的速度就不够。我把原来常用的2000改成了500。
*/
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
```

```
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
/* 注释五:
* 让数码管显示的内容转移到以下几个变量接口上，方便以后编写更上一层的窗口程序。
* 只要更改以下对应变量的内容，就可以显示你想显示的数字。初学者应该仔细看看display_drive等函数，
* 了解来龙去脉，就可以知道本驱动程序的框架原理了。
*/
   ucDigShow8=8;  //第8位数码管要显示的内容
   ucDigShow7=7;  //第7位数码管要显示的内容
   ucDigShow6=6;  //第6位数码管要显示的内容
   ucDigShow5=5;  //第5位数码管要显示的内容
   ucDigShow4=4;  //第4位数码管要显示的内容
   ucDigShow3=3;  //第3位数码管要显示的内容
   ucDigShow2=2;  //第2位数码管要显示的内容
   ucDigShow1=1;  //第1位数码管要显示的内容
   ucDigDot8=0;
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=1;  //显示第5位的小数点
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;     //开总中断
   ET0=1;    //允许定时中断
   TR0=1;    //启动定时中断
}
```

总结陈词:

有的朋友会质疑，很多教科书上说，定时中断函数里面的内容应该越少越好，你把动态驱动数码管的函数放在中断里面，难道不会影响其它任务的执行吗？我的回答是，大部分的小项目都不会影响，只有少数实时性要求非常高的项目会影响，而对于这类项目，我的做法是从一开始设计硬件电路板的时候，就应该放弃用动态扫描数码管的方案，而是应该选数码管专用驱动芯片来实现静态驱动。因为动态扫描数码管本来就不适合应用在实时性非常高的项目。

前面这两节都讲了数码管的驱动程序，要在此基础上，做一些项目中经常遇到的界面应用，我们该怎么写程序？欲知详情，请听下回分解-----数码管通过切换窗口来设置参数。

（未完待续，下节更精彩，不要走开哦）

第二十八节：数码管通过切换窗口来设置参数。

开场白:

上一节讲了数码管的驱动程序，这节在上节的基础上，通过按键切换不同的窗口来设置不同的参数。

这一节要教会大家三个知识点:

第一个：鸿哥首次提出的“一二级菜单显示理论”：凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。

第二个：如何通过一个窗口变量来把按键，数码管，被设置的参数关联起来。

第三个：需要特别注意，在显示被设置参数时，应该先分解出每一位，然后再把分解出来的数据过渡到显示缓冲变量里。

具体内容，请看源代码讲解。
（1）硬件平台：基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换窗口按键对应S9键
（2）实现功能：
通过按键设置4个不同的参数。
一共有4个窗口。每个窗口显示一个参数。
第8,7,6,5位数码管显示当前窗口，P-1代表第1个窗口，P-2代表第2个窗口，P-3代表第3个窗口，P-4代表第1个窗口。
第4,3,2,1位数码管显示当前窗口被设置的参数。范围是从0到9999。
有三个按键。一个是加按键，按下此按键会依次增加当前窗口的参数。一个是减按键，按下此按键会依次减少当前窗口的参数。一个是切换窗口按键，按下此按键会依次循环切换不同的窗口。
（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
```

```
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd4Update=0; //窗口4更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
```

```
   initial_peripheral();
   while(1)
   {
     key_service(); //按键服务的应用程序
         display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
*/
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示P--1窗口的数据
            if(ucWd1Update==1)  //窗口1要全部更新显示
   {
               ucWd1Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=12;  //第8位数码管显示P
               ucDigShow7=11;  //第7位数码管显示-
               ucDigShow6=1;   //第6位数码管显示1
               ucDigShow5=10;  //第5位数码管显示无
/* 注释二:
* 此处为什么要多加4个中间过渡变量ucTemp? 是因为uiSetData1分解数据的时候
* 需要进行除法和求余数的运算，就会用到好多条指令，就会耗掉一点时间，类似延时
* 了一会。我们的定时器每隔一段时间都会产生中断，然后在中断里驱动数码管显示，
* 当uiSetData1还没完全分解出4位有效数据时，这个时候来的定时中断，就有可能导致
* 显示的数据瞬间产生不完整，影响显示效果。因此，为了把需要显示的数据过渡最快，
* 所以采取了先分解，再过渡显示的方法。
*/
             //先分解数据
                      ucTemp4=uiSetData1/1000;
                      ucTemp3=uiSetData1%1000/100;
                      ucTemp2=uiSetData1%100/10;
                      ucTemp1=uiSetData1%10;

                         //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
               ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
           }
           break;
        case 2:  //显示P--2窗口的数据
           if(ucWd2Update==1)  //窗口2要全部更新显示
   {
```

```
            ucWd2Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=2;  //第6位数码管显示2
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData2/1000;     //分解数据
                    ucTemp3=uiSetData2%1000/100;
                    ucTemp2=uiSetData2%100/10;
                    ucTemp1=uiSetData2%10;
            ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
            ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
         break;
    case 3:  //显示P--3窗口的数据
        if(ucWd3Update==1)  //窗口3要全部更新显示
{
            ucWd3Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=3;  //第6位数码管显示3
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData3/1000;     //分解数据
                    ucTemp3=uiSetData3%1000/100;
                    ucTemp2=uiSetData3%100/10;
                    ucTemp1=uiSetData3%10;
            ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
            ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
        break;
    case 4:  //显示P--4窗口的数据
        if(ucWd4Update==1)  //窗口4要全部更新显示
{
            ucWd4Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=4;  //第6位数码管显示4
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData4/1000;     //分解数据
                    ucTemp3=uiSetData4%1000/100;
                    ucTemp2=uiSetData4%100/10;
                    ucTemp1=uiSetData4%10;
            ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
            ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
```

```
            break;
          }

}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
```

```
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
                  {
                     case 1:
                  uiSetData1++;
                                  if(uiSetData1>9999) //最大值是9999
                                  {
                                     uiSetData1=9999;
                                  }
                           ucWd1Update=1;  //窗口1更新显示
                              break;
                     case 2:
                  uiSetData2++;
                                  if(uiSetData2>9999) //最大值是9999
                                  {
                                     uiSetData2=9999;
                                  }
                           ucWd2Update=1;  //窗口2更新显示
                              break;
                     case 3:
                  uiSetData3++;
                                  if(uiSetData3>9999) //最大值是9999
                                  {
                                     uiSetData3=9999;
                                  }
                           ucWd3Update=1;  //窗口3更新显示
                              break;
                     case 4:
                  uiSetData4++;
                                  if(uiSetData4>9999) //最大值是9999
                                  {
                                     uiSetData4=9999;
                                  }
                           ucWd4Update=1;  //窗口4更新显示
                              break;
                  }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

    case 2:// 减按键 对应朱兆祺学习板的S5键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
                  {
                     case 1:
```

```
                uiSetData1--;
/* 注释三:
* 单片机C编译有一个特点，当一个无符号类型的数据0减去1时，就会溢出反而变成这个类型数据的最大值
* 对于int类型的数据，最大值肯定比9999大，因此下面的临界点用if(uiSetData1>9999)来判断。
*/
                                if(uiSetData1>9999)
                                {
                                   uiSetData1=0;  //最小值是0
                                }
                            ucWd1Update=1;  //窗口1更新显示
                               break;
                      case 2:
                uiSetData2--;
                                if(uiSetData2>9999)
                                {
                                   uiSetData2=0;  //最小值是0
                                }
                            ucWd2Update=1;  //窗口2更新显示
                               break;
                      case 3:
                uiSetData3--;
                                if(uiSetData3>9999)
                                {
                                   uiSetData3=0;  //最小值是0
                                }
                            ucWd3Update=1;  //窗口3更新显示
                               break;
                      case 4:
                uiSetData4--;
                                if(uiSetData4>9999)
                                {
                                   uiSetData4=0;  //最小值是0
                                }
                            ucWd4Update=1;  //窗口4更新显示
                               break;
                   }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
       case 3:// 切换窗口按键 对应朱兆祺学习板的S9键
            ucWd++;  //切换窗口
                   if(ucWd>4)
                   {
                     ucWd=1;
                   }
            switch(ucWd)  //在不同的窗口下，在不同的窗口下，更新显示不同的窗口
                   {
                      case 1:
                            ucWd1Update=1;  //窗口1更新显示
                               break;
```

```
                    case 2:
                          ucWd2Update=1;  //窗口2更新显示
                             break;
                    case 3:
                          ucWd3Update=1;  //窗口3更新显示
                             break;
                    case 4:
                          ucWd4Update=1;  //窗口4更新显示
                             break;
                 }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

  }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                  if(ucDigDot1==1)
                  {
                     ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                  }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                  if(ucDigDot2==1)
                  {
                     ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                  }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                  if(ucDigDot3==1)
                  {
                     ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                  }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
                  if(ucDigDot4==1)
                  {
                     ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
```

```
                }
            dig_hc595_drive(ucDigShowTemp,0xf7);
                break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
```

```
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
```

```
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
```

```
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释四:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

总结陈词:

这节在第4,3,2,1位显示设置的参数时，还有一点小瑕疵。比如设置参数等于56时，实际显示的是“0056”，也就是高位为0的如果不显示，效果才会更好。我们要把高位为0的去掉不显示，该怎么改程序呢？欲知详情，请听下回分解-----数码管通过切换窗口来设置参数，并且不显示为0的高位。

（未完待续，下节更精彩，不要走开哦）

第二十九节: 数码管通过切换窗口来设置参数，并且不显示为0的高位。

开场白:

上一节在第4,3,2,1位显示设置的参数时，还有一点小瑕疵。比如设置参数等于56时，实际显示的是“0056”，也就是高位为0的如果不显示，效果才会更好。

这一节要教会大家两个知识点:

第一个: 在上一节display_service()函数里略作修改，把高位为0的去掉不显示。

第二个: 加深熟悉鸿哥首次提出的“一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变

量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
具体内容，请看源代码讲解。
（1）硬件平台：基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换窗口按键对应S9键
（2）实现功能：
通过按键设置4个不同的参数。
一共有4个窗口。每个窗口显示一个参数。
第8,7,6,5位数码管显示当前窗口，P-1代表第1个窗口，P-2代表第2个窗口，P-3代表第3个窗口，P-4代表第1个窗口。
第4,3,2,1位数码管显示当前窗口被设置的参数。范围是从0到9999。
有三个按键。一个是加按键，按下此按键会依次增加当前窗口的参数。一个是减按键，按下此按键会依次减少当前窗口的参数。一个是切换窗口按键，按下此按键会依次循环切换不同的窗口。
并且要求被设置的数据不显示为0的高位。比如参数是12时，不能显示“0012”，只能第4,3位不显示，第2,1位显示“12”。
（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
```

```
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd4Update=0; //窗口4更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
```

```
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
     key_service(); //按键服务的应用程序
         display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
*/
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示P--1窗口的数据
            if(ucWd1Update==1)  //窗口1要全部更新显示
   {
              ucWd1Update=0;  //及时清零标志，避免一直进来扫描
              ucDigShow8=12;  //第8位数码管显示P
              ucDigShow7=11;  //第7位数码管显示-
              ucDigShow6=1;   //第6位数码管显示1
              ucDigShow5=10;  //第5位数码管显示无
             //先分解数据
                      ucTemp4=uiSetData1/1000;
                      ucTemp3=uiSetData1%1000/100;
                      ucTemp2=uiSetData1%100/10;
                      ucTemp1=uiSetData1%10;

                         //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
/* 注释二:
* 就是在这里略作修改，把高位为0的去掉不显示。
*/
              if(uiSetData1<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
              else
                        {
                 ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
              if(uiSetData1<100)
                        {
```

```
                ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
                ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
                if(uiSetData1<10)
                        {
                ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
                ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                }
                ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
            }
            break;
        case 2:  //显示P--2窗口的数据
            if(ucWd2Update==1)  //窗口2要全部更新显示
{
                ucWd2Update=0;  //及时清零标志，避免一直进来扫描
                ucDigShow8=12;  //第8位数码管显示P
                ucDigShow7=11;  //第7位数码管显示-
                ucDigShow6=2;  //第6位数码管显示2
                ucDigShow5=10;   //第5位数码管显示无
                        ucTemp4=uiSetData2/1000;     //分解数据
                        ucTemp3=uiSetData2%1000/100;
                        ucTemp2=uiSetData2%100/10;
                        ucTemp1=uiSetData2%10;
                if(uiSetData2<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
                else
                        {
                ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
                if(uiSetData2<100)
                        {
                ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
                ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
                if(uiSetData2<10)
                        {
                ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
```

```
                    {
                ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
              }
              ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
 }
           break;
      case 3:  //显示P--3窗口的数据
          if(ucWd3Update==1)  //窗口3要全部更新显示
{
              ucWd3Update=0;  //及时清零标志，避免一直进来扫描
              ucDigShow8=12;  //第8位数码管显示P
              ucDigShow7=11;  //第7位数码管显示-
              ucDigShow6=3;  //第6位数码管显示3
              ucDigShow5=10;   //第5位数码管显示无
                      ucTemp4=uiSetData3/1000;      //分解数据
                      ucTemp3=uiSetData3%1000/100;
                      ucTemp2=uiSetData3%100/10;
                      ucTemp1=uiSetData3%10;
              if(uiSetData3<1000)
                      {
                         ucDigShow4=10;  //如果小于1000，千位显示无
                      }
              else
                      {
                ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                      }
              if(uiSetData3<100)
                      {
                ucDigShow3=10;  //如果小于100，百位显示无
                      }
                      else
                      {
                ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                      }
              if(uiSetData3<10)
                      {
                ucDigShow2=10;  //如果小于10，十位显示无
                      }
                      else
                      {
                ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
              }
              ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
          break;
      case 4:  //显示P--4窗口的数据
          if(ucWd4Update==1)  //窗口4要全部更新显示
{
              ucWd4Update=0;  //及时清零标志，避免一直进来扫描
              ucDigShow8=12;  //第8位数码管显示P
```

```
                ucDigShow7=11;  //第7位数码管显示-
                ucDigShow6=4;  //第6位数码管显示4
                ucDigShow5=10;   //第5位数码管显示无
                        ucTemp4=uiSetData4/1000;     //分解数据
                        ucTemp3=uiSetData4%1000/100;
                        ucTemp2=uiSetData4%100/10;
                        ucTemp1=uiSetData4%10;
                if(uiSetData4<1000)
                            {
                               ucDigShow4=10;  //如果小于1000，千位显示无
                            }
                else
                            {
                   ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                            }
                if(uiSetData4<100)
                            {
                   ucDigShow3=10;  //如果小于100，百位显示无
                            }
                            else
                            {
                   ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                            }
                if(uiSetData4<10)
                            {
                   ucDigShow2=10;  //如果小于10，十位显示无
                            }
                            else
                            {
                   ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                }
                ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
    }
              break;
           }

}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
```

```
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
      }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
                  {
                     case 1:
                  uiSetData1++;
                                 if(uiSetData1>9999) //最大值是9999
                                 {
                                    uiSetData1=9999;
                                 }
                           ucWd1Update=1;  //窗口1更新显示
                              break;
```

```
                case 2:
            uiSetData2++;
                            if(uiSetData2>9999) //最大值是9999
                            {
                               uiSetData2=9999;
                            }
                    ucWd2Update=1;  //窗口2更新显示
                       break;
                case 3:
            uiSetData3++;
                            if(uiSetData3>9999) //最大值是9999
                            {
                               uiSetData3=9999;
                            }
                    ucWd3Update=1;  //窗口3更新显示
                       break;
                case 4:
            uiSetData4++;
                            if(uiSetData4>9999) //最大值是9999
                            {
                               uiSetData4=9999;
                            }
                    ucWd4Update=1;  //窗口4更新显示
                       break;
            }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
                {
                case 1:
            uiSetData1--;
                            if(uiSetData1>9999)
                            {
                               uiSetData1=0;  //最小值是0
                            }
                    ucWd1Update=1;  //窗口1更新显示
                       break;
                case 2:
            uiSetData2--;
                            if(uiSetData2>9999)
                            {
                               uiSetData2=0;  //最小值是0
                            }
                    ucWd2Update=1;  //窗口2更新显示
                       break;
                case 3:
            uiSetData3--;
```

```
                                if(uiSetData3>9999)
                                {
                                   uiSetData3=0;  //最小值是0
                                }
                            ucWd3Update=1;  //窗口3更新显示
                               break;
                        case 4:
                      uiSetData4--;
                                if(uiSetData4>9999)
                                {
                                   uiSetData4=0;  //最小值是0
                                }
                            ucWd4Update=1;  //窗口4更新显示
                               break;
                      }
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
              break;
     case 3:// 切换窗口按键 对应朱兆祺学习板的S9键
              ucWd++;  //切换窗口
                      if(ucWd>4)
                      {
                        ucWd=1;
                      }
              switch(ucWd)  //在不同的窗口下，在不同的窗口下，更新显示不同的窗口
                      {
                        case 1:
                              ucWd1Update=1;  //窗口1更新显示
                                 break;
                        case 2:
                              ucWd2Update=1;  //窗口2更新显示
                                 break;
                        case 3:
                              ucWd3Update=1;  //窗口3更新显示
                                 break;
                        case 4:
                              ucWd4Update=1;  //窗口4更新显示
                                 break;
                      }
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
              ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
              break;

   }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
```

```
case 1:  //显示第1位
     ucDigShowTemp=dig_table[ucDigShow1];
             if(ucDigDot1==1)
             {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
     dig_hc595_drive(ucDigShowTemp,0xfe);
         break;
case 2:  //显示第2位
     ucDigShowTemp=dig_table[ucDigShow2];
             if(ucDigDot2==1)
             {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
     dig_hc595_drive(ucDigShowTemp,0xfd);
         break;
case 3:  //显示第3位
     ucDigShowTemp=dig_table[ucDigShow3];
             if(ucDigDot3==1)
             {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
     dig_hc595_drive(ucDigShowTemp,0xfb);
         break;
case 4:  //显示第4位
     ucDigShowTemp=dig_table[ucDigShow4];
             if(ucDigDot4==1)
             {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
     dig_hc595_drive(ucDigShowTemp,0xf7);
         break;
case 5:  //显示第5位
     ucDigShowTemp=dig_table[ucDigShow5];
             if(ucDigDot5==1)
             {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
     dig_hc595_drive(ucDigShowTemp,0xef);
         break;
case 6:  //显示第6位
     ucDigShowTemp=dig_table[ucDigShow6];
             if(ucDigDot6==1)
             {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
             }
     dig_hc595_drive(ucDigShowTemp,0xdf);
         break;
case 7:  //显示第7位
     ucDigShowTemp=dig_table[ucDigShow7];
```

```
                    if(ucDigDot7==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
       case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
```

```
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
```

```
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
```

```
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
   ucDigDot8=0;    //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

总结陈词:

数码管通过切换窗口来设置参数，这里的窗口类似于一级菜单，在一级菜单下，还可以分解出二级菜单。一级菜单的特点是整屏数码管的显示内容全部都改变，而二级菜单的特点是只改变其中一部分数码管的内容。二级菜单的程序怎么编写？欲知详情，请听下回分解-----数码管通过闪烁来设置数据。

（未完待续，下节更精彩，不要走开哦）

第三十节：数码管通过闪烁来设置数据。

开场白:

上一节讲了一级菜单，这一节要教会大家两个知识点:

第一个: 二级菜单的程序的程序框架。

第二个: 继续加深熟悉鸿哥首次提出的“一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换“光标闪烁”按键对应S9键

（2）实现功能:

通过按键设置4个不同的参数。

只有1个窗口。这个窗口显示4个参数。

第8,7位数码管显示第1个参数。第6,5位数码管显示第2个参数。第4,3位数码管显示第3个参数。第2,1位数码管显示第4个参数。每个参数的范围是从0到99。

有三个按键。一个是“光标闪烁”按键，依次按下此按键，每两位数码管会依次处于闪烁的状态，哪两位数码管处于闪烁状态时，此时按加键或者减键就可以设置当前选中的参数。依次按下“光标闪烁”按键，数码管会在以下5种状态中循环: 只有第8,7位数码管闪烁---只有第6,5位数码管闪烁---只有第4,3位数码管闪烁---只有第2,1位数码管闪烁---所有的数码管都不闪烁。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_dpy_time_half  200  //数码管闪烁时间的半值
#define const_dpy_time_all   400  //数码管闪烁时间的全值 一定要比const_dpy_time_half 大
void initial_myself();
void initial_peripheral();
```

```
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
```

```
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucPart=0;//本程序的核心变量，局部显示变量。类似于二级菜单的变量。代表显示不同的局部。
unsigned char ucWd1Part1Update=0;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=0; //在窗口1中，局部2的更新显示标志
unsigned char ucWd1Part3Update=0; //在窗口1中，局部3的更新显示标志
unsigned char ucWd1Part4Update=0; //在窗口1中，局部4的更新显示标志
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
unsigned int  uiDpyTimeCnt=0;  //数码管的闪烁计时器,放在定时中断里不断累加
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
     key_service(); //按键服务的应用程序
         display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
```

```
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
*/
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示窗口1的数据
                        if(ucWd1Part1Update==1)  //仅仅参数1局部更新
                        {
                           ucWd1Part1Update=0;   //及时清零标志，避免一直进来扫描
                ucTemp8=uiSetData1/10;  //第1个参数
                ucTemp7=uiSetData1%10;
                           if(uiSetData1<10)
                           {
                              ucDigShow8=10;
                           }
                           else
                           {
                              ucDigShow8=ucTemp8;
                           }
                           ucDigShow7=ucTemp7;
                        }
                        if(ucWd1Part2Update==1)  //仅仅参数2局部更新
                        {
                           ucWd1Part2Update=0;  //及时清零标志，避免一直进来扫描
                ucTemp6=uiSetData2/10;  //第2个参数
                ucTemp5=uiSetData2%10;
                           if(uiSetData2<10)
                           {
                              ucDigShow6=10;
                           }
                           else
                           {
                              ucDigShow6=ucTemp6;
                           }
                           ucDigShow5=ucTemp5;
                        }
                        if(ucWd1Part3Update==1)  //仅仅参数3局部更新
                        {
                           ucWd1Part3Update=0;  //及时清零标志，避免一直进来扫描
                ucTemp4=uiSetData3/10;  //第3个参数
                ucTemp3=uiSetData3%10;
                           if(uiSetData3<10)
                           {
                              ucDigShow4=10;
                           }
                           else
                           {
```

```
                        ucDigShow4=ucTemp4;
                    }
                    ucDigShow3=ucTemp3;
                }
                if(ucWd1Part4Update==1)  //仅仅参数4局部更新
                {
                    ucWd1Part4Update=0;   //及时清零标志，避免一直进来扫描
            ucTemp2=uiSetData4/10;  //第4个参数
            ucTemp1=uiSetData4%10;
                        if(uiSetData4<10)
                        {
                            ucDigShow2=10;
                        }
                        else
                        {
                            ucDigShow2=ucTemp2;
                        }
                        ucDigShow1=ucTemp1;
                }
/* 注释二:
* 必须注意局部更新和全部更新的编写顺序，局部更新应该写在全部更新之前，
* 当局部更新和全部更新同时发生时，这样就能保证到全部更新的优先响应。
*/
            if(ucWd1Update==1)  //窗口1要全部更新显示
                        {
            ucWd1Update=0;  //及时清零标志，避免一直进来扫描
            ucTemp8=uiSetData1/10;  //第1个参数
            ucTemp7=uiSetData1%10;
            ucTemp6=uiSetData2/10;  //第2个参数
            ucTemp5=uiSetData2%10;
            ucTemp4=uiSetData3/10;  //第3个参数
            ucTemp3=uiSetData3%10;
            ucTemp2=uiSetData4/10;  //第4个参数
            ucTemp1=uiSetData4%10;
                        if(uiSetData1<10)
                        {
                            ucDigShow8=10;
                        }
                        else
                        {
                            ucDigShow8=ucTemp8;
                        }
                        ucDigShow7=ucTemp7;
                        if(uiSetData2<10)
                        {
                            ucDigShow6=10;
                        }
                        else
                        {
                            ucDigShow6=ucTemp6;
```

```
                    }
                    ucDigShow5=ucTemp5;
                    if(uiSetData3<10)
                    {
                       ucDigShow4=10;
                    }
                    else
                    {
                       ucDigShow4=ucTemp4;
                    }
                    ucDigShow3=ucTemp3;
                    if(uiSetData4<10)
                    {
                       ucDigShow2=10;
                    }
                    else
                    {
                       ucDigShow2=ucTemp2;
                    }
                    ucDigShow1=ucTemp1;
        }
                  //数码管闪烁
        switch(ucPart)  //根据局部变量的值，使对应的参数产生闪烁的动态效果。
                  {
                     case 0:  //4个参数都不闪烁
                          break;
                     case 1:  //第1个参数闪烁
                          if(uiDpyTimeCnt==const_dpy_time_half)
                                  {
                             if(uiSetData1<10)        //数码管显示内容
                             {
                                ucDigShow8=10;
                             }
                             else
                             {
                                ucDigShow8=ucTemp8;
                             }
                             ucDigShow7=ucTemp7;
                                  }
                          else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                                  {
                                     uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                             ucDigShow8=10;   //数码管显示空，什么都不显示
                             ucDigShow7=10;
                                  }
                          break;
                     case 2:   //第2个参数闪烁
                          if(uiDpyTimeCnt==const_dpy_time_half)
                                  {
```

```
                    if(uiSetData2<10)        //数码管显示内容
                    {
                       ucDigShow6=10;
                    }
                    else
                    {
                       ucDigShow6=ucTemp6;
                    }
                    ucDigShow5=ucTemp5;
                         }
                   else if(uiDpyTimeCnt>const_dpy_time_all)  //const_dpy_time_all一定要比
const_dpy_time_half 大
                         {
                            uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                    ucDigShow6=10;   //数码管显示空，什么都不显示
                    ucDigShow5=10;
                         }
                   break;
              case 3:  //第3个参数闪烁
                   if(uiDpyTimeCnt==const_dpy_time_half)
                         {
                    if(uiSetData3<10)        //数码管显示内容
                    {
                       ucDigShow4=10;
                    }
                    else
                    {
                       ucDigShow4=ucTemp4;
                    }
                    ucDigShow3=ucTemp3;
                         }
                   else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                         {
                            uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                    ucDigShow4=10;   //数码管显示空，什么都不显示
                    ucDigShow3=10;
                         }
                   break;
              case 4:  //第4个参数闪烁
                   if(uiDpyTimeCnt==const_dpy_time_half)
                         {
                    if(uiSetData4<10)        //数码管显示内容
                    {
                       ucDigShow2=10;
                    }
                    else
                    {
                       ucDigShow2=ucTemp2;
                    }
```

```
                                    ucDigShow1=ucTemp1;
                                        }
                                else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                                        {
                                            uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                    ucDigShow2=10;    //数码管显示空，什么都不显示
                                    ucDigShow1=10;
                                        }
                                break;
                        }
            break;

     }

}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
```

```
    ucKeyLock3=0; //按键自锁标志清零
    uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
    uiKeyTimeCnt3++; //累加定时中断次数
    if(uiKeyTimeCnt3>const_key_time3)
    {
      uiKeyTimeCnt3=0;
      ucKeyLock3=1;  //自锁按键置位,避免一直触发
      ucKeySec=3;    //触发3号键
    }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                   switch(ucPart)  //在窗口1下，根据不同的局部闪烁位置来设置不同的参数
                                  {
                                     case 0:
                                              break;
                                     case 1:
                             uiSetData1++;
                             if(uiSetData1>99) //最大值是99
                             {
                                 uiSetData1=99;
                             }
                                                 ucWd1Part1Update=1; //局部更新显示参数1
                                              break;
                                     case 2:
                             uiSetData2++;
                             if(uiSetData2>99) //最大值是99
                             {
                                 uiSetData2=99;
                             }
                                                 ucWd1Part2Update=1; //局部更新显示参数2
                                              break;
                                     case 3:
                             uiSetData3++;
                             if(uiSetData3>99) //最大值是99
                             {
                                 uiSetData3=99;
                             }
                                                 ucWd1Part3Update=1; //局部更新显示参数3
                                              break;
```

```
                            case 4:
                        uiSetData4++;
                        if(uiSetData4>99) //最大值是99
                        {
                            uiSetData4=99;
                        }
                                                ucWd1Part4Update=1; //局部更新显示参数4
                                            break;
                                }
                    break;
            }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;

    case 2:// 减按键 对应朱兆祺学习板的S5键
            switch(ucWd)  //在不同的窗口下，设置不同的参数
            {
                case 1:
                    switch(ucPart)  //在窗口1下，根据不同的局部闪烁位置来设置不同的参数
                                    {
                                        case 0:
                                                break;
                                        case 1:
                                uiSetData1--;
                                if(uiSetData1>99) //0减去1溢出肯定大于99
                                {
                                    uiSetData1=0;
                                }
                                                ucWd1Part1Update=1; //局部更新显示参数1
                                            break;
                                        case 2:
                                uiSetData2--;
                                if(uiSetData2>99) //0减去1溢出肯定大于99
                                {
                                    uiSetData2=0;
                                }
                                                ucWd1Part2Update=1; //局部更新显示参数2
                                            break;
                                        case 3:
                                uiSetData3--;
                                if(uiSetData3>99) //0减去1溢出肯定大于99
                                {
                                    uiSetData3=0;
                                }
                                                ucWd1Part3Update=1; //局部更新显示参数3
                                            break;
                                        case 4:
                                uiSetData4--;
                                if(uiSetData4>99) //0减去1溢出肯定大于99
```

```
                        {
                            uiSetData4=0;
                        }
                                                ucWd1Part4Update=1; //局部更新显示参数4
                                            break;
                                    }
                        break;
                }
                uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
                break;
    case 3:// 切换"光标闪烁"按键 对应朱兆祺学习板的S9键
                switch(ucWd)  //在不同的窗口下，设置不同的参数
                {
                    case 1:  //在窗口1下，切换"光标闪烁"
                        ucPart++;
                                    if(ucPart>4)
                                    {
                                      ucPart=0;
                                    }
                                    ucWd1Update=1;  //窗口1全部更新显示
                        break;
                }

                uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
                break;

  }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
```

```
            break;
    case 3:  //显示第3位
         ucDigShowTemp=dig_table[ucDigShow3];
                if(ucDigDot3==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
         dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
         ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
         dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
         ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
         dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
         ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
         dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
         ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
         }
         dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
    case 8:  //显示第8位
         ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
         dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
}
```

```
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
```

```
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  uiDpyTimeCnt++;  //数码管的闪烁计时器
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
```

```
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;     //开总中断
   ET0=1;    //允许定时中断
```

```
   TR0=1;     //启动定时中断
}
```

总结陈词:

这节讲了数码管通过闪烁来设置数据的基本程序，但是该程序只有一个窗口。实际应用中，有些项目会有几个窗口，而且每个窗口都要设置几个参数，这样的程序该怎么写？欲知详情，请听下回分解-----数码管通过一二级菜单来设置数据的综合程序。

（未完待续，下节更精彩，不要走开哦）

编者注:

闪烁程序: if(uiDpyTimeCnt==const_dpy_time_half){显示}

else if(uiDpyTimeCnt>const_dpy_time_all){熄灭}

闪烁原理: 闪烁放在void display_service()中。因为没有ucWd1Part1Update==1时，不会一直给局部变量赋值。所以，开始一直显示，uiDpyTimeCnt++

直到(uiDpyTimeCnt>const_dpy_time_all)熄灭（第一次显示时间较长，因为在第一次(uiDpyTimeCnt==const_dpy_time_half)还是显示的），uiDpyTimeCnt清0。

uiDpyTimeCnt++累加，到（uiDpyTimeCnt==const_dpy_time_half)再显示，到(uiDpyTimeCnt>const_dpy_time_all)再熄灭，如此往复。

第三十一节: 数码管通过一二级菜单来设置数据的综合程序。

开场白:

上一节讲了二级菜单，这一节要教会大家两个知识点:

第一个: 数码管通过一二级菜单来设置数据的综合程序框架。

第二个: 继续加深熟悉鸿哥首次提出的“一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换“光标闪烁”按键对应S9键，切换窗口按键对应S13键。

（2）实现功能:

通过按键设置4个不同的参数。

有2个窗口。每个窗口显示2个参数。

第8,7,6,5位数码管显示”P-1 ”代表第1个窗口，显示”P-2 ”代表第2个窗口。第4,3位数码管显示该窗口下其中一个参数，第2,1位数码管显示该窗口下其中另外一个参数。每个参数的范围是从0到99。

有四个按键。

一个是切换窗口按键，依次按下此按键，会依次切换窗口显示。一个是“光标闪烁”按键，依次按下此按键，每两位数码管会依次处于闪烁的状态，哪两位数码管处于闪烁状态时，此时按加键或者减键就可以设置当前选中的参数。依次按下“光标闪烁”按键，数码管会在以下3种状态中循环: 只有第4,3位数码管闪烁---只有第2,1位数码管闪烁---所有的数码管都不闪烁。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
#define const_dpy_time_half  200  //数码管闪烁时间的半值
#define const_dpy_time_all   400  //数码管闪烁时间的全值 一定要比const_dpy_time_half 大
void initial_myself();
void initial_peripheral();
```

```
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
```

```
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucPart=0;//本程序的核心变量，局部显示变量。类似于二级菜单的变量。代表显示不同的局部。
unsigned char ucWd1Part1Update=0;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=0; //在窗口1中，局部2的更新显示标志
unsigned char ucWd2Part1Update=0; //在窗口2中，局部1的更新显示标志
unsigned char ucWd2Part2Update=0; //在窗口2中，局部2的更新显示标志
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
unsigned int  uiDpyTimeCnt=0;  //数码管的闪烁计时器,放在定时中断里不断累加
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
     key_service(); //按键服务的应用程序
         display_service(); //显示的窗口菜单服务程序
   }
```

```
}
/* 注释一:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
*/
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示窗口1的数据
                        if(ucWd1Part1Update==1)  //仅仅参数1局部更新
                        {
                           ucWd1Part1Update=0;   //及时清零标志，避免一直进来扫描
                ucTemp4=uiSetData1/10;  //第1个参数
                ucTemp3=uiSetData1%10;
                           if(uiSetData1<10)
                           {
                              ucDigShow4=10;
                           }
                           else
                           {
                              ucDigShow4=ucTemp4;
                           }
                           ucDigShow3=ucTemp3;
                        }
                        if(ucWd1Part2Update==1)  //仅仅参数2局部更新
                        {
                           ucWd1Part2Update=0;  //及时清零标志，避免一直进来扫描
                ucTemp2=uiSetData2/10;  //第2个参数
                ucTemp1=uiSetData2%10;
                           if(uiSetData2<10)
                           {
                              ucDigShow2=10;
                           }
                           else
                           {
                              ucDigShow2=ucTemp2;
                           }
                           ucDigShow1=ucTemp1;
                        }

/* 注释二:
* 必须注意局部更新和全部更新的编写顺序，局部更新应该写在全部更新之前，
* 当局部更新和全部更新同时发生时，这样就能保证到全部更新的优先响应。
*/
            if(ucWd1Update==1)  //窗口1要全部更新显示
                        {
```

```
    ucWd1Update=0;  //及时清零标志，避免一直进来扫描
    ucTemp8=12;  //显示P
    ucTemp7=11;  //显示-
    ucTemp6=1;  //显示1
    ucTemp5=10;  //显示空
    ucTemp4=uiSetData1/10;  //第1个参数
    ucTemp3=uiSetData1%10;
    ucTemp2=uiSetData2/10;  //第2个参数
    ucTemp1=uiSetData2%10;
    ucDigShow8=ucTemp8;
    ucDigShow7=ucTemp7;
    ucDigShow6=ucTemp6;
    ucDigShow5=ucTemp5;
                if(uiSetData1<10)
                {
                   ucDigShow4=10;
                }
                else
                {
                   ucDigShow4=ucTemp4;
                }
                ucDigShow3=ucTemp3;
                if(uiSetData2<10)
                {
                   ucDigShow2=10;
                }
                else
                {
                   ucDigShow2=ucTemp2;
                }
                ucDigShow1=ucTemp1;

}
            //数码管闪烁
switch(ucPart)  //根据局部变量的值，使对应的参数产生闪烁的动态效果。
            {
               case 0:  //2个参数都不闪烁
                    break;
               case 1:  //第1个参数闪烁
                    if(uiDpyTimeCnt==const_dpy_time_half)
                            {
                       if(uiSetData1<10)        //数码管显示内容
                       {
                          ucDigShow4=10;
                       }
                       else
                       {
                          ucDigShow4=ucTemp4;
                       }
                          ucDigShow3=ucTemp3;
```

```
                                    }
                                else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                                    {
                                       uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                   ucDigShow4=10;    //数码管显示空，什么都不显示
                                   ucDigShow3=10;
                                    }
                                break;
                           case 2:    //第2个参数闪烁
                                if(uiDpyTimeCnt==const_dpy_time_half)
                                    {
                                   if(uiSetData2<10)         //数码管显示内容
                                   {
                                      ucDigShow2=10;
                                   }
                                   else
                                   {
                                      ucDigShow2=ucTemp2;
                                   }
                                   ucDigShow1=ucTemp1;
                                    }
                                else if(uiDpyTimeCnt>const_dpy_time_all)  //const_dpy_time_all一定要比
const_dpy_time_half 大
                                    {
                                       uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                   ucDigShow2=10;    //数码管显示空，什么都不显示
                                   ucDigShow1=10;
                                    }
                                break;

                       }
            break;
       case 2:    //显示窗口2的数据
                       if(ucWd2Part1Update==1)  //在窗口2中，仅仅参数1局部更新
                       {
                          ucWd2Part1Update=0;    //及时清零标志，避免一直进来扫描
              ucTemp4=uiSetData3/10;  //第3个参数
              ucTemp3=uiSetData3%10;
                          if(uiSetData3<10)
                          {
                             ucDigShow4=10;
                          }
                          else
                          {
                             ucDigShow4=ucTemp4;
                          }
                          ucDigShow3=ucTemp3;
                       }
                       if(ucWd2Part2Update==1)  //在窗口2中，仅仅参数2局部更新
```

```
            {
               ucWd2Part2Update=0;  //及时清零标志，避免一直进来扫描
   ucTemp2=uiSetData4/10;  //第4个参数
   ucTemp1=uiSetData4%10;
               if(uiSetData4<10)
               {
                  ucDigShow2=10;
               }
               else
               {
                  ucDigShow2=ucTemp2;
               }
               ucDigShow1=ucTemp1;
            }
if(ucWd2Update==1)  //窗口2要全部更新显示
            {
   ucWd2Update=0;  //及时清零标志，避免一直进来扫描
   ucTemp8=12;  //显示P
   ucTemp7=11;  //显示-
   ucTemp6=2;  //显示2
   ucTemp5=10;  //显示空
   ucTemp4=uiSetData3/10;  //第3个参数
   ucTemp3=uiSetData3%10;
   ucTemp2=uiSetData4/10;  //第4个参数
   ucTemp1=uiSetData4%10;
   ucDigShow8=ucTemp8;
   ucDigShow7=ucTemp7;
   ucDigShow6=ucTemp6;
   ucDigShow5=ucTemp5;
               if(uiSetData3<10)
               {
                  ucDigShow4=10;
               }
               else
               {
                  ucDigShow4=ucTemp4;
               }
               ucDigShow3=ucTemp3;
               if(uiSetData4<10)
               {
                  ucDigShow2=10;
               }
               else
               {
                  ucDigShow2=ucTemp2;
               }
               ucDigShow1=ucTemp1;

}
            //数码管闪烁
```

```
        switch(ucPart)  //根据局部变量的值，使对应的参数产生闪烁的动态效果。
                    {
                        case 0:  //2个参数都不闪烁
                             break;
                        case 1:  //第3个参数闪烁
                             if(uiDpyTimeCnt==const_dpy_time_half)
                                    {
                                if(uiSetData3<10)        //数码管显示内容
                                {
                                   ucDigShow4=10;
                                }
                                else
                                {
                                   ucDigShow4=ucTemp4;
                                }
                                ucDigShow3=ucTemp3;
                                    }
                             else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                                    {
                                       uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                                ucDigShow4=10;   //数码管显示空，什么都不显示
                                ucDigShow3=10;
                                    }
                             break;
                        case 2:   //第4个参数闪烁
                             if(uiDpyTimeCnt==const_dpy_time_half)
                                    {
                                if(uiSetData4<10)        //数码管显示内容
                                {
                                   ucDigShow2=10;
                                }
                                else
                                {
                                   ucDigShow2=ucTemp2;
                                }
                                ucDigShow1=ucTemp1;
                                    }
                             else if(uiDpyTimeCnt>const_dpy_time_all)  //const_dpy_time_all一定要比
const_dpy_time_half 大
                                    {
                                       uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                                ucDigShow2=10;   //数码管显示空，什么都不显示
                                ucDigShow1=10;
                                    }
                             break;

                    }
        break;
    }
```

```
}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
```

```
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;    //触发4号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                   switch(ucPart)  //在窗口1下，根据不同的局部闪烁位置来设置不同的参数
                                  {
                                     case 0:
                                              break;
                                     case 1:
                             uiSetData1++;
                             if(uiSetData1>99) //最大值是99
                             {
                                 uiSetData1=99;
                             }
                                                  ucWd1Part1Update=1; //局部更新显示参数1
                                              break;
                                     case 2:
                             uiSetData2++;
                             if(uiSetData2>99) //最大值是99
                             {
                                 uiSetData2=99;
                             }
                                                  ucWd1Part2Update=1; //局部更新显示参数2
                                              break;
                                  }
                   break;
              case 2:
                   switch(ucPart)  //在窗口2下，根据不同的局部闪烁位置来设置不同的参数
                                  {
                                     case 0:
                                              break;
```

```
                        case 1:
                    uiSetData3++;
                    if(uiSetData3>99) //最大值是99
                    {
                        uiSetData3=99;
                    }
                                        ucWd2Part1Update=1; //局部更新显示参数1
                                    break;
                        case 2:
                    uiSetData4++;
                    if(uiSetData4>99) //最大值是99
                    {
                        uiSetData4=99;
                    }
                                        ucWd2Part2Update=1; //局部更新显示参数2
                                    break;
                            }
            break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart)  //在窗口1下，根据不同的局部闪烁位置来设置不同的参数
                                {
                                    case 0:
                                        break;
                                    case 1:
                    uiSetData1--;
                    if(uiSetData1>99) //0减去1溢出肯定大于99
                    {
                        uiSetData1=0;
                    }
                                        ucWd1Part1Update=1; //局部更新显示参数1
                                    break;
                                case 2:
                    uiSetData2--;
                    if(uiSetData2>99) //0减去1溢出肯定大于99
                    {
                        uiSetData2=0;
                    }
                                        ucWd1Part2Update=1; //局部更新显示参数2
                                    break;
                            }
                break;
            case 2:
```

```
            switch(ucPart)  //在窗口2下，根据不同的局部闪烁位置来设置不同的参数
                            {
                               case 0:
                                        break;
                               case 1:
                        uiSetData3--;
                        if(uiSetData3>99) //0减去1溢出肯定大于99
                        {
                            uiSetData3=0;
                        }
                                            ucWd2Part1Update=1; //局部更新显示参数1
                                        break;
                               case 2:
                        uiSetData4--;
                        if(uiSetData4>99) //0减去1溢出肯定大于99
                        {
                            uiSetData4=0;
                        }
                                            ucWd2Part2Update=1; //局部更新显示参数2
                                        break;
                            }
                break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 切换"光标闪烁"按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:  //在窗口1下，切换"光标闪烁"
                ucPart++;
                            if(ucPart>2)
                            {
                              ucPart=0;
                            }
                            ucWd1Update=1;  //窗口1全部更新显示
                break;
            case 2:  //在窗口2下，切换"光标闪烁"
                ucPart++;
                            if(ucPart>2)
                            {
                              ucPart=0;
                            }
                            ucWd2Update=1;  //窗口2全部更新显示
                break;
        }

        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
```

```
    case 4:// 切换窗口按键 对应朱兆祺学习板的S13键
              ucWd++;
                  if(ucWd>2)
                  {
                     ucWd=1;
                  }
                  ucPart=0; //强行把局部变量复位，让新切换的窗口不闪烁
          switch(ucWd)  //在不同的窗口下，更新显示不同的窗口
          {
              case 1:  //在窗口1下
                                  ucWd1Update=1;  //窗口1全部更新显示
                   break;
              case 2:  //在窗口2下
                                  ucWd2Update=1;  //窗口2全部更新显示
                   break;
          }

          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
```

```
            dig_hc595_drive(ucDigShowTemp,0xfb);
                break;
        case 4:  //显示第4位
            ucDigShowTemp=dig_table[ucDigShow4];
                    if(ucDigDot4==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xf7);
                break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                    if(ucDigDot5==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                    if(ucDigDot6==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                    if(ucDigDot7==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
```

```
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   dig_hc595_sh_dr=0;
   dig_hc595_st_dr=0;
   ucTempData=ucDigStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucDigStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   dig_hc595_st_dr=1;
   delay_short(1);
   dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
   dig_hc595_st_dr=0;
   dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
```

```
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  uiDpyTimeCnt++;  //数码管的闪烁计时器
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
```

```
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
    ucDigDot8=0;   //小数点全部不显示
    ucDigDot7=0;
    ucDigDot6=0;
    ucDigDot5=0;
    ucDigDot4=0;
    ucDigDot3=0;
    ucDigDot2=0;
    ucDigDot1=0;
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
```

总结陈词:

这节讲了数码管通过一二级菜单来设置数据的综合程序，鸿哥的人机界面程序框架基本上都涉及到了，为了继续加深熟悉鸿哥的“一二级菜单显示理论”，下一节会继续讲一个常用的数码管项目小程序，这个项目小程序鸿哥是怎么写的？欲知详情，请听下回分解-----数码管中的倒计时程序。

（未完待续，下节更精彩，不要走开哦）

第三十二节：数码管中的倒计时程序。

开场白：

上一节讲了一二级菜单的综合程序，这一节要教会大家三个知识点：

第一个：通过本程序，继续加深理解按键与数码管的关联方法。

第二个：复习一下我在第五节教给大家的时间校正法。

第三个：继续加深熟悉鸿哥首次提出的“一二级菜单显示理论”：凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。启动和暂停键对应S1键，复位键对应S5键。

（2）实现功能：按下启动暂停按键时，倒计时开始工作，再按一次启动暂停按键时，则暂停倒计时。在任何时候，按下复位按键，倒计时将暂停工作，并且恢复倒计时当前默认值99。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_voice_long   200    //蜂鸣器长叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_dpy_time_half  200  //数码管闪烁时间的半值
#define const_dpy_time_all   400  //数码管闪烁时间的全值 一定要比const_dpy_time_half 大
/* 注释一:
* 如何知道1秒钟需要多少个定时中断?
* 这个需要编写一段小程序测试，得到测试的结果后再按比例修正。
* 步骤:
* 第一步：在程序代码上先写入1秒钟大概需要200个定时中断。
* 第二步：把程序烧录进单片机后，上电开始测试，手上同步打开手机里的秒表。
*         如果单片机倒计时跑完了99秒，而手机上的秒表才走了45秒。
* 第三步：那么最终得出1秒钟需要的定时中断次数是:const_1s=(200*99)/45=440
*/
#define const_1s  440   //大概一秒钟所需要的定时中断次数
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
```

```
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;    //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucCountDown=99;  //倒计时的当前值
unsigned char ucStartFlag=0;  //暂停与启动的标志位
unsigned int  uiTimeCnt=0;  //倒计时的时间计时器
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
```

```
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
       display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释二:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
*/
void display_service() //显示的窗口菜单服务程序
{
  //由于本程序只有一个窗口，读者在做实际项目的时候，可以省略switch(ucWd)
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示窗口1的数据
            if(ucWd1Update==1)  //窗口1要全部更新显示
                       {
              ucWd1Update=0;  //及时清零标志，避免一直进来扫描
              ucTemp8=10;  //显示空
              ucTemp7=10;  //显示空
              ucTemp6=10;  //显示空
              ucTemp5=10;  //显示空
              ucTemp4=10;  //显示空
              ucTemp3=10;  //显示空
              ucTemp2=ucCountDown/10;  //倒计时的当前值
              ucTemp1=ucCountDown%10;
              ucDigShow8=ucTemp8;
              ucDigShow7=ucTemp7;
              ucDigShow6=ucTemp6;
              ucDigShow5=ucTemp5;
              ucDigShow4=ucTemp4;
```

```
                ucDigShow3=ucTemp3;
                            if(ucCountDown<10)
                            {
                               ucDigShow2=10;
                            }
                            else
                            {
                               ucDigShow2=ucTemp2;
                            }
                            ucDigShow1=ucTemp1;

            }
            break;

     }

}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
}
```

```
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 启动和暂停按键 对应朱兆祺学习板的S1键

         //由于本程序只有一个窗口，读者在做实际项目的时候，可以省略switch(ucWd)
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                   if(ucStartFlag==0)  //如果原来处于暂停的状态，则启动
                                       {
                      ucStartFlag=1; //启动
                                       }
                                       else     //如果原来处于启动的状态，则暂停
                                       {
                                          ucStartFlag=0;  //暂停
                                       }
                   break;

          }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

    case 2:// 复位按键 对应朱兆祺学习板的S5键
         //由于本程序只有一个窗口，读者在做实际项目的时候，可以省略switch(ucWd)
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                                       ucStartFlag=0;  //暂停
                   ucCountDown=99;  //恢复倒计时的默认值99
                   uiTimeCnt=0;  //倒计时的时间计时器清零
                                       ucWd1Update=1; //窗口1更新显示标志  只要ucCountDown变化了，就要更新显
示一次
                   break;

          }
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
```

```
            if(ucDigDot1==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xfe);
         break;
case 2:  //显示第2位
     ucDigShowTemp=dig_table[ucDigShow2];
            if(ucDigDot2==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xfd);
         break;
case 3:  //显示第3位
     ucDigShowTemp=dig_table[ucDigShow3];
            if(ucDigDot3==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xfb);
         break;
case 4:  //显示第4位
     ucDigShowTemp=dig_table[ucDigShow4];
            if(ucDigDot4==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xf7);
         break;
case 5:  //显示第5位
     ucDigShowTemp=dig_table[ucDigShow5];
            if(ucDigDot5==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xef);
         break;
case 6:  //显示第6位
     ucDigShowTemp=dig_table[ucDigShow6];
            if(ucDigDot6==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xdf);
         break;
case 7:  //显示第7位
     ucDigShowTemp=dig_table[ucDigShow7];
            if(ucDigDot7==1)
            {
```

```
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
```

```
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;     //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;     //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(ucStartFlag==1)  //启动倒计时的计时器
  {
```

```
      uiTimeCnt++;
      if(uiTimeCnt>=const_1s)     //1秒钟的时间到
      {
             if(ucCountDown!=0) //加这个判断，就是避免在0的情况下减1
             {
                ucCountDown--;  //倒计时当前显示值减1
             }
          if(ucCountDown==0)  //倒计时结束
              {
                 ucStartFlag=0;  //暂停
             uiVoiceCnt=const_voice_long; //蜂鸣器触发提醒，滴一声就停。
              }
          ucWd1Update=1; //窗口1更新显示标志
          uiTimeCnt=0;    //计时器清零，准备从新开始计时
      }
   }
   if(uiVoiceCnt!=0)
   {
      uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
      beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//      beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
   }
   else
   {
      ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
      beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//      beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
   }
   display_drive();  //数码管字模的驱动函数
   TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
   TL0=0x0b;
   TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;    //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
```

```
        }
    }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

总结陈词:

这节讲了数码管中的倒计时程序。如果要在此程序上多增加两个按键，用来控制数码管倒计时的速度档位，并且需要在数码管中闪烁显示被设置的速度档位，该怎么编写这个程序？欲知详情，请听下回分解-----能设置速度档位的数码管倒计时程序。

（未完待续，下节更精彩，不要走开哦）

第三十三节: 能设置速度档位的数码管倒计时程序。

开场白:

上一节讲了数码管中的倒计时程序。这节要在此程序上多增加两个按键，用来控制数码管倒计时的速度档位，并且需要在数码管中闪烁显示被设置的速度档位。这一节要教会大家三个知识点:

第一个: 把一个按键的短按与长按复合应用在项目中的程序结构。

第二个: 通过本程序，继续加深理解按键与数码管的关联方法。

第三个: 继续加深熟悉鸿哥首次提出的“一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。启动和暂停键对应S1键，复位键对应S5键。加键对应S9键，减键对应S13键。

（2）实现功能：按下启动暂停按键时，倒计时开始工作，再按一次启动暂停按键时，则暂停倒计时。在任何时候，按下复位按键，倒计时将暂停工作，并且恢复倒计时当前默认值99。如果长按复位按键，在数码管会切换到第2个闪烁窗口，用来设置速度档位，修改完速度档位后，再一次按下复位按键，或者直接按启动暂停按键，会切换回窗口1显示倒计时的当前数据。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_voice_long   200    //蜂鸣器长叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
#define const_key_long_time 200  //长按复位按键的时间
#define const_dpy_time_half  200  //数码管闪烁时间的半值
#define const_dpy_time_all   400  //数码管闪烁时间的全值 一定要比const_dpy_time_half 大
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
```

```
unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=1; //窗口2更新显示标志
unsigned char ucCountDown=99;  //倒计时的当前值
unsigned char ucStartFlag=0;  //暂停与启动的标志位
unsigned int  uiTimeCnt=0;  //倒计时的时间计时器
unsigned int  uiDpyTimeCnt=0;  //数码管的闪烁计时器,放在定时中断里不断累加
unsigned int  uiSetData1=50;  //速度档位
unsigned int  uiSpeedCnt=400;  //影响速度变量，它跟速度档位uiSetData1有关联
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
```

```
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       key_service(); //按键服务的应用程序
       display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
*/
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示窗口1的数据
            if(ucWd1Update==1)  //窗口1要全部更新显示
                       {
              ucWd1Update=0;  //及时清零标志，避免一直进来扫描
              ucTemp8=10;  //显示空
              ucTemp7=10;  //显示空
              ucTemp6=10;  //显示空
              ucTemp5=10;  //显示空
              ucTemp4=10;  //显示空
              ucTemp3=10;  //显示空
              ucTemp2=ucCountDown/10;  //倒计时的当前值
              ucTemp1=ucCountDown%10;
              ucDigShow8=ucTemp8;
              ucDigShow7=ucTemp7;
              ucDigShow6=ucTemp6;
              ucDigShow5=ucTemp5;
              ucDigShow4=ucTemp4;
              ucDigShow3=ucTemp3;
                          if(ucCountDown<10)
                          {
                             ucDigShow2=10;
                          }
                          else
                          {
```

```
                    ucDigShow2=ucTemp2;
                }
                ucDigShow1=ucTemp1;

    }
    break;
case 2:   //显示窗口2的数据
    if(ucWd2Update==1)  //窗口2要全部更新显示
    {
       ucWd2Update=0;  //及时清零标志，避免一直进来扫描
       ucTemp8=10;  //显示空
       ucTemp7=10;  //显示空
       ucTemp6=10;  //显示空
       ucTemp5=10;  //显示空
       ucTemp4=10;  //显示空
       ucTemp3=10;  //显示空
       ucTemp2=uiSetData1/10;  //倒计时的速度档位
       ucTemp1=uiSetData1%10;
       ucDigShow8=ucTemp8;
       ucDigShow7=ucTemp7;
       ucDigShow6=ucTemp6;
       ucDigShow5=ucTemp5;
       ucDigShow4=ucTemp4;
       ucDigShow3=ucTemp3;
                if(uiSetData1<10)
                {
                   ucDigShow2=10;
                }
                else
                {
                   ucDigShow2=ucTemp2;
                }
                ucDigShow1=ucTemp1;

    }
    //数码管闪烁
    if(uiDpyTimeCnt==const_dpy_time_half)
    {
       if(uiSetData1<10)        //数码管显示内容
       {
            ucDigShow2=10;
       }
       else
       {
            ucDigShow2=ucTemp2;
       }
       ucDigShow1=ucTemp1;
    }
    else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比const_dpy_time_half 大
    {
```

```
                uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                ucDigShow2=10;    //数码管显示空，什么都不显示
                ucDigShow1=10;
              }
              break;

      }

}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
/* 注释二:
* 请注意以下长按复位按键与短按复位按键的写法。在本程序中，每次长按复位按键必然
* 触发一次短按复位按键。
*/
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  else if(uiKeyTimeCnt2<const_key_long_time)   //长按复位按键
  {
      uiKeyTimeCnt2++;
          if(uiKeyTimeCnt2==const_key_long_time)
          {
```

```
            ucKeySec=17;     //触发17号长按复位键
          }
  }
if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;     //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;     //触发4号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 启动和暂停按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                   if(ucStartFlag==0)  //如果原来处于暂停的状态，则启动
                                {
                      ucStartFlag=1; //启动
                                }
                                else     //如果原来处于启动的状态，则暂停
                                {
                                   ucStartFlag=0;  //暂停
                                }
```

```
                break;

        }
        ucWd=1;  //不管在哪个窗口，强行切换回窗口1
                ucWd1Update=1; //窗口1更新显示标志
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 2:// 复位按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:    //在窗口1中
                                ucStartFlag=0;  //暂停
                ucCountDown=99;  //恢复倒计时的默认值99
                uiTimeCnt=0;  //倒计时的时间计时器清零
                                ucWd1Update=1; //窗口1更新显示标志  只要ucCountDown变化了，就要更新显
示一次
                break;
            case 2:    //在窗口2中
                ucWd=1;  //切换回窗口1
                                ucWd1Update=1; //窗口1更新显示标志
                break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3:// 加按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 2:   //在窗口2中
                uiSetData1++;       //速度档位累加，档位越大，速度越快.
                                if(uiSetData1>99)
                                {
                                   uiSetData1=99;
                                }
                uiSpeedCnt=440-(uiSetData1*2);  //速度档位越大，累计中断数uiSpeedCnt越小，从而倒计时
的时间越快
                                ucWd2Update=1; //窗口2更新显示
                break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 4:// 减按键 对应朱兆祺学习板的S13键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 2:   //在窗口2中
                                if(uiSetData1>0)  //加此条件判断，避免0减1
```

```
                                {
                    uiSetData1--;       //速度档位累减，档位越小，速度越慢.
                                }
                uiSpeedCnt=440-(uiSetData1*2);  //速度档位越小，累计中断数uiSpeedCnt越大，从而倒计时的时间越慢
                                ucWd2Update=1; //窗口2更新显示
                break;
            }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 17:// 长按复位按键 对应朱兆祺学习板的S5键
            switch(ucWd)  //在不同的窗口下，设置不同的参数
            {
                case 1:  //窗口1下
                    ucWd=2;  //切换到闪烁窗口2  进行设置速度档位显示
                                ucWd2Update=1; //窗口2更新显示标志
                    break;

            }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                if(ucDigDot1==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                if(ucDigDot2==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                if(ucDigDot3==1)
                {
```

```
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
        ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
        }
        dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
    case 8:  //显示第8位
        ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
}
ucDisplayDriveStep++;
if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
{
  ucDisplayDriveStep=1;
}
```

```
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   dig_hc595_sh_dr=0;
   dig_hc595_st_dr=0;
   ucTempData=ucDigStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucDigStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   dig_hc595_st_dr=1;
   delay_short(1);
   dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
   dig_hc595_st_dr=0;
   dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
```

```
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(ucStartFlag==1)  //启动倒计时的计时器
  {
     uiTimeCnt++;
     if(uiTimeCnt>=uiSpeedCnt)    //时间到
     {
            if(ucCountDown!=0) //加这个判断，就是避免在0的情况下减1
            {
               ucCountDown--;  //倒计时当前显示值减1
            }
        if(ucCountDown==0)  //倒计时结束
            {
               ucStartFlag=0;  //暂停
           uiVoiceCnt=const_voice_long; //蜂鸣器触发提醒，滴一声就停。
            }
        ucWd1Update=1; //窗口1更新显示标志
        uiTimeCnt=0;   //计时器清零，准备从新开始计时
     }
  }
  uiDpyTimeCnt++;  //数码管的闪烁计时器
  if(uiVoiceCnt!=0)
```

```
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
```

```
}
void initial_peripheral() //第二区 初始化外围
{
   ucDigDot8=0;    //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   uiSpeedCnt=440-(uiSetData1*2);  //速度档位越大，累计中断数uiSpeedCnt越小，从而倒计时的时间越快
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

总结陈词:

这节讲了能设置速度档位的数码管倒计时程序。现在很多人用iphone4S的手机，这个手机每次开机显示的时候，都要通过4个密码开锁，如果我们要用4位数码管来实现这个密码锁功能，该怎么编写这个程序？欲知详情，请听下回分解-----在数码管中实现iphone4S开机密码锁的程序。

（未完待续，下节更精彩，不要走开哦）

第三十四节：在数码管中实现iphone4S开机密码锁的程序。

开场白:

这一节要教会大家四个知识点:

第一个：类似手机上10秒钟内无按键操作将自动进入锁屏的程序。

第二个：如何用一个数组来接收按键的一串数字输入。

第三个：矩阵键盘中，数字按键的输入，由于这部分按键的代码相似度非常高，因此把它封装在一个函数里可以非常简洁方便。

第四个：继续加深熟悉鸿哥首次提出的“一二级菜单显示理论”：凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。数字1键对应S1键，数字2键对应S2键，数字3键对应S3键….数字9键对应S9键，数字0键对应S10键。其他的按键不用。

（2）实现功能:

本程序有3个窗口。

开机显示第1个密码登录框窗口“----”，在这个窗口下输入密码，如果密码等于”9922”表示密码正确，将会切换到第2个显示按键值的窗口。在窗口2下，按不同的按键会显示不同的按键值，如果10秒内没有按键操作，将会自动切换到第1个密码登录窗口，类似手机上的自动锁屏操作。在密码登录窗口1下，如果密码不正确，会自动清除密码的数字，继续在窗口1下显示”----”。

窗口3是用来停留0.5秒显示全部密码的信息，然后根据密码的正确与否自动切换到对应的窗口。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_no_key_push 4400   //大概10秒内无按键按下的时间
#define const_0_1s  220            //大概0.5秒的时间
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
```

```
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void number_key_input(unsigned char ucWhichKey);  //由于数字按键的代码相似度高，因此封装在这个函数里
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
```

```
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned char ucInputPassword[4];  //在第1个窗口下，显示输入的4个密码
unsigned char ucPasswordCnt=0; //记录当前已经输入到哪一位密码了
unsigned char ucKeyNumber=1;  //在第2个窗口下，显示当前被按下的按键
unsigned int  uiNoKeyPushTimer=const_no_key_push;  //10秒内无按键按下的计时器
unsigned int  uiPasswordTimer=const_0_1s;  //显示0.5秒钟全部密码的计时器，让窗口3停留显示0.5秒钟之后自动消失
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
     key_service(); //按键服务的应用程序
         display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
*鸿哥首次提出的"一二级菜单显示理论":
*凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，
*每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。
*局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，
*表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
```

```
*/
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示输入密码的登录框
            if(ucWd1Update==1)  //窗口1要全部更新显示
            {
               ucWd1Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=10;  //第8位数码管显示无
               ucDigShow7=10;  //第7位数码管显示无
               ucDigShow6=10;  //第6位数码管显示无
               ucDigShow5=10;  //第5位数码管显示无
               ucDigShow4=ucInputPassword[0];  //第4位数码管显示输入的密码
               ucDigShow3=ucInputPassword[1];  //第3位数码管显示输入的密码
               ucDigShow2=ucInputPassword[2];  //第2位数码管显示输入的密码
               ucDigShow1=ucInputPassword[3];  //第1位数码管显示输入的密码
            }
            break;
        case 2:  //显示被按下的键值
            if(ucWd2Update==1)  //窗口2要全部更新显示
            {
               ucWd2Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=10;  //第8位数码管显示无
               ucDigShow7=10;  //第7位数码管显示无
               ucDigShow6=10;  //第6位数码管显示无
               ucDigShow5=10;  //第5位数码管显示无
               ucDigShow4=10;  //第4位数码管显示无
               ucDigShow3=10;  //第3位数码管显示无
               ucDigShow2=10;  //第2位数码管显示无
               ucDigShow1=ucKeyNumber; //第1位数码管显示被按下的键值
            }
            break;
       case 3:   //当输入完4个密码后，显示1秒钟的密码登录框，
            if(ucWd3Update==1)  //窗口3要全部更新显示
            {
               ucWd3Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=10;  //第8位数码管显示无
               ucDigShow7=10;  //第7位数码管显示无
               ucDigShow6=10;  //第6位数码管显示无
               ucDigShow5=10;  //第5位数码管显示无
               ucDigShow4=ucInputPassword[0];  //第4位数码管显示输入的密码
               ucDigShow3=ucInputPassword[1];  //第3位数码管显示输入的密码
               ucDigShow2=ucInputPassword[2];  //第2位数码管显示输入的密码
               ucDigShow1=ucInputPassword[3];  //第1位数码管显示输入的密码
            }
            break;
    }

}
```

```
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
     case 1:    //按键扫描输出第ucRowRecord列低电平
               if(ucRowRecord==1)  //第一列输出低电平
                   {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==2)  //第二列输出低电平
                   {
              key_dr1=1;
              key_dr2=0;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==3)  //第三列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=0;
              key_dr4=1;
                   }
               else   //第四列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=1;
              key_dr4=0;
                   }
          uiKeyTimeCnt=0;  //延时计数器清零
          ucKeyStep++;     //切换到下一个运行步骤
              break;
     case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
          uiKeyTimeCnt++;
                   if(uiKeyTimeCnt>1)
                   {
                      uiKeyTimeCnt=0;
              ucKeyStep++;     //切换到下一个运行步骤
                   }
               break;
     case 3:
          if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
          {
              ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
              ucKeyLock=0;  //按键自锁标志清零
              uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙
```

```
                ucRowRecord++;  //输出下一列
                if(ucRowRecord>4)
                {
                   ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                }
        }
            else if(ucKeyLock==0)  //有按键按下，且是第一次触发
            {
               if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                   {
                      uiKeyTimeCnt++;  //去抖动延时计数器
                          if(uiKeyTimeCnt>const_key_time)
                          {
                             uiKeyTimeCnt=0;
                             ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                 if(ucRowRecord==1)  //第一列输出低电平
                     {
                                 ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                     }
                 else if(ucRowRecord==2)  //第二列输出低电平
                     {
                                 ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                     }
                 else if(ucRowRecord==3)  //第三列输出低电平
                     {
                                 ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                     }
                 else   //第四列输出低电平
                     {
                                 ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                     }
                          }

                 }
               else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                   {
                      uiKeyTimeCnt++;  //去抖动延时计数器
                          if(uiKeyTimeCnt>const_key_time)
                          {
                             uiKeyTimeCnt=0;
                             ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                 if(ucRowRecord==1)  //第一列输出低电平
                     {
                                 ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
                     }
                 else if(ucRowRecord==2)  //第二列输出低电平
                     {
```

```
                    ucKeySec=6;   //触发6号键 对应朱兆祺学习板的S6键
            }
        else if(ucRowRecord==3)   //第三列输出低电平
            {
                    ucKeySec=7;   //触发7号键 对应朱兆祺学习板的S7键
            }
        else    //第四列输出低电平
            {
                    ucKeySec=8;   //触发8号键 对应朱兆祺学习板的S8键
            }
                }

      }
    else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
      {
        uiKeyTimeCnt++;   //去抖动延时计数器
            if(uiKeyTimeCnt>const_key_time)
            {
               uiKeyTimeCnt=0;
               ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
        if(ucRowRecord==1)   //第一列输出低电平
            {
                    ucKeySec=9;   //触发9号键 对应朱兆祺学习板的S9键
            }
        else if(ucRowRecord==2)   //第二列输出低电平
            {
                    ucKeySec=10;   //触发10号键 对应朱兆祺学习板的S10键
            }
        else if(ucRowRecord==3)   //第三列输出低电平
            {
                    ucKeySec=11;   //触发11号键 对应朱兆祺学习板的S11键
            }
        else    //第四列输出低电平
            {
                    ucKeySec=12;   //触发12号键 对应朱兆祺学习板的S12键
            }
                }

      }
    else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
      {
        uiKeyTimeCnt++;   //去抖动延时计数器
            if(uiKeyTimeCnt>const_key_time)
            {
               uiKeyTimeCnt=0;
               ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
        if(ucRowRecord==1)   //第一列输出低电平
            {
```

```
                                ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                        }
                    else if(ucRowRecord==2)  //第二列输出低电平
                        {
                                ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                        }
                    else if(ucRowRecord==3)  //第三列输出低电平
                        {
                                ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                        }
                    else   //第四列输出低电平
                        {
                                ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                        }
                            }

                      }

                }
            break;
  }
}
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
          number_key_input(1);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S2键
          number_key_input(2);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 3号键 对应朱兆祺学习板的S3键
          number_key_input(3);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 4:// 4号键 对应朱兆祺学习板的S4键
          number_key_input(4);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 5:// 5号键 对应朱兆祺学习板的S5键
          number_key_input(5);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
```

```
        break;
    case 6:// 6号键 对应朱兆祺学习板的S6键
        number_key_input(6);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 7:// 7号键 对应朱兆祺学习板的S7键
        number_key_input(7);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 8:// 8号键 对应朱兆祺学习板的S8键
        number_key_input(8);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 9:// 9号键 对应朱兆祺学习板的S9键
        number_key_input(9);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 10:// 把这个按键专门用来输入数字0     对应朱兆祺学习板的S10键
        number_key_input(0);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 11:// 11号键 对应朱兆祺学习板的S11键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 12:// 12号键 对应朱兆祺学习板的S12键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 13:// 13号键 对应朱兆祺学习板的S13键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 14:// 14号键 对应朱兆祺学习板的S14键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 15:// 15号键 对应朱兆祺学习板的S15键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 16:// 16号键 对应朱兆祺学习板的S16键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
```

```
  }
}
void number_key_input(unsigned char ucWhichKey)  //由于数字按键的代码相似度高，因此封装在这个函数里
{
    switch(ucWd)
           {
           case 1:   //在显示密码登录框的窗口下
            ucInputPassword[ucPasswordCnt]=ucWhichKey;   //输入的密码值显示
            ucPasswordCnt++;
                        if(ucPasswordCnt>=4)
                        {
                            ucPasswordCnt=0;
                                ucWd=3;//切换到第3个的窗口,停留显示1秒钟全部密码
                            ucWd3Update=1;  //更新显示窗口3
                    uiPasswordTimer=const_0_1s;  //显示0.5秒钟全部密码的计时器，让窗口3停留显示0.5秒钟之后自
动消失
                        }
                        ucWd1Update=1; //更新显示窗口1
                        uiNoKeyPushTimer=const_no_key_push;  //10秒内无按键按下的计时器赋新值
                        break;
           case 2:   //在显示按键值的窗口下
                ucKeyNumber=ucWhichKey; //输入的按键数值显示
                    ucWd2Update=1;  //更新显示窗口2
                        uiNoKeyPushTimer=const_no_key_push;  //10秒内无按键按下的计时器赋新值
                    break;
    }
}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
```

```
                if(ucDigDot3==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
        ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
        }
        dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
    case 8:  //显示第8位
        ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
}
ucDisplayDriveStep++;
if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
{
```

```
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
```

```
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  unsigned int i;
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(ucWd==3)  //在窗口3下
  {
     if(uiPasswordTimer>0)
         {
            uiPasswordTimer--;
         }
     if(uiPasswordTimer==0)
         {
if(ucInputPassword[0]==9&&ucInputPassword[1]==9&&ucInputPassword[2]==2&&ucInputPassword[3]==2)
            {     //如果密码等于9922，则正确
                    ucWd=2;//切换到第2个显示按键的窗口
                    ucWd2Update=1;  //更新显示窗口2
            }
                else //如果密码不正确，则继续显示----
            {
              for(i=0;i<4;i++)
              {
                 ucInputPassword[i]=11;  //开机默认密码全部显示"----"
```

```
            }
                        ucWd=1;
                    ucWd1Update=1; //更新显示窗口1
                  }
          }
  }
  if(ucWd==2)  //在窗口2下
  {
     if(uiNoKeyPushTimer>0)
         {
            uiNoKeyPushTimer--;
         }
     if(uiNoKeyPushTimer==0)//如果10秒内无按键按下，则自动切换到显示密码登录框的界面
         {
       for(i=0;i<4;i++)
       {
          ucInputPassword[i]=11;  //开机默认密码全部显示"----"
       }
           ucWd=1;
           ucWd1Update=1; //更新显示窗口1
         }
  }
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
```

```
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=0;  //关闭独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
    unsigned int i; //个人的变量命名习惯，i,j,k等单个字母的变量名只用在for循环里
    for(i=0;i<4;i++)
    {
      ucInputPassword[i]=11;  //开机默认密码全部显示"----"
    }
    ucDigDot8=0;   //小数点全部不显示
    ucDigDot7=0;
    ucDigDot6=0;
    ucDigDot5=0;
    ucDigDot4=0;
    ucDigDot3=0;
    ucDigDot2=0;
    ucDigDot1=0;
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节讲了iphone4S开机密码锁的程序。2014年春节的时候，一帮朋友举行小规模的象棋比赛，有一些朋友下棋的速度实在是太慢了，为了限制比赛时间，我专门用朱兆祺的51学习板做了一个棋类比赛专用计时器给他们用，这个程序该怎么编写？欲知详情，请听下回分解-----带数码管显示的象棋比赛专用计时器。

第三十五节：带数码管显示的象棋比赛专用计时器。

开场白:

2014年春节的时候，一帮朋友举行小规模的象棋比赛，有一些朋友下棋的速度实在是太慢了，为了限制比赛时间，我专门用朱兆祺的51学习板做了一个棋类比赛专用计时器给他们用。这一节要教会大家两个知识点:

第一个: 按键服务程序操作的精髓在于根据当前系统处于什么窗口状态下就执行什么操作。紧紧围绕着不同的窗口ucWd来执行不同的操作。

第二个: 继续加深熟悉鸿哥首次提出的“一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可

以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。
具体内容，请看源代码讲解。
（1）硬件平台：基于朱兆祺51单片机学习板。
刚上电开机时，红棋加时键对应S1键，红棋减时键对应S2键.。
刚上电开机时，黑棋加时键对应S3键，黑棋减时键对应S4键.。
比赛中途暂停双方计时的暂停按键对应S6键。刚上电时，复位双方默认20分时间的复位按键对应S7按键。
红棋的抢时按键对应S13键，黑棋的抢时按键对应S16按键。
（2）实现功能：
棋类计时器有点像抢答器，本质上有两个计时器。比赛的时候对弈的两个棋友各用一个不同的按键抢时间，红棋走一步棋后，就按一下自己的抢时按键，这个时候红棋的计时器停止计时，而黑棋的计时器开始计时，黑棋走了一步棋后，按一下自己的计时器，黑棋停止计时，红棋继续计时，依次循环，谁的时间最先用完谁就输，蜂鸣器也会发出长鸣的声音提示时间到。
上电开机默认双方各有20分钟的时间，左边显示的是红棋的时间，右边显示的是黑棋的时间。此时可以通过S1,S2.,S3,S4的加减按键来设置各自的最大倒计时时间。此时如果按下复位按键S7，会自动把双方的时间设置为默认的20分钟。
设置好最大倒计时的时间后，此时任意一方按下各自的抢时按键（S13或者S16），则自己的计时器停止计时，而对方开始倒计时。此时数码管显示的是对方的时间，而自己的时间屏蔽不显示。
在开始倒计时的时候，如果中途有棋友要接听电话或者忙别的事情，需要暂时暂停一下双方的时间，这个时候可以按S6暂停按键来暂停双方的计时，忙完后再次按下暂停按键会继续倒计时。任何一方的时间走完，都会蜂鸣器长鸣提示。
（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_voice_long   900   //蜂鸣器长叫的持续时间
#define const_key_time  10    //按键去抖动延时的时间
#define const_1s     422   //产生一秒钟的时间基准
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service();
void key_scan(); //按键扫描函数 放在定时中断里
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive();  //放在定时中断里的数码管驱动函数
void time_service();  //放在定时中断里的时间应用程序
void display_service();
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管 的74HC595程序
sbit dig_hc595_st_dr=P2^1;
```

```
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8=0;  //第8位数码管要显示的内容
unsigned char ucDigShow7=0;  //第7位数码管要显示的内容
unsigned char ucDigShow6=0;  //第6位数码管要显示的内容
unsigned char ucDigShow5=0;  //第5位数码管要显示的内容
unsigned char ucDigShow4=0;  //第4位数码管要显示的内容
unsigned char ucDigShow3=0;  //第3位数码管要显示的内容
unsigned char ucDigShow2=0;  //第2位数码管要显示的内容
unsigned char ucDigShow1=0;  //第1位数码管要显示的内容
unsigned char ucDigDot3=1;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot7=1;  //数码管7的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned int uiRedTimeCnt=0;    //红棋产生秒基准的时间计时器
unsigned int uiBlackTimeCnt=0;  //黑棋产生秒基准的时间计时器
unsigned int uiRedTotal=1200;    //红棋的总时间
unsigned int uiBlackTotal=1200;  //黑棋的总时间
unsigned char ucRedFlag=0;  //红棋是否开始计时的标志
unsigned char ucBlackFlag=0;  //黑棋是否开始计时的标志
unsigned char ucDisplayUpdate=1; //更新显示标志
/* 注释一:
*  ucWd变量是本程序最核心的变量，代表显示哪一个窗口和系统处于当前哪种状态
*/
unsigned char ucWd=1;
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //不显示  序号10
};
void main()
  {
   initial_myself();
```

```
    delay_long(100);
    initial_peripheral();
    while(1)
    {
        key_service();
        display_service();
    }
}
void time_service()  //放在定时中断里的时间应用程序
{
  if(ucRedFlag==1)  //1代表红棋在运行中
  {
     uiRedTimeCnt++;
         if(uiRedTimeCnt>const_1s)
         {
        uiRedTimeCnt=0;
        if(uiRedTotal>0)
                {
                   uiRedTotal--;
                }
                else  //时间到
                {
                    ucRedFlag=0;    //红棋和黑棋同时停止计时
                   ucBlackFlag=0;
                   ucWd=1;  //切换到第一个窗口的状态
                   uiVoiceCnt=const_voice_long; //报警声音触发
                }

        ucDisplayUpdate=1;  //更新显示
         }
  }
  if(ucBlackFlag==1)  //1代表黑棋在运行中
  {
     uiBlackTimeCnt++;
         if(uiBlackTimeCnt>const_1s)
         {
        uiBlackTimeCnt=0;
        if(uiBlackTotal>0)
                {
                   uiBlackTotal--;
                }
                else  //时间到
                {
                    ucRedFlag=0;  //红棋和黑棋同时停止计时
                   ucBlackFlag=0;
                   ucWd=1;  //切换到第一个窗口的状态
                   uiVoiceCnt=const_voice_long; //报警声音触发
                }

        ucDisplayUpdate=1;  //更新显示
```

```
        }
  }
}
void display_service()  //放在定时中断里的显示应用程序
{
  if(ucDisplayUpdate==1)  //有数据更新显示
  {
     ucDisplayUpdate=0;
         switch(ucWd)      //本程序最核心的变量ucWd
         {
           case 1:  //窗口1，代表刚上电或者复位后的状态
                     //红棋分解出分
                      ucDigShowTemp=uiRedTotal/60;
            ucDigShow8=ucDigShowTemp/10;
            ucDigShow7=ucDigShowTemp%10;
                      //红棋分解出秒
                    ucDigShowTemp=uiRedTotal%60;
            ucDigShow6=ucDigShowTemp/10;
            ucDigShow5=ucDigShowTemp%10;
                       ucDigDot7=1;  //数码管7的小数点显示
                     //黑棋分解出分
                      ucDigShowTemp=uiBlackTotal/60;
            ucDigShow4=ucDigShowTemp/10;
            ucDigShow3=ucDigShowTemp%10;
                      //黑棋分解出秒
                    ucDigShowTemp=uiBlackTotal%60;
            ucDigShow2=ucDigShowTemp/10;
            ucDigShow1=ucDigShowTemp%10;
                       ucDigDot3=1;  //数码管3的小数点显示
            led_dr=1;  //计时器处于停止状态,LED亮
                break;
           case 2:  //窗口2，代表黑棋正在运行中的状态
                     //红棋全部不显示
            ucDigShow8=10;
            ucDigShow7=10;
            ucDigShow6=10;
            ucDigShow5=10;
                       ucDigDot7=0;  //数码管7的小数点不显示
                     //黑棋分解出分
                      ucDigShowTemp=uiBlackTotal/60;
            ucDigShow4=ucDigShowTemp/10;
            ucDigShow3=ucDigShowTemp%10;
                      //黑棋分解出秒
                    ucDigShowTemp=uiBlackTotal%60;
            ucDigShow2=ucDigShowTemp/10;
            ucDigShow1=ucDigShowTemp%10;
                       ucDigDot3=1;  //数码管3的小数点显示
            led_dr=0;  //计时器处于计时状态,LED灭
                break;
           case 3:  //窗口3，代表黑棋在中途暂停的状态
```

```
            //红棋全部不显示
 ucDigShow8=10;
 ucDigShow7=10;
 ucDigShow6=10;
 ucDigShow5=10;
              ucDigDot7=0;  //数码管7的小数点不显示
            //黑棋分解出分
             ucDigShowTemp=uiBlackTotal/60;
 ucDigShow4=ucDigShowTemp/10;
 ucDigShow3=ucDigShowTemp%10;
             //黑棋分解出秒
          ucDigShowTemp=uiBlackTotal%60;
 ucDigShow2=ucDigShowTemp/10;
 ucDigShow1=ucDigShowTemp%10;
              ucDigDot3=1;  //数码管3的小数点显示
 led_dr=1;  //计时器处于暂停状态,LED亮
     break;
case 4:  //窗口4，代表红棋正在运行中的状态
            //红棋分解出分
             ucDigShowTemp=uiRedTotal/60;
 ucDigShow8=ucDigShowTemp/10;
 ucDigShow7=ucDigShowTemp%10;
             //红棋分解出秒
          ucDigShowTemp=uiRedTotal%60;
 ucDigShow6=ucDigShowTemp/10;
 ucDigShow5=ucDigShowTemp%10;
              ucDigDot7=1;  //数码管7的小数点显示
            //黑棋全部不显示
 ucDigShow4=10;
 ucDigShow3=10;
 ucDigShow2=10;
 ucDigShow1=10;
              ucDigDot3=0;  //数码管3的小数点不显示
 led_dr=0;  //计时器处于倒计时状态,LED灭
     break;
case 5:  //窗口5，代表红棋在中途暂停的状态
            //红棋分解出分
             ucDigShowTemp=uiRedTotal/60;
 ucDigShow8=ucDigShowTemp/10;
 ucDigShow7=ucDigShowTemp%10;
             //红棋分解出秒
          ucDigShowTemp=uiRedTotal%60;
 ucDigShow6=ucDigShowTemp/10;
 ucDigShow5=ucDigShowTemp%10;
              ucDigDot7=1;  //数码管7的小数点显示
            //黑棋全部不显示
 ucDigShow4=10;
 ucDigShow3=10;
 ucDigShow2=10;
 ucDigShow1=10;
```

```
                        ucDigDot3=0;  //数码管3的小数点不显示
            led_dr=1;  //计时器处于暂停状态,LED亮
                break;
        }
  }
}
void display_drive()  //放在定时中断里的数码管驱动函数
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
           dig_hc595_drive(ucDigShowTemp,0xf7);
               break;
      case 5:  //显示第5位
           ucDigShowTemp=dig_table[ucDigShow5];
           dig_hc595_drive(ucDigShowTemp,0xef);
               break;
      case 6:  //显示第6位
           ucDigShowTemp=dig_table[ucDigShow6];
           dig_hc595_drive(ucDigShowTemp,0xdf);
               break;
      case 7:  //显示第7位
           ucDigShowTemp=dig_table[ucDigShow7];
                   if(ucDigDot7==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
           }
           dig_hc595_drive(ucDigShowTemp,0xbf);
               break;
      case 8:  //显示第8位
           ucDigShowTemp=dig_table[ucDigShow8];
           dig_hc595_drive(ucDigShowTemp,0x7f);
```

```
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
/* 注释二:
*  注意，此处的延时delay_short必须尽可能小，否则动态扫描数码管的速度就不够。
*/
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
```

```
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucLedStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   hc595_st_dr=1;
   delay_short(1);
   hc595_sh_dr=0;    //拉低，抗干扰就增强
   hc595_st_dr=0;
   hc595_ds_dr=0;
}
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
     case 1:   //按键扫描输出第ucRowRecord列低电平
               if(ucRowRecord==1)  //第一列输出低电平
                   {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                   }
                else if(ucRowRecord==2)  //第二列输出低电平
                   {
              key_dr1=1;
```

```
        key_dr2=0;
        key_dr3=1;
        key_dr4=1;
             }
         else if(ucRowRecord==3)  //第三列输出低电平
             {
        key_dr1=1;
        key_dr2=1;
        key_dr3=0;
        key_dr4=1;
             }
         else   //第四列输出低电平
             {
        key_dr1=1;
        key_dr2=1;
        key_dr3=1;
        key_dr4=0;
             }
      uiKeyTimeCnt=0;  //延时计数器清零
      ucKeyStep++;     //切换到下一个运行步骤
          break;
case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
      uiKeyTimeCnt++;
             if(uiKeyTimeCnt>1)
             {
                uiKeyTimeCnt=0;
        ucKeyStep++;     //切换到下一个运行步骤
             }
          break;
case 3:
      if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
      {
        ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
        ucKeyLock=0;  //按键自锁标志清零
        uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                    ucRowRecord++;  //输出下一列
                    if(ucRowRecord>4)
                    {
                       ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                    }
      }
             else if(ucKeyLock==0)  //有按键按下，且是第一次触发
             {
                if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                    {
                       uiKeyTimeCnt++;  //去抖动延时计数器
                           if(uiKeyTimeCnt>const_key_time)
                           {
                              uiKeyTimeCnt=0;
```

```
                    ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
      }
  else   //第四列输出低电平
      {
                ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
      }
          }

    }
else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
    {
       uiKeyTimeCnt++;  //去抖动延时计数器
           if(uiKeyTimeCnt>const_key_time)
           {
              uiKeyTimeCnt=0;
              ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
      }
  else   //第四列输出低电平
      {
                ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
      }
          }

    }
else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
    {
```

```
            uiKeyTimeCnt++;  //去抖动延时计数器
                if(uiKeyTimeCnt>const_key_time)
                {
                   uiKeyTimeCnt=0;
                   ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S10键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
      }
  else   //第四列输出低电平
      {
                ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
      }
           }

    }
else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
    {
       uiKeyTimeCnt++;  //去抖动延时计数器
           if(uiKeyTimeCnt>const_key_time)
           {
              uiKeyTimeCnt=0;
              ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
      }
  else   //第四列输出低电平
      {
                ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
      }
           }
```

```
                        }

                    }
                break;
  }
}
/* 注释三:
*  按键服务程序操作的精髓在于根据当前系统处于什么窗口下就执行什么操作。
*  紧紧围绕着不同的窗口ucWd来执行不同的操作。
*/
void key_service() //第三区 放在定时中断里的按键服务应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键  红棋加分 按键
              switch(ucWd)  //本程序最核心的变量ucWd
              {
                 case 1:  //窗口1，代表刚上电，完成或者复位后的状态
                             uiRedTotal=uiRedTotal+60;  //加红棋分的时间,此处60秒代表一分
                                 if(uiRedTotal>5940)
                                 {
                                    uiRedTotal=5940;
                                 }
                  uiRedTotal=uiRedTotal-(uiRedTotal%60);  //去秒取整分
                                 ucDisplayUpdate=1;  //更新显示
                  uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                      break;
                 case 2:  //窗口2，代表黑棋正在运行中的状态
                      break;
                 case 3:  //窗口3，代表黑棋在中途暂停的状态
                      break;
                 case 4:  //窗口4，代表红棋正在运行中的状态
                      break;
                 case 5:  //窗口5，代表红棋在中途暂停的状态
                      break;
          }
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S2键  红棋减分 按键
              switch(ucWd)  //本程序最核心的变量ucWd
              {
                 case 1:  //窗口1，代表刚上电，完成或者复位后的状态
                             if(uiRedTotal>=60)
                                 {
                                uiRedTotal=uiRedTotal-60;  //减红棋分的时间,此处60秒代表一分
                                 }
                  uiRedTotal=uiRedTotal-(uiRedTotal%60);  //去秒取整分
                                 ucDisplayUpdate=1;  //更新显示
                  uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
```

```
                break;
            case 2:  //窗口2，代表黑棋正在运行中的状态
                break;
            case 3:  //窗口3，代表黑棋在中途暂停的状态
                break;
            case 4:  //窗口4，代表红棋正在运行中的状态
                break;
            case 5:  //窗口5，代表红棋在中途暂停的状态
                break;
        }
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 3号键 对应朱兆祺学习板的S3键  黑棋加分 按键
        switch(ucWd)  //本程序最核心的变量ucWd
        {
           case 1:  //窗口1，代表刚上电，完成或者复位后的状态
                       uiBlackTotal=uiBlackTotal+60;  //加黑棋分的时间,此处60秒代表一分
                           if(uiBlackTotal>5940)
                           {
                              uiBlackTotal=5940;
                           }
            uiBlackTotal=uiBlackTotal-(uiBlackTotal%60);  //去秒取整分
                           ucDisplayUpdate=1;  //更新显示
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                break;
            case 2:  //窗口2，代表黑棋正在运行中的状态
                break;
            case 3:  //窗口3，代表黑棋在中途暂停的状态
                break;
            case 4:  //窗口4，代表红棋正在运行中的状态
                break;
            case 5:  //窗口5，代表红棋在中途暂停的状态
                break;
        }
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 4号键 对应朱兆祺学习板的S4键  黑棋减分 按键
        switch(ucWd)  //本程序最核心的变量ucWd
        {
           case 1:  //窗口1，代表刚上电，完成或者复位后的状态
                       if(uiBlackTotal>=60)
                           {
                          uiBlackTotal=uiBlackTotal-60;  //减黑棋分的时间,此处60秒代表一分
                           }
            uiBlackTotal=uiBlackTotal-(uiBlackTotal%60);  //去秒取整分
                           ucDisplayUpdate=1;  //更新显示
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                break;
            case 2:  //窗口2，代表黑棋正在运行中的状态
                break;
```

```
            case 3:  //窗口3，代表黑棋在中途暂停的状态
                 break;
            case 4:  //窗口4，代表红棋正在运行中的状态
                 break;
            case 5:  //窗口5，代表红棋在中途暂停的状态
                 break;
      }
      ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
      break;
case 5:// 5号键 对应朱兆祺学习板的S5键
      ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
      break;
case 6:// 6号键 对应朱兆祺学习板的S6键  中途暂停和启动按键
          switch(ucWd)  //本程序最核心的变量ucWd
          {
             case 1:  //窗口1，代表刚上电，完成或者复位后的状态
                 break;
             case 2:  //窗口2，代表黑棋正在运行中的状态
              ucRedFlag=0;    //暂停计时
              ucBlackFlag=0;//暂停计时
                             ucWd=3; //切换到黑棋中途暂停的状态
                             ucDisplayUpdate=1;  //更新显示
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                 break;
             case 3:  //窗口3，代表黑棋在中途暂停的状态
              ucRedFlag=0;   //红棋暂停计时
              ucBlackFlag=1; //黑棋继续计时
                             ucWd=2;       //切换到黑棋正在运行中的状态
                             ucDisplayUpdate=1;  //更新显示
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                 break;
             case 4:  //窗口4，代表红棋正在运行中的状态
              ucRedFlag=0;    //暂停计时
              ucBlackFlag=0;//暂停计时
                             ucWd=5;       //切换到红棋中途暂停的状态
                             ucDisplayUpdate=1;  //更新显示
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                 break;
             case 5:  //窗口5，代表红棋在中途暂停的状态
              ucRedFlag=1;   //红棋继续计时
              ucBlackFlag=0; //黑棋暂停计时
                             ucWd=4;       //切换到红棋正在运行中的状态
                             ucDisplayUpdate=1;  //更新显示
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                 break;
      }
      ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
      break;
case 7:// 7号键 对应朱兆祺学习板的S7键  在第一个窗口下，把计时器的值恢复为开机时的默认值20分钟
          switch(ucWd)  //本程序最核心的变量ucWd
```

```
          {
             case 1:  //窗口1，代表刚上电，完成或者复位后的状态
              uiRedTotal=1200;     //红棋的总时间
              uiBlackTotal=1200;  //黑棋的总时间
                             ucDisplayUpdate=1;  //更新显示
                             uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                  break;
             case 2:  //窗口2，代表黑棋正在运行中的状态
                  break;
             case 3:  //窗口3，代表黑棋在中途暂停的状态
                  break;
             case 4:  //窗口4，代表红棋正在运行中的状态
                  break;
             case 5:  //窗口5，代表红棋在中途暂停的状态
                  break;
       }
       ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
       break;
case 8:// 8号键 对应朱兆祺学习板的S8键
       ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
       break;
case 9:// 9号键 对应朱兆祺学习板的S9键
       ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
       break;
case 10:// 10号键 对应朱兆祺学习板的S10键
       ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
       break;
case 11:// 11号键 对应朱兆祺学习板的S11键
       ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
       break;
case 12:// 12号键 对应朱兆祺学习板的S12键
       ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
       break;
case 13:// 13号键 对应朱兆祺学习板的S13键  红棋按下
          switch(ucWd)  //本程序最核心的变量ucWd
          {
             case 1:  //窗口1，代表刚上电，完成或者复位后的状态
              ucRedFlag=0;     //红棋暂停计时
              ucBlackFlag=1;  //黑棋继续计时
                             ucWd=2; //切换到黑棋正在运行中的状态
                             ucDisplayUpdate=1;  //更新显示
                  break;
             case 2:  //窗口2，代表黑棋正在运行中的状态
                  break;
             case 3:  //窗口3，代表黑棋在中途暂停的状态
                  break;
             case 4:  //窗口4，代表红棋正在运行中的状态
              ucRedFlag=0;     //红棋暂停计时
              ucBlackFlag=1;  //黑棋继续计时
                             ucWd=2; //切换到黑棋正在运行中的状态
```

```
                                ucDisplayUpdate=1;  //更新显示
                        break;
                    case 5:  //窗口5，代表红棋在中途暂停的状态
                        break;
            }
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 14:// 14号键 对应朱兆祺学习板的S14键
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 15:// 15号键 对应朱兆祺学习板的S15键
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      case 16:// 16号键 对应朱兆祺学习板的S16键     黑棋按下
                switch(ucWd)  //本程序最核心的变量ucWd
                {
                   case 1:  //窗口1，代表刚上电，完成或者复位后的状态
                    ucRedFlag=1;     //红棋继续计时
                    ucBlackFlag=0;  //黑棋暂停计时
                                ucWd=4; //切换到红棋正在运行中的状态
                                ucDisplayUpdate=1;  //更新显示
                        break;
                   case 2:  //窗口2，代表黑棋正在运行中的状态
                    ucRedFlag=1;     //红棋继续计时
                    ucBlackFlag=0;  //黑棋暂停计时
                                ucWd=4; //切换到红棋正在运行中的状态
                                ucDisplayUpdate=1;  //更新显示
                        break;
                   case 3:  //窗口3，代表黑棋在中途暂停的状态
                        break;
                   case 4:  //窗口4，代表红棋正在运行中的状态
                        break;
                   case 5:  //窗口5，代表红棋在中途暂停的状态
                        break;
            }
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
   }
}
void T0_time() interrupt 1
{
   TF0=0;  //清除中断标志
   TR0=0; //关中断
   key_scan(); //放在定时中断里的按键扫描函数
   time_service();  //放在定时中断里的时间应用程序
   if(uiVoiceCnt!=0)
   {
      uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
          beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
   }
```

```
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
          beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //放在定时中断里的数码管驱动函数
/* 注释四:
*  注意，此处的重装初始值不能太大，否则动态扫描数码管的速度就不够。我把原来常用的2000改成了500。
*/
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=1;
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
}
```

复制代码

总结陈词:

这节讲了象棋比赛专用计时器的项目程序。为了继续加深读者理解按键和显示是如何有规律关联起来的，下节会继续讲一个相关的小项目程序。欲知详情，请听下回分解-----带数码管显示的加法简易计算器。
（未完待续，下节更精彩，不要走开哦）

第三十六节：带数码管显示的加法简易计算器。

开场白：

这一节要做一个简单的计算器。这个计算器不带小数点，只能进行不超过8位数据的加法运算，它麻雀虽小但是五脏俱全，它能清晰地勾勒出商业计算器的程序框架和思路。读者只要看懂本节程序框架的规律，以后自己想做一个复杂一点的计算器应该是没问题的。复杂的计算器在算法上要用数组进行特殊处理，不能简单地直接用C语言的+，-，*,/运算符，这方面的内容我会在以后的章节中跟大家分享。

这一节要教会大家两个知识点：

第一个：数字按键的输入和十进制数值的移位方法。

第二个：继续加深理解按键与数码管的关联程序框架。

具体内容，请看源代码讲解。

（1）硬件平台：

基于朱兆祺51单片机学习板。数字1键对应S1键，数字2键对应S2键，数字3键对应S3键…. 数字9键对应S9键，数字0键对应S10键。加号键对应S13，等于号键对应S14，清除复位按键对应S16。其它按键不用。

（2）实现功能：

常用的加法计算器功能。有连加功能。

本程序有2个窗口。

第1个窗口：原始数据和运算结果窗口。　比如加法运算中的被加数

第2个窗口：第二个参与运行的数据窗口。比如加法运算中的加数

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_voice_long   900   //蜂鸣器长叫的持续时间
#define const_key_time  10    //按键去抖动延时的时间
#define const_1s     422   //产生一秒钟的时间基准
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void T0_time();  //定时中断函数
void key_service();
void key_scan(); //按键扫描函数 放在定时中断里
void number_key_input(unsigned long ucWhichKey);  //由于数字按键的代码相似度高，因此封装在这个函数里
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive();  //放在定时中断里的数码管驱动函数
void display_service();
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5; //LED指示灯
sbit dig_hc595_sh_dr=P2^0;     //数码管 的74HC595程序
sbit dig_hc595_st_dr=P2^1;
```

```
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucDigShow8=0;  //第8位数码管要显示的内容
unsigned char ucDigShow7=0;  //第7位数码管要显示的内容
unsigned char ucDigShow6=0;  //第6位数码管要显示的内容
unsigned char ucDigShow5=0;  //第5位数码管要显示的内容
unsigned char ucDigShow4=0;  //第4位数码管要显示的内容
unsigned char ucDigShow3=0;  //第3位数码管要显示的内容
unsigned char ucDigShow2=0;  //第2位数码管要显示的内容
unsigned char ucDigShow1=0;  //第1位数码管要显示的内容
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucDisplayUpdate=1; //更新显示标志
unsigned long ulSource=0;  //原始数据     比如在加法运算中的被加数
unsigned long ulOther=0; //另外一个参与运算的数据  比如在加法运算中的加数
unsigned long ulResult=0; //运算结果
unsigned char ucOperator=0; //运行符号。0代表当前没有选择运行符号。1代表当前的运算符是加法。
/* 注释一:
*  ucWd变量是本程序最核心的变量，代表数码管显示哪一个窗口
*  本程序只有两个窗口，他们分别是:
*  第一个窗口: 原始数据和运算结果窗口。   比如加法运算中的被加数
*  第二个窗口: 第二个参与运行的数据窗口。比如加法运算中的加数
*/
unsigned char ucWd=1;
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //不显示  序号10
};
void main()
  {
   initial_myself();
   delay_long(100);
```

```
    initial_peripheral();
    while(1)
    {
        key_service();
        display_service();
    }
}
void display_service()  //放在定时中断里的显示应用程序
{
  if(ucDisplayUpdate==1)  //有数据更新显示
  {
     ucDisplayUpdate=0;
         switch(ucWd)     //本程序最核心的变量ucWd
         {
           case 1:  //窗口1  原始数据和运算结果窗口
                if(ulSource>=10000000)
                                {
                                   ucDigShow8=ulSource/10000000;
                                }
                                else
                                {
                               ucDigShow8=10;//数据显示空
                                }
                if(ulSource>=1000000)
                                {
                                   ucDigShow7=ulSource%10000000/1000000;
                                }
                                else
                                {
                               ucDigShow7=10;//数据显示空
                                }
                if(ulSource>=100000)
                                {
                                   ucDigShow6=ulSource%1000000/100000;
                                }
                                else
                                {
                               ucDigShow6=10;//数据显示空
                                }
                if(ulSource>=10000)
                                {
                                   ucDigShow5=ulSource%100000/10000;
                                }
                                else
                                {
                               ucDigShow5=10;//数据显示空
                                }
                if(ulSource>=1000)
                                {
                                   ucDigShow4=ulSource%10000/1000;
```

```
                    }
                    else
                    {
                   ucDigShow4=10;//数据显示空
                    }
    if(ulSource>=100)
                    {
                       ucDigShow3=ulSource%1000/100;
                    }
                    else
                    {
                   ucDigShow3=10;//数据显示空
                    }
    if(ulSource>=10)
                    {
                       ucDigShow2=ulSource%100/10;
                    }
                    else
                    {
                   ucDigShow2=10;//数据显示空
                    }
                    ucDigShow1=ulSource%10;
    break;
case 2:  //窗口2  第二个参与运算数据的窗口  比如加法运算中的加数
    if(ulOther>=10000000)
                    {
                       ucDigShow8=ulOther/10000000;
                    }
                    else
                    {
                   ucDigShow8=10;//数据显示空
                    }
    if(ulOther>=1000000)
                    {
                       ucDigShow7=ulOther%10000000/1000000;
                    }
                    else
                    {
                   ucDigShow7=10;//数据显示空
                    }
    if(ulOther>=100000)
                    {
                       ucDigShow6=ulOther%1000000/100000;
                    }
                    else
                    {
                   ucDigShow6=10;//数据显示空
                    }
    if(ulOther>=10000)
                    {
```

```
                                ucDigShow5=ul0ther%100000/10000;
                            }
                            else
                            {
                           ucDigShow5=10;//数据显示空
                            }
                if(ul0ther>=1000)
                            {
                                ucDigShow4=ul0ther%10000/1000;
                            }
                            else
                            {
                           ucDigShow4=10;//数据显示空
                            }
                if(ul0ther>=100)
                            {
                                ucDigShow3=ul0ther%1000/100;
                            }
                            else
                            {
                           ucDigShow3=10;//数据显示空
                            }
                if(ul0ther>=10)
                            {
                                ucDigShow2=ul0ther%100/10;
                            }
                            else
                            {
                           ucDigShow2=10;//数据显示空
                            }
                            ucDigShow1=ul0ther%10;
                break;
        }
  }
}
void display_drive()  //放在定时中断里的数码管驱动函数
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
```

```
            dig_hc595_drive(ucDigShowTemp,0xfb);
                break;
        case 4:  //显示第4位
            ucDigShowTemp=dig_table[ucDigShow4];
            dig_hc595_drive(ucDigShowTemp,0xf7);
                break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
/* 注释二:
*  注意，此处的延时delay_short必须尽可能小，否则动态扫描数码管的速度就不够。
*/
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
```

```
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
```

```
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;     //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
     case 1:    //按键扫描输出第ucRowRecord列低电平
               if(ucRowRecord==1)  //第一列输出低电平
                   {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==2)  //第二列输出低电平
                   {
              key_dr1=1;
              key_dr2=0;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==3)  //第三列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=0;
              key_dr4=1;
                   }
               else   //第四列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=1;
              key_dr4=0;
                   }
          uiKeyTimeCnt=0;  //延时计数器清零
          ucKeyStep++;      //切换到下一个运行步骤
              break;
     case 2:      //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
          uiKeyTimeCnt++;
                   if(uiKeyTimeCnt>1)
                   {
                      uiKeyTimeCnt=0;
              ucKeyStep++;      //切换到下一个运行步骤
                   }
               break;
```

```
case 3:
    if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
    {
       ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
       ucKeyLock=0;  //按键自锁标志清零
       uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                  ucRowRecord++;  //输出下一列
                  if(ucRowRecord>4)
                  {
                     ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                  }
    }
           else if(ucKeyLock==0)  //有按键按下，且是第一次触发
           {
              if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                  {
                     uiKeyTimeCnt++;  //去抖动延时计数器
                         if(uiKeyTimeCnt>const_key_time)
                         {
                            uiKeyTimeCnt=0;
                            ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                  if(ucRowRecord==1)  //第一列输出低电平
                      {
                                 ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                      }
                  else if(ucRowRecord==2)  //第二列输出低电平
                      {
                                 ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                      }
                  else if(ucRowRecord==3)  //第三列输出低电平
                      {
                                 ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                      }
                  else   //第四列输出低电平
                      {
                                 ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                      }
                         }

                  }
              else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                  {
                     uiKeyTimeCnt++;  //去抖动延时计数器
                         if(uiKeyTimeCnt>const_key_time)
                         {
                            uiKeyTimeCnt=0;
                            ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
```

```
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
      }
  else   //第四列输出低电平
      {
                ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
      }
          }

    }
else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
    {
       uiKeyTimeCnt++;  //去抖动延时计数器
           if(uiKeyTimeCnt>const_key_time)
           {
              uiKeyTimeCnt=0;
              ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零

  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S10键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
      }
  else   //第四列输出低电平
      {
                ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
      }
          }

    }
else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
    {
       uiKeyTimeCnt++;  //去抖动延时计数器
           if(uiKeyTimeCnt>const_key_time)
```

```
                                {
                                    uiKeyTimeCnt=0;
                                    ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                        if(ucRowRecord==1)  //第一列输出低电平
                            {
                                    ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                            }
                        else if(ucRowRecord==2)  //第二列输出低电平
                            {
                                    ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                            }
                        else if(ucRowRecord==3)  //第三列输出低电平
                            {
                                    ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                            }
                        else   //第四列输出低电平
                            {
                                    ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                            }
                                }

                          }

                    }
                break;
  }
}
/* 注释三:
*  按键服务程序操作的精髓在于根据当前系统处于什么窗口下，在此窗口下的运算符处于
*  什么状态，然后紧紧围绕着不同的窗口ucWd，不同的ucOperator来执行不同的操作。
*/
void key_service() //第三区 按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
          number_key_input(1);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 2号键 对应朱兆祺学习板的S2键
          number_key_input(2);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 3号键 对应朱兆祺学习板的S3键
          number_key_input(3);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
```

```
        break;
case 4:// 4号键 对应朱兆祺学习板的S4键
        number_key_input(4);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 5:// 5号键 对应朱兆祺学习板的S5键
        number_key_input(5);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 6:// 6号键 对应朱兆祺学习板的S6键
        number_key_input(6);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 7:// 7号键 对应朱兆祺学习板的S7键
        number_key_input(7);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 8:// 8号键 对应朱兆祺学习板的S8键
        number_key_input(8);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 9:// 9号键 对应朱兆祺学习板的S9键
        number_key_input(9);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 10:// 把这个按键专门用来输入数字0     对应朱兆祺学习板的S10键
        number_key_input(0);  //由于数字按键的代码相似度高，因此把具体代码封装在这个函数里
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 11:// 11号键 对应朱兆祺学习板的S11键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 12:// 12号键 对应朱兆祺学习板的S12键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 13:// 13号键 加号按键  对应朱兆祺学习板的S13键
        switch(ucWd)
               {
                case 1:    //在原始数据和运算结果的窗口下
                 ucOperator=1; //加法
                 ulOther=ulSource;  //第二个运算数默认等于原始数
```

```
                ucDisplayUpdate=1;  //刷新显示窗口
                        break;
               case 2:   //在第二个参与运算数据的窗口下
                ulResult=ulSource+ulOther;//连加
                ulSource=ulResult; //下一次运算的原始数据默认为当前运算结果，方便连加功能
                ucWd=1;        //切换到第一个窗口
                ucDisplayUpdate=1;  //刷新显示窗口
                        break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 14:// 14号键 等于号按键  对应朱兆祺学习板的S14键
        switch(ucWd)
               {
                case 1:   //在原始数据和运算结果的窗口下
                        switch(ucOperator)  //根据不同的运算符号进行不同的操作
                        {
                                case 0:  //无运算符号
                                     break;
                                case 1:  //加法
                         ulResult=ulSource+ulOther;//连加
                         ulSource=ulResult; //下一次运算的原始数据默认为当前运算结果，方便连加功能
                         ucDisplayUpdate=1;  //刷新显示窗口
                                     break;
                                case 2:  //减法  本程序没有减法功能，如果读者想增加减法程序，可以按键
这个框架添加下去
                                     break;

                        }
                              break;
               case 2:   //在第二个参与运算数据的窗口下
                        switch(ucOperator)  //根据不同的运算符号进行不同的操作
                        {
                                case 1:  //加法
                         ulResult=ulSource+ulOther;//连加
                         ulSource=ulResult; //下一次运算的原始数据默认为当前运算结果，方便连加功能
                         ucWd=1;        //切换到第一个窗口
                         ucDisplayUpdate=1;  //刷新显示窗口
                                     break;
                                case 2:  //减法  本程序没有减法功能，如果读者想增加减法程序，可以按键
这个框架添加下去
                                     break;

                        }
                        break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
```

```
    case 15:// 15号键 对应朱兆祺学习板的S15键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 16:// 16号键 清除按键 相当于复位的功能。重新输入数据  对应朱兆祺学习板的S16键
              ulSource=0;
                  ulOther=0;
          ulResult=0;
                  ucOperator=0;
          ucWd=1;         //切换到第一个窗口
          ucDisplayUpdate=1;  //刷新显示窗口
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
/* 注释四:
* 此处参与运算的输入数字ucWhichKey记得用最大变量类型unsigned long，可以避免数据溢出等错误
*/
void number_key_input(unsigned long ucWhichKey)  //由于数字按键的代码相似度高，因此封装在这个函数里
{
    switch(ucWd)
           {
           case 1:   //在原始数据和运算结果的窗口下
            switch(ucOperator)  //根据不同的运算符号进行不同的操作
                        {
                           case 0:  //无运算符号  按键输入原始数据，比如被加输
                                if(ulSource<=9999999) //最大只能输入8位数
                                        {
                              ulSource=ulSource*10+ucWhichKey;  //十进制的数值移位方法。
                                        }
                                break;
                           default:  //在已经按下了运算符号的情况下
                     ulOther=0;  //第二个运算数先清零，再输入新的数据，然后马上切换到第2个窗口下
                     ulOther=ucWhichKey;
                     ucWd=2; //马上切换到第二个窗口下
                                break;

                        }
            ucDisplayUpdate=1;  //刷新显示窗口
                        break;
           case 2:   //在第二个参与运算数据的窗口下   按键输入第二个参与运算的数据
                        if(ulOther<=9999999) //最大只能输入8位数
                        {
                ulOther=ulOther*10+ucWhichKey;  //十进制的数值移位方法。
                     }
            ucDisplayUpdate=1;  //刷新显示窗口
                     break;
    }
}
```

```
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //放在定时中断里的按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //放在定时中断里的数码管驱动函数
/* 注释五:
*  注意，此处的重装初始值不能太大，否则动态扫描数码管的速度就不够。我把原来常用的2000改成了500。
*/
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  led_dr=0;
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
```

```
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;      //开总中断
  ET0=1;     //允许定时中断
  TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节讲了加法简易计算器的程序项目。为了让读者理解运动，按键，显示是如何有规律关联起来的，下节会继续讲一个相关的小项目程序。欲知详情，请听下回分解-----数码管作为仪表盘显示跑马灯的方向，速度和运行状态。

（未完待续，下节更精彩，不要走开哦）

第三十七节: 数码管作为仪表盘显示跑马灯的方向，速度和运行状态。

开场白:

我在第24节中讲过按键控制跑马灯的方向，速度和运行状态的项目程序，只可惜那个程序不能直观地显示运行中的三种状态，这节我决定在24节的基础上，增加一个数码管显示作为类似汽车仪表盘的界面，实时显示跑马灯的方向，速度，和运行状态。

这一节要教会大家一个知识点: 继续加深理解运动，按键与数码管三者之间的关联程序框架。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。用S1键作为控制跑马灯的方向按键，S5键作为控制跑马灯方向的加速度按键，S9键作为控制跑马灯方向的减速度按键，S13键作为控制跑马灯方向的启动或者暂停按键。记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。

（2）实现功能:

跑马灯运行: 第1个至第8个LED灯一直不亮。在第9个至第16个LED灯，依次逐个亮灯并且每次只能亮一个灯。每按一次独立按键S13键，原来运行的跑马灯会暂停，原来暂停的跑马灯会运行。用S1来改变方向。用S5和S9来改变速度，每按一次按键的递增或者递减以10为单位。

数码管显示: 本程序只有1个窗口，这个窗口分成3个局部显示。8，7，6位数码管显示运行状态，启动时显示“on”，停止时显示“oFF”。5位数码管显示数码管方向，正向显示“n”，反向显示“U”。4,3,2,1位数码管显示速度。数值越大速度越慢，最慢的速度是550，最快的速度是50。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
void led_update();  //LED更新函数
```

```
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit led_dr=P3^5;
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;    //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucLed_dr1=0;    //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=0;  //刷新变量。每次更改LED灯的状态都要更新一次。
unsigned char ucLedStep_09_16=0; //第9个至第16个LED跑马灯的步骤变量
unsigned int  uiTimeCnt_09_16=0; //第9个至第16个LED跑马灯的统计定时中断次数的延时计数器
unsigned char ucLedStatus16_09=0;    //代表底层74HC595输出状态的中间变量
unsigned char ucLedStatus08_01=0;    //代表底层74HC595输出状态的中间变量
unsigned char ucLedDirFlag=0;    //方向变量，把按键与跑马灯关联起来的核心变量,0代表正方向，1代表反方向
unsigned int  uiSetTimeLevel_09_16=300;  //速度变量，此数值越大速度越慢，此数值越小速度越快。
unsigned char ucLedStartFlag=1;    //启动和暂停的变量，0代表暂停，1代表启动
```

```
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Part1Update=1;  //窗口1的局部1更新显示变量
unsigned char ucWd1Part2Update=1;  //窗口1的局部2更新显示变量
unsigned char ucWd1Part3Update=1;  //窗口1的局部3更新显示变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
0x5c,  //o       序号13
0x71,  //F       序号14
0x3e,  //U       序号15
0x37,  //n       序号16
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
```

```
      display_service(); //显示的窗口菜单服务程序
      led_flicker_09_16(); //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
          led_update();  //LED更新函数
   }
}
/* 注释一:
* 由于本程序只有1个窗口，而这个窗口又分成3个局部，因此可以省略去窗口变量uWd,
* 只用三个局部变量ucWdxPartyUpdate就可以了。
*/
void display_service() //显示的窗口菜单服务程序
{
    if(ucWd1Part1Update==1) //更新显示当前系统是处于运行还是暂停的状态
        {
       ucWd1Part1Update=0; //及时把更新变量清零，防止一直进来更新
           if(ucLedStartFlag==1)  //启动，显示on
           {
               ucDigShow8=13;  //显示o
           ucDigShow7=16;  //显示n
           ucDigShow6=10;  //显示空
           }
           else  //暂停，显示oFF
           {
                      ucDigShow8=13;  //显示o
           ucDigShow7=14;  //显示F
           ucDigShow6=14;  //显示F
           }
        }
    if(ucWd1Part2Update==1) //更新显示当前系统是处于正方向还是反方向
        {
       ucWd1Part2Update=0; //及时把更新变量清零，防止一直进来更新
           if(ucLedDirFlag==0)  //正方向，向上，显示n
           {
               ucDigShow5=16;  //显示n
           }
           else  //反方向，向下，显示U
           {
               ucDigShow5=15;  //显示U
           }
        }
    if(ucWd1Part3Update==1) //更新显示当前系统的速度，此数值越大速度越慢，此数值越小速度越快。
        {
       ucWd1Part3Update=0; //及时把更新变量清零，防止一直进来更新
           ucDigShow4=10;  //显示空  这一位不用，作为空格
           if(uiSetTimeLevel_09_16>=100)
           {
          ucDigShow3=uiSetTimeLevel_09_16/100;      //显示速度的百位
           }
           else
           {
          ucDigShow3=10;      //显示空
```

```
            }
            if(uiSetTimeLevel_09_16>=10)
            {
           ucDigShow2=uiSetTimeLevel_09_16%100/10;  //显示速度的十位
            }
            else
            {
           ucDigShow2=10;      //显示空
            }
         ucDigShow1=uiSetTimeLevel_09_16%10;       //显示速度的个位
          }

}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
```

```
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;    //触发4号键
     }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 改变跑马灯方向的按键 对应朱兆祺学习板的S1键
          if(ucLedDirFlag==0) //通过中间变量改变跑马灯的方向
                  {
                     ucLedDirFlag=1;
                  }
                  else
                  {
                           ucLedDirFlag=0;
                  }
          ucWd1Part2Update=1; //及时更新显示方向
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

    case 2:// 加速按键 对应朱兆祺学习板的S5键 uiSetTimeLevel_09_16越小速度越快
          uiSetTimeLevel_09_16=uiSetTimeLevel_09_16-10;
                  if(uiSetTimeLevel_09_16<50)  //最快限定在50
                  {
                      uiSetTimeLevel_09_16=50;
                  }
```

```
            ucWd1Part3Update=1; //及时更新显示速度
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 3:// 减速按键 对应朱兆祺学习板的S9键  uiSetTimeLevel_09_16越大速度越慢
            uiSetTimeLevel_09_16=uiSetTimeLevel_09_16+10;
                    if(uiSetTimeLevel_09_16>550)  //最慢限定在550
                    {
                        uiSetTimeLevel_09_16=550;
                    }
            ucWd1Part3Update=1; //及时更新显示速度
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;

        case 4:// 启动和暂停按键 对应朱兆祺学习板的S13键  ucLedStartFlag为0时代表暂停，为1时代表启动
                if(ucLedStartFlag==1)  //启动和暂停两种状态循环切换
                    {
                       ucLedStartFlag=0;
                    }
                    else                    //启动和暂停两种状态循环切换
                    {
                            ucLedStartFlag=1;
                    }
            ucWd1Part1Update=1; //及时更新显示系统的运行状态，是运行还是暂停.
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
      }
}
void led_update()  //LED更新函数
{
   if(ucLed_update==1)
   {
       ucLed_update=0;   //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x02;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfd;
```

```
    }
if(ucLed_dr3==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x04;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xfb;
    }
if(ucLed_dr4==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x08;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xf7;
    }
if(ucLed_dr5==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x10;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xef;
    }
if(ucLed_dr6==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x20;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
       ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
       ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
```

```
    {
       ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
if(ucLed_dr13==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x10;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xef;
    }
if(ucLed_dr14==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x20;
    }
    else
    {
       ucLedStatus16_09=ucLedStatus16_09&0xdf;
    }
if(ucLed_dr15==1)
    {
       ucLedStatus16_09=ucLedStatus16_09|0x40;
```

```
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
    }
}
void display_drive()
{
    //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
    switch(ucDisplayDriveStep)
    {
       case 1:  //显示第1位
            ucDigShowTemp=dig_table[ucDigShow1];
                    if(ucDigDot1==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xfe);
                break;
       case 2:  //显示第2位
            ucDigShowTemp=dig_table[ucDigShow2];
                    if(ucDigDot2==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xfd);
                break;
       case 3:  //显示第3位
            ucDigShowTemp=dig_table[ucDigShow3];
                    if(ucDigDot3==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xfb);
                break;
       case 4:  //显示第4位
            ucDigShowTemp=dig_table[ucDigShow4];
                    if(ucDigDot4==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
```

```
                }
            dig_hc595_drive(ucDigShowTemp,0xf7);
                break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
```

```
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
```

```
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void led_flicker_09_16() //第9个至第16个LED的跑马灯程序，逐个亮并且每次只能亮一个.
{
  if(ucLedStartFlag==1)  //此变量为1时代表启动
  {
     switch(ucLedStep_09_16)
     {
     case 0:
           if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
                  ucLed_dr16=0;  //第16个灭
                  ucLed_dr9=1;  //第9个亮
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=1; //切换到下一个步骤
                           }
                           else  //反方向
                           {
                  ucLed_dr15=1;  //第15个亮
                  ucLed_dr16=0;  //第16个灭
                  ucLed_update=1;  //更新显示
                  ucLedStep_09_16=7; //返回上一个步骤
                           }
           }
           break;
     case 1:
           if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
           {
               uiTimeCnt_09_16=0; //时间计数器清零
                           if(ucLedDirFlag==0)  //正方向
                           {
                  ucLed_dr9=0;  //第9个灭
                  ucLed_dr10=1;  //第10个亮
```

```
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=2; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr16=1;  //第16个亮
            ucLed_dr9=0;  //第9个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=0; //返回上一个步骤
                    }
      }
      break;
case 2:
      if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr10=0;  //第10个灭
            ucLed_dr11=1;  //第11个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=3; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr9=1;  //第9个亮
            ucLed_dr10=0;  //第10个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=1; //返回上一个步骤
                    }
      }
      break;
case 3:
      if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
      {
          uiTimeCnt_09_16=0; //时间计数器清零
                    if(ucLedDirFlag==0)  //正方向
                    {
            ucLed_dr11=0;  //第11个灭
            ucLed_dr12=1;  //第12个亮
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=4; //切换到下一个步骤
                    }
                    else  //反方向
                    {
            ucLed_dr10=1;  //第10个亮
            ucLed_dr11=0;  //第11个灭
            ucLed_update=1;  //更新显示
            ucLedStep_09_16=2; //返回上一个步骤
                    }
```

```
        }
        break;
case 4:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
               ucLed_dr12=0;  //第12个灭
               ucLed_dr13=1;  //第13个亮
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=5; //切换到下一个步骤
                        }
                        else  //反方向
                        {
               ucLed_dr11=1;  //第11个亮
               ucLed_dr12=0;  //第12个灭
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=3; //返回上一个步骤
                        }
        }
        break;
case 5:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
               ucLed_dr13=0;  //第13个灭
               ucLed_dr14=1;  //第14个亮
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=6; //切换到下一个步骤
                        }
                        else  //反方向
                        {
               ucLed_dr12=1;  //第12个亮
               ucLed_dr13=0;  //第13个灭
               ucLed_update=1;  //更新显示
               ucLedStep_09_16=4; //返回上一个步骤
                        }
        }
        break;
case 6:
        if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
        {
            uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
               ucLed_dr14=0;  //第14个灭
               ucLed_dr15=1;  //第15个亮
```

```
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=7; //切换到下一个步骤
                        }
                        else  //反方向
                        {
                ucLed_dr13=1;  //第13个亮
                ucLed_dr14=0;  //第14个灭
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=5; //返回上一个步骤
                        }
            }
            break;
      case 7:
            if(uiTimeCnt_09_16>=uiSetTimeLevel_09_16) //时间到
            {
                uiTimeCnt_09_16=0; //时间计数器清零
                        if(ucLedDirFlag==0)  //正方向
                        {
                ucLed_dr15=0;  //第15个灭
                ucLed_dr16=1;  //第16个亮
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=0; //返回到开始处，重新开始新的一次循环
                        }
                        else  //反方向
                        {
                ucLed_dr14=1;  //第14个亮
                ucLed_dr15=0;  //第15个灭
                ucLed_update=1;  //更新显示
                ucLedStep_09_16=6; //返回上一个步骤
                        }
            }
            break;

      }
   }
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiTimeCnt_09_16<0xffff)  //设定这个条件，防止uiTimeCnt超范围。
  {
      if(ucLedStartFlag==1)  //此变量为1时代表启动
          {
         uiTimeCnt_09_16++;  //累加定时中断的次数，
          }
  }
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
```

```
      uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
      beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//      beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
   }
   else
   {
      ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
      beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//      beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
   }
   display_drive();  //数码管字模的驱动函数
   TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
   TL0=0x0b;
   TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
   key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
   led_dr=0;  //关闭独立LED灯
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
   TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
```

```
{
   ucDigDot8=0;    //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;       //开总中断
   ET0=1;      //允许定时中断
   TR0=1;      //启动定时中断
}
```

复制代码

总结陈词:

前面花了大量的章节在讲数码管显示，按键，运动的关联程序框架，从下一节开始，我将会用八节内容来讲我常用的串口程序框架，内容非常精彩和震撼，思路非常简单而又实用。欲知详情，请听下回分解-----判断数据尾来接收一串数据的串口通用程序框架。

（未完待续，下节更精彩，不要走开哦）

第三十八节: 判断数据尾来接收一串数据的串口通用程序框架。

开场白:

在实际项目中，串口通讯不可能一次通讯只发送或接收一个字节，大部分的项目都是一次发送或者接受一串的数据。我们还要在这一串数据里解析数据协议，提取有用的数据。

这一节要教会大家三个知识点:

第一个: 如何识别一串数据已经发送接收完毕。

第二个: 如何在已经接收到的一串数据中解析数据尾协议并且提取有效数据。

第三个: 接收一串数据的通用程序框架涉及到main循环里的串口服务程序，定时器的计时程序，串口接收中断程序的密切配合。大家要理解它们三者之间是如何关联起来的。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是: 9600 。

通讯协议: XX YY EB 00 55

其中后三位 EB 00 55就是我所说的数据尾，它的有效数据XX YY在数据尾的前面。

任意时刻，单片机从电脑“串口调试助手”上位机收到的一串数据中，只要此数据中包含关键字EB 00 55，并且此关键字前面两个字节的数据XX YY 分别为01 02,那么蜂鸣器鸣叫一声表示接收的数据尾和有效数据都是正确的。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
```

```
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;      //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;     //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
void usart_service(void)  //串口服务程序,在main函数里
{

/* 注释一:
* 识别一串数据是否已经全部接收完了的原理:
* 在规定的时间里，如果没有接收到任何一个字节数据，那么就认为一串数据被接收完了，然后就进入数据协议
* 解析和处理的阶段。这个功能的实现要配合定时中断，串口中断的程序一起阅读，要理解他们之间的关系。
*/
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
           ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
                  //下面的代码进入数据协议解析和数据处理的阶段
                  uiRcMoveIndex=uiRcregTotal; //由于是判断数据尾，所以下标移动变量从数组的最尾端开始向
0移动
           while(uiRcMoveIndex>=5)    //如果处理的数据量大于等于5（2个有效数据，3个数据头）说明还没有把
缓冲区的数据处理完
           {
              if(ucRcregBuf[uiRcMoveIndex-3]==0xeb&&ucRcregBuf[uiRcMoveIndex-
2]==0x00&&ucRcregBuf[uiRcMoveIndex-1]==0x55)  //数据尾eb 00 55的判断
               {
                                 if(ucRcregBuf[uiRcMoveIndex-5]==0x01&&ucRcregBuf[uiRcMoveIndex-4]==0x02)
//有效数据01 02的判断
                                    {
                                        uiVoiceCnt=const_voice_short; //蜂鸣器发出声音，说明数据尾和有效数
据都接收正确
                                    }
                  break;    //退出循环
               }
               uiRcMoveIndex--; //因为是判断数据尾，下标向着0的方向移动
          }

           uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据
```

```
    }

}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
         uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
     ucSendLock=1;      //开自锁标志
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
```

```
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
    EA=1;      //开总中断
    ES=1;      //允许串口中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这一节讲了判断数据尾的程序框架，但是在大部分的项目中，都是通过判断数据头来接收数据的，这样的程序该怎么写？欲知详情，请听下回分解-----判断数据头来接收一串数据的串口通用程序框架。

（未完待续，下节更精彩，不要走开哦）

第三十九节：判断数据头来接收一串数据的串口通用程序框架。

开场白:

上一节讲了判断数据尾的程序框架，但是在大部分的项目中，都是通过判断数据头来接收数据的，这一节要教会大家两个知识点:

第一个：如何在已经接收到的一串数据中解析数据头协议并且提取有效数据。

第二个：无论是判断数据头还是判断数据尾，无论是单片机还是上位机，最好在固定协议前多发送一个填充的无效字节0x00，因为硬件原因，第一个字节往往容易丢失。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是：9600 。

通讯协议：EB 00 55 XXYY

加无效填充字节后,上位机实际上应该发送：00 EB 00 55 XXYY

其中第1位00是无效填充字节，防止由于硬件原因丢失第一个字节。

其中第2,3,4位EB 00 55就是数据头
后2位XXYY就是有效数据
任意时刻，单片机从电脑“串口调试助手”上位机收到的一串数据中，只要此数据中包含关键字EB 00 55 ，并且此关键字后面两个字节的数据XX YY 分别为01 02,那么蜂鸣器鸣叫一声表示接收的数据头和有效数据都是正确的。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
void usart_service(void)  //串口服务程序,在main函数里
{

/* 注释一:
* 识别一串数据是否已经全部接收完了的原理:
* 在规定的时间里，如果没有接收到任何一个字节数据，那么就认为一串数据被接收完了，然后就进入数据协议
* 解析和处理的阶段。这个功能的实现要配合定时中断，串口中断的程序一起阅读，要理解他们之间的关系。
*/
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
                    //下面的代码进入数据协议解析和数据处理的阶段
                    uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
/* 注释二:
* 判断数据头，进入循环解析数据协议必须满足两个条件:
* 第一: 最大接收缓冲数据必须大于一串数据的长度（这里是5。包括2个有效数据，3个数据头）
* 第二: 游标uiRcMoveIndex必须小于等于最大接收缓冲数据减去一串数据的长度（这里是5。包括2个有效数据，3个数
```

```
据头）
*/
            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
               {
                                if(ucRcregBuf[uiRcMoveIndex+3]==0x01&&ucRcregBuf[uiRcMoveIndex+4]==0x02)
//有效数据01 02的判断
                                    {
                                        uiVoiceCnt=const_voice_short; //蜂鸣器发出声音，说明数据头和有效数
据都接收正确
                                    }
                   break;   //退出循环
               }
               uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
           }

           uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

     }

}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这
个中间没有串口数据过来
      ucSendLock=1;     //开自锁标志
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
   if(RI==1)
```

```
    {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

    }
    else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
    {
        TI = 0;
    }

}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
    EA=1;      //开总中断
    ES=1;      //允许串口中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
```

复制代码
总结陈词:

这一节讲了常用的判断数据头来接收一串数据的程序框架，但是在很多项目中，仅仅靠判断数据头还是不够的，必须要有更加详细的通讯协议，比如可以包含数据类型，有效数据长度，有效数据，数据校验的通讯协议。这样的程序该怎么写？欲知详情，请听下回分解-----常用的自定义串口通讯协议。

（未完待续，下节更精彩，不要走开哦）

第四十节：常用的自定义串口通讯协议。

开场白:

上一节讲了判断数据头的程序框架，但是在很多项目中，仅仅靠判断数据头还是不够的，必须要有更加详细的通讯协议，比如可以包含数据类型，数据地址，有效数据长度，有效数据，数据校验的通讯协议。这一节要教会大家三个知识点:

第一个: 常用自定义串口通讯协议的程序框架。

第二个: 累加校验和的校验方法。累加和的意思是前面所有字节的数据相加，超过一个字节的溢出部分会按照固定的规则自动丢弃，不用我们管。比如以下数据:

eb 00 55 01 00 02 0028 6b

其中eb 00 55为数据头，01为数据类型，00 02为有效数据长度，00 28 分别为具体的有效数据，6b为前面所有字节的累加和。累加和可以用电脑系统自带的计算器来验证。打开电脑上的计算器，点击“查看”下拉的菜单，选“科学型”，然后选左边的“十六进制”，最后选右边的“字节”，然后把前面所有的字节相加，它们的和就是6b，没错吧。

第三个: 原子锁的使用方法，实际上是借鉴了"红金龙吸味"关于原子锁的建议，专门用来保护中断与主函数的共享数据。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是: 9600.

通讯协议: EB 00 55 GG HH HH XX XX …YYYY CY

其中第1,2,3位EB 00 55就是数据头

其中第4位GG就是数据类型。01代表驱动奉命，02代表驱动Led灯。

其中第5,6位HH就是有效数据长度。高位在左，低位在右。

其中第5,6位HH就是有效数据长度。高位在左，低位在右。

其中从第7位开始，到最后一个字节Cy之前，XX..YY都是具体的有效数据。

在本程序中，当数据类型是01时，有效数据代表蜂鸣器鸣叫的时间长度。当数据类型是02时，有效数据代表Led灯点亮的时间长度。

最后一个字节CY是累加和，前面所有字节的累加。

发送以下测试数据，将会分别控制蜂鸣器和Led灯的驱动时间长度。

蜂鸣器短叫发送: eb 00 55 01 00 02 00 28 6b

蜂鸣器长叫发送: eb 00 55 01 00 02 00 fa 3d

Led灯短亮发送: eb 00 55 02 00 02 00 28 6c

Led灯长亮发送: eb 00 55 02 00 02 00 fa3e

（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 请评估实际项目中一串数据的最大长度是多少，并且留点余量，然后调整const_rc_size的大小。
* 本节程序把上一节的缓冲区数组大小10改成了20
*/
#define const_rc_size  20  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
```

```
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void led_service(void);  //Led灯的服务程序。
sbit led_dr=P3^5;  //Led的驱动IO口
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
/* 注释二:
* 为串口计时器多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建
议.
*/
unsigned char  ucSendCntLock=0; //串口计时器的原子锁
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucRcType=0;  //数据类型
unsigned int  uiRcSize=0;  //数据长度
unsigned char ucRcCy=0;  //校验累加和
unsigned int  uiRcVoiceTime=0;  //蜂鸣器发出声音的持续时间
unsigned int  uiRcLedTime=0; //在串口服务程序中，Led灯点亮时间长度的中间变量
unsigned int  uiLedTime=0;  //Led灯点亮时间的长度
unsigned int  uiLedCnt=0;   //Led灯点亮的计时器
unsigned char ucLedLock=0;  //Led灯点亮时间的原子锁
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
       led_service(); //Led灯的服务程序
   }
}
void led_service(void)
{
   if(uiLedCnt<uiLedTime)
   {
      led_dr=1; //开Led灯
   }
   else
   {
      led_dr=0; //关Led灯
   }
}
```

```
void usart_service(void)  //串口服务程序,在main函数里
{
/* 注释三:
* 我借鉴了朱兆祺的变量命名习惯，单个字母的变量比如i,j,k,h，这些变量只用作局部变量，直接在函数内部定义。
*/
    unsigned int i;

    if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
    {
        ucSendLock=0;    //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
        while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
        {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
            {
                ucRcType=ucRcregBuf[uiRcMoveIndex+3];   //数据类型  一个字节
                    uiRcSize=ucRcregBuf[uiRcMoveIndex+4];   //数据长度  两个字节
                    uiRcSize=uiRcSize<<8;
                    uiRcSize=uiRcSize+ucRcregBuf[uiRcMoveIndex+5];

                    ucRcCy=ucRcregBuf[uiRcMoveIndex+6+uiRcSize];   //记录最后一个字
节的校验
                ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=0;  //清零最后一个字节的累加和变量
/* 注释四:
* 计算校验累加和的方法:除了最后一个字节，其它前面所有的字节累加起来，
* 溢出的不用我们管,C语言编译器会按照固定的规则自动处理。
* 以下for循环里的(3+1+2+uiRcSize)，其中3代表3个字节数据头，1代表1个字节数据类型，
* 2代表2个字节的数据长度变量，uiRcSize代表实际上一串数据中的有效数据个数。
*/
                    for(i=0;i<(3+1+2+uiRcSize);i++) //计算校验累加和
                    {
ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=ucRcregBuf[uiRcMoveIndex+6+uiRcSize]+ucRcregBuf[i];
                }
                    if(ucRcCy==ucRcregBuf[uiRcMoveIndex+6+uiRcSize])  //如果校验正
确，则进入以下数据处理
                    {
                switch(ucRcType)   //根据不同的数据类型来做不同的数据处理
                    {
                    case 0x01:   //驱动蜂鸣器发出声音，并且可以控制蜂鸣器持续发出声音的时间长度

                        uiRcVoiceTime=ucRcregBuf[uiRcMoveIndex+6];  //把两个字节合并成一个
int类型的数据
                        uiRcVoiceTime=uiRcVoiceTime<<8;
                        uiRcVoiceTime=uiRcVoiceTime+ucRcregBuf[uiRcMoveIndex+7];
                        ucVoiceLock=1;  //共享数据的原子锁加锁
                        uiVoiceCnt=uiRcVoiceTime; //蜂鸣器发出声音
                        ucVoiceLock=0;  //共享数据的原子锁解锁
                                    break;
```

```
                                case 0x02:    //点亮一个LED灯，并且可以控制LED灯持续亮的时间长度
                                    uiRcLedTime=ucRcregBuf[uiRcMoveIndex+6];  //把两个字节合并成一个int类
型的数据
                                    uiRcLedTime=uiRcLedTime<<8;
                                    uiRcLedTime=uiRcLedTime+ucRcregBuf[uiRcMoveIndex+7];
                                    ucLedLock=1;  //共享数据的原子锁加锁
                                    uiLedTime=uiRcLedTime; //更改点亮Led灯的时间长度
                                                                    uiLedCnt=0;  //在本程序中，清零计数器
就等于自动点亮Led灯
                                    ucLedLock=0;  //共享数据的原子锁解锁
                                                                break;

                            }
                                        }
                        break;    //退出循环
                    }
                    uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
                }

                uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

            }

}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释五:
  * 此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建议.
  */
  if(ucSendCntLock==0)  //原子锁判断
  {
     ucSendCntLock=1; //加锁
     if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
     {
        uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个
中间没有串口数据过来
        ucSendLock=1;      //开自锁标志
     }
     ucSendCntLock=0; //解锁
  }
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
```

```
    }
    else
    {
       ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
       beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

    }
  }
  if(ucLedLock==0)  //原子锁判断
  {
    if(uiLedCnt<uiLedTime)
        {
           uiLedCnt++;  //Led灯点亮的时间计时器
        }
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                    //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;   //将串口接收到的数据缓存到接收缓冲区里
        if(ucSendCntLock==0)  //原子锁判断
        {
            ucSendCntLock=1; //加锁
            uiSendCnt=0;  //及时喂狗，虽然在定时中断那边此变量会不断累加，但是只要串口的数据还没发送完毕
，那么它永远也长不大,因为每个串口接收中断它都被清零。
                    ucSendCntLock=0; //解锁
                }

   }
   else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
```

```
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  led_dr=0; //关Led灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这一节讲了常用的自定义串口通讯协议的程序框架，这种框架在判断一串数据是否接收完毕的时候，都是靠“超过规定的时间内，没有发现串口数据”来判定的，这是我做绝大多数项目的串口程序框架，但是在少数要求实时反应非常快的项目中，我会用另外一种响应速度更快的串口程序框架，这种程序框架是什么样的？欲知详情，请听下回分解-----在串口接收中断里即时解析数据头的特殊程序框架。

（未完待续，下节更精彩，不要走开哦）

第四十一节：在串口接收中断里即时解析数据头的特殊程序框架。

开场白:

上一节讲了常用的自定义串口通讯协议的程序框架，这种框架在判断一串数据是否接收完毕的时候，都是靠“超过规定的时间内，没有发现串口数据”来判定的，这是我做绝大多数项目的串口程序框架，但是在少数要求实时反应非常快的项目中，这样的程序框架可能会满足不了系统对速度的要求，这一节就是要介绍另外一种相应速度更加快的串口程序框架，要教会大家一个知识点：在串口接收中断里即时解析数据头的特殊程序框架。我在这种程序框架里，会尽量简化数据头和数据尾，同时也简化校验，目的都是为了提高相应速度。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

  波特率是：9600.

通讯协议：EB GG XX XX XX XX ED

其中第1位EB就是数据头.

其中第2位GG就是数据类型。01代表驱动蜂鸣器，02代表驱动Led灯。

其中第3,4,5,6位XX就是有效数据长度。高位在左，低位在右。
其中第7位ED就是数据尾，在这里也起一部分校验的作用，虽然不是累加和的方式。
在本程序中，
当数据类型是01时，4个有效数据代表一个long类型数据，如果这个数据等于十进制的123456789，那么蜂鸣器就鸣叫一声表示正确。
当数据类型是02时，4个有效数据代表一个long类型数据，如果这个数据等于十进制的123456789，那么LED灯就会闪烁一下表示正确。
十进制的123456789等于十六进制的75bcd15 。
发送以下测试数据，将会分别控制蜂鸣器Led灯。
控制蜂鸣器发送: eb 01 07 5b cd 15 ed
控制LED灯发送: eb 02 07 5b cd 15 ed
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_rc_size  20  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define const_voice_short  80   //蜂鸣器短叫的持续时间
#define const_led_short  80    //LED灯亮的持续时间
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void led_service(void);  //Led灯的服务程序。
sbit led_dr=P3^5;  //Led的驱动IO口
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned int  uiRcVoiceTime=0;  //蜂鸣器发出声音的持续时间
unsigned int  uiLedCnt=0;   //Led灯点亮的计时器
unsigned char ucLedLock=0;  //Led灯点亮时间的原子锁
unsigned long ulBeepData=0; //蜂鸣器的数据
unsigned long ulLedData=0; //LED的数据
unsigned char ucUsartStep=0; //串口接收字节的步骤变量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       led_service(); //Led灯的服务程序
   }
}
void led_service(void)
{
   if(uiLedCnt<const_led_short)
   {
```

```
        led_dr=1; //开Led灯
    }
    else
    {
        led_dr=0; //关Led灯
    }
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释一:
  * 此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建议.
  */
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  if(ucLedLock==0)  //原子锁判断
  {
     if(uiLedCnt<const_led_short)
     {
            uiLedCnt++;  //Led灯点亮的时间计时器
     }
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
/* 注释二:
  * 以下就是吴坚鸿在串口接收中断里即时解析数据头的特殊程序框架，
  * 它的特点是靠数据头来启动接受有效数据，靠数据尾来识别一串数据接受完毕，
  * 这里的数据尾也起到一部分的校验作用，让数据更加可靠。这种程序结构适合应用
  * 在传输的数据长度不是很长，而且要求响应速度非常高的实时场合。在这种实时要求
  * 非常高的场合中，我就不像之前一样做数据累加和的复杂运算校验，只用数据尾来做简单的
  * 校验确认，目的是尽可能提高处理速度。
  */
```

```
if(RI==1)
{
    RI = 0;
    switch(ucUsartStep) //串口接收字节的步骤变量
    {
        case 0:
                        ucRcregBuf[0]=SBUF;
            if(ucRcregBuf[0]==0xeb)  //数据头判断
            {
                            ucRcregBuf[0]=0;  //数据头及时清零，为下一串数据的接受判断做准备
                            uiRcregTotal=1;  //缓存数组的下标初始化
                ucUsartStep=1;  //如果数据头正确，则切换到下一步，依次把上位机来的数据存入数组缓冲区
            }
            break;
        case 1:
                        ucRcregBuf[uiRcregTotal]=SBUF;  //依次把上位机来的数据存入数组缓冲区
                uiRcregTotal++; //下标移动
                if(uiRcregTotal>=7)  //已经接收了7个字节
                {
              if(ucRcregBuf[6]==0xed)  //数据尾判断，也起到一部分校验的作用，让数据更加可靠,虽然没有用到累加和的检验方法
                  {
                    ucRcregBuf[6]=0;  //数据尾及时清零，为下一串数据的接受判断做准备

                    switch(ucRcregBuf[1]) //根据不同的数据类型来做不同的数据处理
                        {
                            case 0x01:  //与蜂鸣器相关
                      ulBeepData=ucRcregBuf[2]; //把四个字节的数据合并成一个long型的数据
                                ulBeepData=ulBeepData<<8;
                                ulBeepData=ulBeepData+ucRcregBuf[3];
                                ulBeepData=ulBeepData<<8;
                                ulBeepData=ulBeepData+ucRcregBuf[4];
                                ulBeepData=ulBeepData<<8;
                                ulBeepData=ulBeepData+ucRcregBuf[5];
                                    if(ulBeepData==123456789)  //如果此数据等于十进制的123456789，表示数据正确
                                    {
                                        ucVoiceLock=1;  //共享数据的原子锁加锁
                                uiVoiceCnt=const_voice_short; //蜂鸣器发出声音
                                ucVoiceLock=0;  //共享数据的原子锁解锁
                                    }
                                break;
                            case 0x02:  //与Led灯相关
                      ulLedData=ucRcregBuf[2]; //把四个字节的数据合并成一个long型的数据
                                ulLedData=ulLedData<<8;
                                ulLedData=ulLedData+ucRcregBuf[3];
                                ulLedData=ulLedData<<8;
                                ulLedData=ulLedData+ucRcregBuf[4];
```

```
                                                    ulLedData=ulLedData<<8;
                                                    ulLedData=ulLedData+ucRcregBuf[5];
                                                        if(ulLedData==123456789)  //如果此数据等
于十进制的123456789，表示数据正确
                                                        {
                                                            ucLedLock=1;  //共享数据的原子锁加锁
                                        uiLedCnt=0;  //在本程序中，清零计数器就等于自动点亮Led灯
                                        ucLedLock=0;  //共享数据的原子锁解锁
                                                        }
                                                    break;
                                                }
                                            }
                                ucUsartStep=0;      //返回上一步数据头判断，为下一次的新数据接收做准备
                                        }
                            break;
                    }

    }
    else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
    {
         TI = 0;
    }

}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  led_dr=0; //关Led灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
```

```
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

前面花了4节内容仔细讲了各种串口接收数据的常用框架，从下一节开始，我开始讲串口发送数据的程序框架，这种程序框架是什么样的？欲知详情，请听下回分解-----通过串口用delay延时方式发送一串数据。

（未完待续，下节更精彩，不要走开哦）

第

四十二节：通过串口用delay延时方式发送一串数据。

开场白:

上一节讲了在串口接收中断里即时解析数据头的特殊程序框架。这节开始讲串口发送数据需要特别注意的地方和程序框架，要教会大家一个知识点：根据我个人的经验，在发送一串数据中，每个字节之间必须添加一个延时，用来等待串口发送完成。当然，也有一些朋友可能不增加延时，直接靠单片机自带的发送完成标志位来判断，但是我以前在做项目中，感觉单单靠发送完成标志位来判断还是容易出错（当然也有可能是我自身程序的问题），所以后来在大部分的项目中我就干脆靠延时来等待它发送完成。我在51，PIC单片机中都是这么做的。但是，凭我的经验，在stm32单片机中，可以不增加延时，直接靠单片机自带的标志位来判断就很可靠。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是：9600.

按一次按键S1，单片机就往上位机发送以下一串数据:

eb 00 55 01 00 00 00 00 41

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_send_size  10  //串口发送数据的缓冲区数组大小
#define const_key_time1  20    //按键去抖动延时的时间
#define const_voice_short  40   //蜂鸣器短叫的持续时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayshort);
void delay_long(unsigned int uiDelaylong);
void eusart_send(unsigned char ucSendData);  //发送一个字节，内部自带每个字节之间的延时
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void key_service(); //按键服务的应用程序
void key_scan(); //按键扫描函数 放在定时中断里
sbit led_dr=P3^5;  //Led的驱动IO口
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
unsigned char ucSendregBuf[const_send_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucKeySec=0;   //被触发的按键编号
```

```
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
void main()
{
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
   }
}
void eusart_send(unsigned char ucSendData)
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
/* 注释一:
  * 根据我个人的经验，在发送一串数据中，每个字节之间必须添加一个延时，用来等待串口发送完成。
  * 当然，也有一些朋友可能不增加延时，直接靠单片机自带的发送完成标志位来判断，但是我以前
  * 在做项目中，感觉单单靠发送完成标志位来判断还是容易出错（当然也有可能是我自身程序的问题），
  * 所以后来在大部分的项目中我就干脆靠延时来等待它发送完成。我在51，PIC单片机中都是这么做的。
  * 但是，凭我的经验，在stm32单片机中，可以不增加延时，直接靠单片机自带的标志位来判断就很可靠。
  */
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目
来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void key_scan()//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
}
void key_service() //第三区 按键服务的应用程序
{
  unsigned int i;
```

```
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 1号键 对应朱兆祺学习板的S1键
          ucSendregBuf[0]=0xeb;     //把准备发送的数据放入发送缓冲区
          ucSendregBuf[1]=0x00;
          ucSendregBuf[2]=0x55;
          ucSendregBuf[3]=0x01;
          ucSendregBuf[4]=0x00;
          ucSendregBuf[5]=0x00;
          ucSendregBuf[6]=0x00;
          ucSendregBuf[7]=0x00;
          ucSendregBuf[8]=0x41;
                  for(i=0;i<9;i++)
                  {
                     eusart_send(ucSendregBuf[i]);  //发送一串数据给上位机
                  }
          ucVoiceLock=1;  //原子锁加锁，保护中断与主函数的共享数据
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                  ucVoiceLock=0; //原子锁解锁
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释二:
  * 此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建议.
  */
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  key_scan();//按键扫描函数
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
```

```
void usart_receive(void) interrupt 4                    //串口中断
{
   if(RI==1)
   {
        RI = 0;    //接收中断，及时把接收中断标志位清零


   }
   else
   {
        TI = 0;     //发送中断，及时把发送中断标志位清零
   }

}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;    //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
/* 注释三:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0; //关Led灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
```

```
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //串口波特率9600。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节在每个字节之间都添加了delay延时来等待每个字节的发送完成，由于delay(400)这个时间还不算很长，所以可以应用在很多简单任务的系统中。但是在某些任务量很多的系统中，实时运行的主任务不允许被长时间和经常性地中断，这个时候就需要用计数延时来替代delay延时，这种程序框架是什么样的？欲知详情，请听下回分解-----通过串口用计数延时方式发送一串数据。

（未完待续，下节更精彩，不要走开哦）

第四十三节：通过串口用计数延时方式发送一串数据。

开场白:

上一节讲了通过串口用delay延时方式发送一串数据，这种方式要求发送一串数据的时候一气呵成，期间不能执行其它任务，由于delay(400)这个时间还不算很长，所以可以应用在很多简单任务的系统中。但是在某些任务量很多的系统中，实时运行的主任务不允许被长时间和经常性地中断，这个时候就需要用计数延时来替代delay延时。本节要教会大家两个知识点:

第一个：用计数延时方式发送一串数据的程序框架。

第二个：环形消息队列的程序框架。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是：9600.

用朱兆祺51单片机学习板中的S1,S5,S9,S13作为独立按键。

按一次按键S1，发送EB 00 55 01 00 00 00 00 41

按一次按键S5，发送EB 00 55 02 00 00 00 00 42

按一次按键S9，发送EB 00 55 03 00 00 00 00 43

按一次按键S13，发送EB 00 55 04 00 00 00 00 44

（3）源代码讲解如下:

```
#include REG52.H
#define const_send_time  100  累计主循环次数的计数延时 请根据项目实际情况来调整此数据大小
#define const_send_size  10  串口发送数据的缓冲区数组大小
#define const_Message_size  10  环形消息队列的缓冲区数组大小
#define const_key_time1  20    按键去抖动延时的时间
#define const_key_time2  20    按键去抖动延时的时间
#define const_key_time3  20    按键去抖动延时的时间
#define const_key_time4  20    按键去抖动延时的时间
#define const_voice_short  40   蜂鸣器短叫的持续时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayshort);
void delay_long(unsigned int uiDelaylong);
```

```
void eusart_send(unsigned char ucSendData);  发送一个字节，内部没有每个字节之间的延时
void send_service(void);  利用累计主循环次数的计数延时方式来发送一串数据
void T0_time(void);  定时中断函数
void usart_receive(void); 串口接收中断函数
void key_service(void); 按键服务的应用程序
void key_scan(void); 按键扫描函数 放在定时中断里
void insert_message(unsigned char ucMessageTemp);  插入新的消息到环形消息队列里
unsigned char get_message(void);  从环形消息队列里提取消息
sbit led_dr=P3^5;  Led的驱动IO口
sbit beep_dr=P2^7; 蜂鸣器的驱动IO口
sbit key_sr1=P0^0; 对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; 对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; 对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; 对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; 模拟独立按键的地GND，因此必须一直输出低电平
unsigned char ucSendregBuf[const_send_size]; 串口发送数据的缓冲区数组
unsigned char ucMessageBuf[const_Message_size]; 环形消息队列的缓冲区数据
unsigned int  uiMessageCurrent=0;  环形消息队列的取数据当前位置
unsigned int  uiMessageInsert=0;  环形消息队列的插入新消息时候的位置
unsigned int  uiMessageCnt=0;  统计环形消息队列的消息数量  等于0时表示消息队列里没有消息
unsigned char ucMessage=0; 当前获取到的消息
unsigned int  uiVoiceCnt=0;  蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  蜂鸣器鸣叫的原子锁
unsigned char ucKeySec=0;   被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; 按键去抖动延时计数器
unsigned char ucKeyLock1=0; 按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; 按键去抖动延时计数器
unsigned char ucKeyLock2=0; 按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; 按键去抖动延时计数器
unsigned char ucKeyLock3=0; 按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; 按键去抖动延时计数器
unsigned char ucKeyLock4=0; 按键触发后自锁的变量标志
unsigned char ucSendStep=0;  发送一串数据的运行步骤
unsigned int  uiSendTimeCnt=0; 累计主循环次数的计数延时器
unsigned int uiSendCnt=0; 发送数据时的中间变量
void main()
{
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); 按键服务的应用程序
          send_service();  利用累计主循环次数的计数延时方式来发送一串数据
   }
}
```

注释一:

通过判断数组下标是否超范围的条件，把一个数组的首尾连接起来，就像一个环形，
因此命名为环形消息队列。环形消息队列有插入消息，获取消息两个核心函数，以及一个
统计消息总数的uiMessageCnt核心变量，通过此变量，我们可以知道消息队列里面是否有消息需要处理.

我在做项目中很少用消息队列的，印象中我只在两个项目中用过消息队列这种方法。大部分的单片机项目其实直接用一两个中间变量就可以起到传递消息的作用，就能满足系统的要求。以下是各变量的含义:

```
#define const_Message_size  10  环形消息队列的缓冲区数组大小
unsigned char ucMessageBuf[const_Message_size]; 环形消息队列的缓冲区数据
unsigned int  uiMessageCurrent=0;  环形消息队列的取数据当前位置
unsigned int  uiMessageInsert=0;  环形消息队列的插入新消息时候的位置
unsigned int  uiMessageCnt=0;  统计环形消息队列的消息数量  等于0时表示消息队列里没有消息

void insert_message(unsigned char ucMessageTemp)  插入新的消息到环形消息队列里
{
   if(uiMessageCntconst_Message_size)  消息总数小于环形消息队列的缓冲区才允许插入新消息
   {
      ucMessageBuf[uiMessageInsert]=ucMessageTemp;
          uiMessageInsert++;  插入新消息时候的位置
          if(uiMessageInsert=const_Message_size) 到了缓冲区末尾，则从缓冲区的开头重新开始。数组的首尾连接，看起来就像环形
          {
             uiMessageInsert=0;
          }
      uiMessageCnt++; 消息数量累加  等于0时表示消息队列里没有消息
   }
}
unsigned char get_message(void)  从环形消息队列里提取消息
{
   unsigned char ucMessageTemp=0;  返回的消息中间变量，默认为0
   if(uiMessageCnt0)  只有消息数量大于0时才可以提取消息
   {
      ucMessageTemp=ucMessageBuf[uiMessageCurrent];
          uiMessageCurrent++;  环形消息队列的取数据当前位置
          if(uiMessageCurrent=const_Message_size) 到了缓冲区末尾，则从缓冲区的开头重新开始。数组的首尾连接，看起来就像环形
          {
             uiMessageCurrent=0;
          }
      uiMessageCnt--; 每提取一次，消息数量就减一  等于0时表示消息队列里没有消息
   }
   return ucMessageTemp;
}
void send_service(void)  利用累计主循环次数的计数延时方式来发送一串数据
{
  switch(ucSendStep)  发送一串数据的运行步骤
  {
    case 0   从环形消息队列里提取消息
         if(uiMessageCnt0)  说明有消息需要处理
                {
                   ucMessage=get_message();
            switch(ucMessage)   消息处理
                        {
                           case 1
                   ucSendregBuf[0]=0xeb;     把准备发送的数据放入发送缓冲区
```

```
ucSendregBuf[1]=0x00;
ucSendregBuf[2]=0x55;
ucSendregBuf[3]=0x01;     01代表1号键
ucSendregBuf[4]=0x00;
ucSendregBuf[5]=0x00;
ucSendregBuf[6]=0x00;
ucSendregBuf[7]=0x00;
ucSendregBuf[8]=0x41;
uiSendCnt=0; 发送数据的中间变量清零
uiSendTimeCnt=0; 累计主循环次数的计数延时器清零
ucSendStep=1; 切换到下一步发送一串数据
            break;
       case 2
ucSendregBuf[0]=0xeb;     把准备发送的数据放入发送缓冲区
ucSendregBuf[1]=0x00;
ucSendregBuf[2]=0x55;
ucSendregBuf[3]=0x02;     02代表2号键
ucSendregBuf[4]=0x00;
ucSendregBuf[5]=0x00;
ucSendregBuf[6]=0x00;
ucSendregBuf[7]=0x00;
ucSendregBuf[8]=0x42;
uiSendCnt=0; 发送数据的中间变量清零
uiSendTimeCnt=0; 累计主循环次数的计数延时器清零
ucSendStep=1; 切换到下一步发送一串数据
            break;
       case 3
ucSendregBuf[0]=0xeb;     把准备发送的数据放入发送缓冲区
ucSendregBuf[1]=0x00;
ucSendregBuf[2]=0x55;
ucSendregBuf[3]=0x03;     03代表3号键
ucSendregBuf[4]=0x00;
ucSendregBuf[5]=0x00;
ucSendregBuf[6]=0x00;
ucSendregBuf[7]=0x00;
ucSendregBuf[8]=0x43;
uiSendCnt=0; 发送数据的中间变量清零
uiSendTimeCnt=0; 累计主循环次数的计数延时器清零
ucSendStep=1; 切换到下一步发送一串数据
            break;
       case 4
ucSendregBuf[0]=0xeb;     把准备发送的数据放入发送缓冲区
ucSendregBuf[1]=0x00;
ucSendregBuf[2]=0x55;
ucSendregBuf[3]=0x04;     04代表4号键
ucSendregBuf[4]=0x00;
ucSendregBuf[5]=0x00;
ucSendregBuf[6]=0x00;
ucSendregBuf[7]=0x00;
ucSendregBuf[8]=0x44;
```

```
                    uiSendCnt=0; 发送数据的中间变量清零
                    uiSendTimeCnt=0; 累计主循环次数的计数延时器清零
                    ucSendStep=1; 切换到下一步发送一串数据
                            break;
              default  如果没有符合要求的消息，则不处理
                            ucSendStep=0; 维持现状，不切换
                            break;
                        }
                  }
             break;
    case 1  利用累加主循环次数的计数延时方式来发送一串数据
```

注释二:

这里的计数延时为什么不用累计定时中断次数的延时，而用累计主循环次数的计数延时?
因为本程序定时器中断一次需要500个指令时间，时间分辨率太低，不方便微调时间。因此我
就用累计主循环次数的计数延时方式，在做项目的时候，各位读者应该根据系统的实际情况
来调整const_send_time的大小。

```
        uiSendTimeCnt++;  累计主循环次数的计数延时，为每个字节之间增加延时，
              if(uiSendTimeCntconst_send_time)  请根据实际系统的情况，调整const_send_time的大小
              {
                 uiSendTimeCnt=0;
                     eusart_send(ucSendregBuf[uiSendCnt]);  发送一串数据给上位机
           uiSendCnt++;
                     if(uiSendCnt=9) 说明数据已经发送完毕
                     {
                        uiSendCnt=0;
             ucSendStep=0; 返回到上一步，处理其它未处理的消息
                     }
              }
             break;
  }
}
void eusart_send(unsigned char ucSendData)
{
  ES = 0; 关串口中断
  TI = 0; 清零串口发送完成中断请求标志
  SBUF =ucSendData; 发送一个字节
```

注释三:

根据我个人的经验，在发送一串数据中，每个字节之间必须添加一个延时，用来等待串口发送完成。
当然，也有一些朋友可能不增加延时，直接靠单片机自带的发送完成标志位来判断，但是我以前
在做项目中，感觉单单靠发送完成标志位来判断还是容易出错（当然也有可能是我自身程序的问题），
所以后来在大部分的项目中我就干脆靠延时来等待它发送完成。我在51，PIC单片机中都是这么做的。
但是，凭我的经验，在stm32单片机中，可以不增加延时，直接靠单片机自带的标志位来判断就很可靠。

```
  delay_short(400);  因为外部在每个发送字节之间用了累计主循环次数的计数延时，因此不要此行的delay延时
  TI = 0; 清零串口发送完成中断请求标志
  ES = 1; 允许串口中断
}
void key_scan(void)按键扫描函数 放在定时中断里
{
```

```
if(key_sr1==1)IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock1=0; 按键自锁标志清零
   uiKeyTimeCnt1=0;按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
else if(ucKeyLock1==0)有按键按下，且是第一次被按下
{
   uiKeyTimeCnt1++; 累加定时中断次数
   if(uiKeyTimeCnt1const_key_time1)
   {
      uiKeyTimeCnt1=0;
      ucKeyLock1=1;  自锁按键置位,避免一直触发
      ucKeySec=1;    触发1号键
   }
}
if(key_sr2==1)IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock2=0; 按键自锁标志清零
   uiKeyTimeCnt2=0;按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
else if(ucKeyLock2==0)有按键按下，且是第一次被按下
{
   uiKeyTimeCnt2++; 累加定时中断次数
   if(uiKeyTimeCnt2const_key_time2)
   {
      uiKeyTimeCnt2=0;
      ucKeyLock2=1;  自锁按键置位,避免一直触发
      ucKeySec=2;    触发2号键
   }
}
if(key_sr3==1)IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock3=0; 按键自锁标志清零
   uiKeyTimeCnt3=0;按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
else if(ucKeyLock3==0)有按键按下，且是第一次被按下
{
   uiKeyTimeCnt3++; 累加定时中断次数
   if(uiKeyTimeCnt3const_key_time3)
   {
      uiKeyTimeCnt3=0;
      ucKeyLock3=1;  自锁按键置位,避免一直触发
      ucKeySec=3;    触发3号键
   }
}
if(key_sr4==1)IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock4=0; 按键自锁标志清零
   uiKeyTimeCnt4=0;按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
```

```
  else if(ucKeyLock4==0)有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; 累加定时中断次数
     if(uiKeyTimeCnt4const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  自锁按键置位,避免一直触发
        ucKeySec=4;    触发4号键
     }
  }
}
void key_service(void) 第三区 按键服务的应用程序
{
  switch(ucKeySec) 按键服务状态切换
  {
    case 1 1号键 对应朱兆祺学习板的S1键
          insert_message(0x01);  把新消息插入到环形消息队列里等待处理
          ucVoiceLock=1;  原子锁加锁，保护中断与主函数的共享数据
          uiVoiceCnt=const_voice_short; 按键声音触发，滴一声就停。
                  ucVoiceLock=0; 原子锁解锁
          ucKeySec=0;  响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2 2号键 对应朱兆祺学习板的S5键
          insert_message(0x02);  把新消息插入到环形消息队列里等待处理
          ucVoiceLock=1;  原子锁加锁，保护中断与主函数的共享数据
          uiVoiceCnt=const_voice_short; 按键声音触发，滴一声就停。
                  ucVoiceLock=0; 原子锁解锁
          ucKeySec=0;  响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3 3号键 对应朱兆祺学习板的S9键
          insert_message(0x03);  把新消息插入到环形消息队列里等待处理
          ucVoiceLock=1;  原子锁加锁，保护中断与主函数的共享数据
          uiVoiceCnt=const_voice_short; 按键声音触发，滴一声就停。
                  ucVoiceLock=0; 原子锁解锁
          ucKeySec=0;  响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 4 4号键 对应朱兆祺学习板的S13键
          insert_message(0x04);  把新消息插入到环形消息队列里等待处理
          ucVoiceLock=1;  原子锁加锁，保护中断与主函数的共享数据
          uiVoiceCnt=const_voice_short; 按键声音触发，滴一声就停。
                  ucVoiceLock=0; 原子锁解锁
          ucKeySec=0;  响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void T0_time(void) interrupt 1    定时中断
{
  TF0=0;  清除中断标志
  TR0=0; 关中断
 注释四:
```

此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了红金龙吸味关于原子锁的建议.

```
  if(ucVoiceLock==0) 原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; 每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; 此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  key_scan();按键扫描函数
  TH0=0xfe;   重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  开中断
}
void usart_receive(void) interrupt 4                串口中断
{
   if(RI==1)
   {
        RI = 0;   接收中断，及时把接收中断标志位清零


   }
   else
   {
        TI = 0;    发送中断，及时把发送中断标志位清零
   }

}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;iuiDelayShort;i++)
   {
     ;   一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;iuiDelayLong;i++)
   {
```

```
      for(j=0;j500;j++)   内嵌循环的空指令数量
          {
             ; 一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  第一区 初始化单片机
{
 注释五:
 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
  key_gnd_dr=0; 模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0; 关Led灯
  beep_dr=1; 用PNP三极管控制蜂鸣器，输出高电平时不叫。
  配置定时器
  TMOD=0x01;  设置定时器0为工作方式1
  TH0=0xfe;   重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L12329600);  串口波特率9600。
  TR1=1;
}
void initial_peripheral(void) 第二区 初始化外围
{
   EA=1;      开总中断
   ES=1;      允许串口中断
   ET0=1;     允许定时中断
   TR0=1;     启动定时中断
}
```

复制代码

总结陈词:

前面几个章节中，每个章节要么独立地讲解串口收数据，要么独立地讲解发数据，实际上在大部分的项目中，串口都需要“一收一应答”的握手协议，上位机作为主机，单片机作为从机，主机先发一串数据，从机收到数据后进行校验判断，如果校验正确则返回正确应答指令，如果校验错误则返回错误应答指令，主机收到应答指令后，如果发现是正确应答指令则继续发送其它的新数据，如果发现是错误应答指令，或者超时没有接收到任何应答指令，则继续重发，如果连续重发三次都是错误应答或者无应答，主机就进行报错处理。读者只要把我的串口收发程序结合起来，就很容易实现这样的功能，我就不再详细讲解了。从下一节开始我讲解单片机掉电后数据保存的内容，欲知详情，请听下回分解-----利用AT24C02进行掉电后的数据保存。

（未完待续，下节更精彩，不要走开哦）

第四十四节: 从机的串口收发综合程序框架

开场白:

根据上一节的预告，本来这一节内容打算讲“利用AT24C02进行掉电后的数据保存”的，但是由于网友“261854681”强烈建议我讲一个完整的串口收发程序实例，因此我决定再花两节篇幅讲讲这方面的内容。

实际上在大部分的项目中，串口都需要“一收一应答”的握手协议，上位机作为主机，单片机作为从机，主机先发一串数据，从机收到数据后进行校验判断，如果校验正确则返回正确应答指令，如果校验错误则返回错误应答指令，主机收到应答指令后，如果发现是正确应答指令则继续发送其它的新数据，如果发现是错误应答指令，或者超时没有接收到任何应答指令，则继续重发，如果连续重发三次都是错误应答或者无应答，主机就进行报错处理。

这节先讲从机的收发端程序实例。要教会大家三个知识点:
第一个: 为了保证串口中断接收的数据不丢失，在初始化时必须设置IP = 0x10，相当于把串口中断设置为最高优先级，这个时候，串口中断可以打断任何其他的中断服务函数，实现中断嵌套。
第二个: 从机端的收发端程序框架。
第三个: 从机的状态指示程序框架。可以指示待机，通讯中，超时出错三种状态。
具体内容，请看源代码讲解。
（1）硬件平台:
基于朱兆祺51单片机学习板。
（2）实现功能:
显示和独立按键部分根据第29节的程序来改编，用朱兆祺51单片机学习板中的S1,S5,S9,S13作为独立按键。
一共有4个窗口。每个窗口显示一个参数。有两种更改参数的方式:
第一种: 按键更改参数:
第8,7,6,5位数码管显示当前窗口，P-1代表第1个窗口，P-2代表第2个窗口，P-3代表第3个窗口，P-4代表第1个窗口。
第4,3,2,1位数码管显示当前窗口被设置的参数。范围是从0到9999。S1是加按键，按下此按键会依次增加当前窗口的参数。S5是减按键，按下此按键会依次减少当前窗口的参数。S9是切换窗口按键，按下此按键会依次循环切换不同的窗口。S13是复位按键，当通讯超时蜂鸣器报警时，可以按下此键清除报警。
第二种: 通过串口来更改参数:
波特率是: 9600.
通讯协议: EB 00 55 GG 00 02 XX XX CY
其中第1,2,3位EB 00 55就是数据头
其中第4位GG就是数据类型。01代表更改参数1，02代表更改参数2，03代表更改参数3，04代表更改参数4，
其中第5,6位00 02就是有效数据长度。高位在左，低位在右。
其中从第7，8位XX XX是被更改的参数。高位在左，低位在右。
第9位CY是累加和，前面所有字节的累加。
一个完整的通讯必须接收完4串数据，每串数据之间的间隔时间不能超过10秒钟，否则认为通讯超时出错引发蜂鸣器报警。如果接收到得数据校验正确，
则返回校验正确应答: eb 00 55 f5 00 00 35，
否则返回校验出错应答:: eb 00 55 fa 00 00 3a。
系统处于待机状态时，LED灯一直亮，
系统处于非待机状态时，LED灯闪烁，
系统处于通讯超时出错状态时，LED灯闪烁，并且蜂鸣器间歇鸣叫报警。
通过电脑的串口助手，依次发送以下测试数据，将会分别更改参数1，参数2，参数3，参数4。注意，每串数据之间的时间最大不能超过10秒，否则系统认为通讯超时报警。
把参数1更改为十进制的1: eb 00 55 01 00 02 00 01 44
把参数2更改为十进制的12: eb 00 55 02 00 02 00 0c 50
把参数3更改为十进制的123: eb 00 55 03 00 02 00 7b c0
把参数4更改为十进制的1234: eb 00 55 04 00 02 04 d2 1c
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
#define const_led_0_5s  200   //大概0.5秒的时间
#define const_led_1s    400   //大概1秒的时间
#define const_send_time_out   4000  //通讯超时出错的时间 大概10秒
#define const_rc_size  20  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时
```

```
间根据实际情况来调整大小
#define const_send_size  10  //串口发送数据的缓冲区数组大小
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData); //发送一个字节，内部自带每个字节之间的delay延时
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void status_service(void);  //状态显示的应用程序
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为状态指示灯 亮的时候表示待机状态.闪烁表示非待机状态，处于正在发送数据或者出错的
状态
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucSendregBuf[const_send_size]; //发送的缓冲区数组
unsigned int  uiSendCnt=0;      //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;     //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char  ucSendCntLock=0; //串口计时器的原子锁
unsigned char ucRcType=0;  //数据类型
unsigned int  uiRcSize=0;  //数据长度
unsigned char ucRcCy=0;  //校验累加和
unsigned int  uiLedCnt=0;  //控制Led闪烁的延时计时器
unsigned int  uiSendTimeOutCnt=0; //用来识别接收数据超时的计时器
unsigned char ucSendTimeOutLock=0; //原子锁
unsigned char ucStatus=0; //当前状态变量 0代表待机 1代表正在通讯过程 2代表发送出错
unsigned char ucKeySec=0;    //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
```

```
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd4Update=0; //窗口4更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
```

```
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
          usart_service();  //串口服务程序
      display_service(); //显示的窗口菜单服务程序
          status_service();  //状态显示的应用程序
   }
}
void status_service(void)  //状态显示的应用程序
{
   if(ucStatus!=0) //处于非待机的状态,Led闪烁
   {
      if(uiLedCnt<const_led_0_5s)  //大概0.5秒
          {
             led_dr=1;  //前半秒亮
                 if(ucStatus==2)  //处于发送数据出错的状态，则蜂鸣器间歇鸣叫报警
                 {
             ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
             uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
             ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
                 }
          }
          else if(uiLedCnt<const_led_1s)  //大概1秒
          {
             led_dr=0; //前半秒灭
          }
          else
          {
             uiLedCnt=0; //延时计时器清零，让Led灯处于闪烁的反复循环中
          }

   }
   else  //处于待机状态，Led一直亮
   {
      led_dr=1;

   }
}
void usart_service(void)  //串口服务程序,在main函数里
```

```
{
    unsigned int i;

    if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
    {
        ucSendLock=0;    //处理一次就锁起来，不用每次都进来,除非有新接收的数据
        //下面的代码进入数据协议解析和数据处理的阶段
        uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
        while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
        {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
            {
                ucRcType=ucRcregBuf[uiRcMoveIndex+3];   //数据类型  一个字节
                uiRcSize=ucRcregBuf[uiRcMoveIndex+4];   //数据长度  两个字节
                uiRcSize=uiRcSize<<8;
                uiRcSize=uiRcSize+ucRcregBuf[uiRcMoveIndex+5];

                ucRcCy=ucRcregBuf[uiRcMoveIndex+6+uiRcSize];   //记录最后一个字节的校验
                ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=0;  //清零最后一个字节的累加和变量
                for(i=0;i<(3+1+2+uiRcSize);i++) //计算校验累加和
                {
ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=ucRcregBuf[uiRcMoveIndex+6+uiRcSize]+ucRcregBuf[i];
                }
                if(ucRcCy==ucRcregBuf[uiRcMoveIndex+6+uiRcSize])  //如果校验正确，则进入以下数据处理
                {
                    switch(ucRcType)   //根据不同的数据类型来做不同的数据处理
                    {
                        case 0x01:   //设置参数1
                                                    ucStatus=1; //从设置参数1开始，表示当前处
于正在发送数据的状态
                            uiSetData1=ucRcregBuf[uiRcMoveIndex+6];  //把两个字节合并成一个int类型
的数据
                            uiSetData1=uiSetData1<<8;
                            uiSetData1=uiSetData1+ucRcregBuf[uiRcMoveIndex+7];
                            ucWd1Update=1; //窗口1更新显示
                            break;

                        case 0x02:   //设置参数2
                            uiSetData2=ucRcregBuf[uiRcMoveIndex+6];  //把两个字节合并成一个int类型
的数据
                            uiSetData2=uiSetData2<<8;
                            uiSetData2=uiSetData2+ucRcregBuf[uiRcMoveIndex+7];
                            ucWd2Update=1; //窗口2更新显示
                            break;
                        case 0x03:   //设置参数3
                            uiSetData3=ucRcregBuf[uiRcMoveIndex+6];  //把两个字节合并成一个int类型
的数据
                            uiSetData3=uiSetData3<<8;
                            uiSetData3=uiSetData3+ucRcregBuf[uiRcMoveIndex+7];
```

```
                        ucWd3Update=1; //窗口3更新显示
                        break;
                    case 0x04:    //设置参数4
                                                        ucStatus=0; //从设置参数4结束发送数据的状
态，表示发送数据的过程成功，切换回待机状态
                        uiSetData4=ucRcregBuf[uiRcMoveIndex+6];  //把两个字节合并成一个int类型
的数据
                        uiSetData4=uiSetData4<<8;
                        uiSetData4=uiSetData4+ucRcregBuf[uiRcMoveIndex+7];
                        ucWd4Update=1; //窗口4更新显示
                        break;

                }
                                    ucSendregBuf[0]=0xeb;     //把准备发送的数据放入发送缓冲
区
                ucSendregBuf[1]=0x00;
                ucSendregBuf[2]=0x55;
                ucSendregBuf[3]=0xf5;  //代表校验正确
                ucSendregBuf[4]=0x00;
                ucSendregBuf[5]=0x00;
                ucSendregBuf[6]=0x35;
                for(i=0;i<7;i++)  //返回校验正确的应答指令
                {
                   eusart_send(ucSendregBuf[i]);  //发送一串数据给上位机
                }
            }
                        else
                           {
                ucSendTimeOutLock=1; //原子锁加锁
                                uiSendTimeOutCnt=0;  //超时计时器计时清零
                ucSendTimeOutLock=0; //原子锁解锁
                                    ucSendregBuf[0]=0xeb;     //把准备发送的数据放入发送缓冲
区
                ucSendregBuf[1]=0x00;
                ucSendregBuf[2]=0x55;
                ucSendregBuf[3]=0xfa;   //代表校验错误
                ucSendregBuf[4]=0x00;
                ucSendregBuf[5]=0x00;
                ucSendregBuf[6]=0x3a;
                for(i=0;i<7;i++)  //返回校验错误的应答指令
                {
                   eusart_send(ucSendregBuf[i]);  //发送一串数据给上位机
                }

                            }
                            ucSendTimeOutLock=1; //原子锁加锁
                            uiSendTimeOutCnt=0;  //超时计时器计时清零
        ucSendTimeOutLock=0; //原子锁解锁
        break;   //退出循环
    }
```

```
            uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
        }

        uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

    }

}
void eusart_send(unsigned char ucSendData) //发送一个字节，内部自带每个字节之间的delay延时
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
      case 1:   //显示P--1窗口的数据
           if(ucWd1Update==1)  //窗口1要全部更新显示
   {
              ucWd1Update=0;  //及时清零标志，避免一直进来扫描
              ucDigShow8=12;  //第8位数码管显示P
              ucDigShow7=11;  //第7位数码管显示-
              ucDigShow6=1;   //第6位数码管显示1
              ucDigShow5=10;  //第5位数码管显示无
             //先分解数据
                      ucTemp4=uiSetData1/1000;
                      ucTemp3=uiSetData1%1000/100;
                      ucTemp2=uiSetData1%100/10;
                      ucTemp1=uiSetData1%10;

                         //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
              if(uiSetData1<1000)
                          {
                             ucDigShow4=10;  //如果小于1000，千位显示无
                          }
              else
                          {
                 ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                          }
              if(uiSetData1<100)
                          {
                 ucDigShow3=10;  //如果小于100，百位显示无
                          }
                          else
```

```
                    {
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                    }
            if(uiSetData1<10)
                    {
            ucDigShow2=10;  //如果小于10，十位显示无
                    }
                    else
                    {
            ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
        }
        break;
    case 2:  //显示P--2窗口的数据
        if(ucWd2Update==1)  //窗口2要全部更新显示
{
            ucWd2Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=2;  //第6位数码管显示2
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData2/1000;     //分解数据
                    ucTemp3=uiSetData2%1000/100;
                    ucTemp2=uiSetData2%100/10;
                    ucTemp1=uiSetData2%10;
            if(uiSetData2<1000)
                    {
                       ucDigShow4=10;  //如果小于1000，千位显示无
                    }
            else
                    {
            ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                    }
            if(uiSetData2<100)
                    {
            ucDigShow3=10;  //如果小于100，百位显示无
                    }
                    else
                    {
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                    }
            if(uiSetData2<10)
                    {
            ucDigShow2=10;  //如果小于10，十位显示无
                    }
                    else
                    {
            ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
```

```
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
 }
          break;
     case 3:  //显示P--3窗口的数据
         if(ucWd3Update==1)  //窗口3要全部更新显示
{
            ucWd3Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=3;  //第6位数码管显示3
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData3/1000;      //分解数据
                    ucTemp3=uiSetData3%1000/100;
                    ucTemp2=uiSetData3%100/10;
                    ucTemp1=uiSetData3%10;
            if(uiSetData3<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData3<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData3<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
         break;
     case 4:  //显示P--4窗口的数据
         if(ucWd4Update==1)  //窗口4要全部更新显示
{
            ucWd4Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=4;  //第6位数码管显示4
            ucDigShow5=10;   //第5位数码管显示无
```

```
                    ucTemp4=uiSetData4/1000;      //分解数据
                    ucTemp3=uiSetData4%1000/100;
                    ucTemp2=uiSetData4%100/10;
                    ucTemp1=uiSetData4%10;
                if(uiSetData4<1000)
                            {
                               ucDigShow4=10;  //如果小于1000，千位显示无
                            }
                else
                            {
                   ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                            }
                if(uiSetData4<100)
                            {
                   ucDigShow3=10;  //如果小于100，百位显示无
                            }
                            else
                            {
                   ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                            }
                if(uiSetData4<10)
                            {
                   ucDigShow2=10;  //如果小于10，十位显示无
                            }
                            else
                            {
                   ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                }
                ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
    }
              break;
           }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
```

```
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;    //触发4号键
     }
  }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
```

```
{
  case 1:// 加按键 对应朱兆祺学习板的S1键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
                {
                   case 1:
                uiSetData1++;
                                if(uiSetData1>9999) //最大值是9999
                                {
                                   uiSetData1=9999;
                                }
                           ucWd1Update=1;  //窗口1更新显示
                              break;
                   case 2:
                uiSetData2++;
                                if(uiSetData2>9999) //最大值是9999
                                {
                                   uiSetData2=9999;
                                }
                           ucWd2Update=1;  //窗口2更新显示
                              break;
                   case 3:
                uiSetData3++;
                                if(uiSetData3>9999) //最大值是9999
                                {
                                   uiSetData3=9999;
                                }
                           ucWd3Update=1;  //窗口3更新显示
                              break;
                   case 4:
                uiSetData4++;
                                if(uiSetData4>9999) //最大值是9999
                                {
                                   uiSetData4=9999;
                                }
                           ucWd4Update=1;  //窗口4更新显示
                              break;
                }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

  case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
                {
                   case 1:
                uiSetData1--;
                                if(uiSetData1>9999)
                                {
```

```
                        uiSetData1=0;  //最小值是0
                    }
                ucWd1Update=1;  //窗口1更新显示
                    break;
            case 2:
            uiSetData2--;
                    if(uiSetData2>9999)
                    {
                        uiSetData2=0;  //最小值是0
                    }
                ucWd2Update=1;  //窗口2更新显示
                    break;
            case 3:
            uiSetData3--;
                    if(uiSetData3>9999)
                    {
                        uiSetData3=0;  //最小值是0
                    }
                ucWd3Update=1;  //窗口3更新显示
                    break;
            case 4:
            uiSetData4--;
                    if(uiSetData4>9999)
                    {
                        uiSetData4=0;  //最小值是0
                    }
                ucWd4Update=1;  //窗口4更新显示
                    break;
            }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 切换窗口按键 对应朱兆祺学习板的S9键
        ucWd++;  //切换窗口
                if(ucWd>4)
                {
                  ucWd=1;
                }
        switch(ucWd)  //在不同的窗口下，在不同的窗口下，更新显示不同的窗口
                {
                    case 1:
                        ucWd1Update=1;  //窗口1更新显示
                            break;
                    case 2:
                        ucWd2Update=1;  //窗口2更新显示
                            break;
                    case 3:
                        ucWd3Update=1;  //窗口3更新显示
```

```
                        break;
                    case 4:
                        ucWd4Update=1;  //窗口4更新显示
                        break;
                }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 4:// 复位按键 对应朱兆祺学习板的S13键
            switch(ucStatus)  //在不同的状态下，进行不同的操作
            {
               case 0:  //处于待机状态
                    break;
               case 1:  //处于正在通讯的过程
                    break;
               case 2: //发送数据出错，比如中间超时没有接收到数据
                    ucStatus=0; //切换回待机的状态
                    break;
            }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;

  }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                if(ucDigDot1==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                if(ucDigDot2==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
```

```
    case 3:  //显示第3位
         ucDigShowTemp=dig_table[ucDigShow3];
                 if(ucDigDot3==1)
                 {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                 }
         dig_hc595_drive(ucDigShowTemp,0xfb);
             break;
    case 4:  //显示第4位
         ucDigShowTemp=dig_table[ucDigShow4];
                 if(ucDigDot4==1)
                 {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                 }
         dig_hc595_drive(ucDigShowTemp,0xf7);
             break;
    case 5:  //显示第5位
         ucDigShowTemp=dig_table[ucDigShow5];
                 if(ucDigDot5==1)
                 {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                 }
         dig_hc595_drive(ucDigShowTemp,0xef);
             break;
    case 6:  //显示第6位
         ucDigShowTemp=dig_table[ucDigShow6];
                 if(ucDigDot6==1)
                 {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                 }
         dig_hc595_drive(ucDigShowTemp,0xdf);
             break;
    case 7:  //显示第7位
         ucDigShowTemp=dig_table[ucDigShow7];
                 if(ucDigDot7==1)
                 {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
         }
         dig_hc595_drive(ucDigShowTemp,0xbf);
             break;
    case 8:  //显示第8位
         ucDigShowTemp=dig_table[ucDigShow8];
                 if(ucDigDot8==1)
                 {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                 }
         dig_hc595_drive(ucDigShowTemp,0x7f);
             break;
}
ucDisplayDriveStep++;
```

```
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
```

```
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucLedStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   hc595_st_dr=1;
   delay_short(1);
   hc595_sh_dr=0;    //拉低，抗干扰就增强
   hc595_st_dr=0;
   hc595_ds_dr=0;
}
void usart_receive(void) interrupt 4   //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
         ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        if(ucSendCntLock==0)  //原子锁判断
        {
            ucSendCntLock=1; //加锁
            uiSendCnt=0;  //及时喂狗，虽然在定时中断那边此变量会不断累加，但是只要串口的数据还没发送完毕
，那么它永远也长不大,因为每个串口接收中断它都被清零。
            ucSendCntLock=0; //解锁
        }

   }
   else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
```

```
   {
       TI = 0;  //如果不是串口接收中断，那么必然是串口发送中断，及时清除发送中断的标志，否则一直发送中
断
   }

}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释一:
  * 此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建议.
  */
  if(ucSendCntLock==0)  //原子锁判断
  {
     ucSendCntLock=1; //加锁
     if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
     {
        uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个
中间没有串口数据过来
        ucSendLock=1;     //开自锁标志
     }
     ucSendCntLock=0; //解锁
  }
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  if(ucStatus!=0) //处于非待机的状态,Led闪烁
  {
     uiLedCnt++; //Led闪烁计时器不断累加
  }
  if(ucStatus==1) //处于正在通讯的状态,
  {
     if(ucSendTimeOutLock==0)  //原子锁判断
         {
        uiSendTimeOutCnt++;   //超时计时器累加
            if(uiSendTimeOutCnt>const_send_time_out)  //超时出错
            {
```

```
                uiSendTimeOutCnt=0;
                ucStatus=2;  //切换到出错报警状态
              }
          }
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=1;  //点亮独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
/* 注释三:
* 为了保证串口中断接收的数据不丢失，必须设置IP = 0x10，相当于把串口中断设置为最高优先级，
```

```
* 这个时候，串口中断可以打断任何其他的中断服务函数实现嵌套，
*/
  IP =0x10;  //把串口中断设置为最高优先级，必须的。
  TH1=TL1=-(11059200L/12/32/9600);  //串口波特率为9600。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=0;    //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节详细讲了从机收发端的程序框架，而主机端的程序则用电脑的串口助手来模拟。实际上，主机端的程序也有很多内容，它包括依次发送每一串数据，根据返回的应答来决定是否需要重发数据，重发三次如果没反应则进行报错，以及超时没接收到数据等等内容。主机收发端的程序框架是什么样的？欲知详情，请听下回分解-----主机的串口收发综合程序框架

（未完待续，下节更精彩，不要走开哦）

第四十五节：主机的串口收发综合程序框架

开场白:

在大部分的项目中，串口都需要“一收一应答”的握手协议，主机先发一串数据，从机收到数据后进行校验判断，如果校验正确则返回正确应答指令，如果校验错误则返回错误应答指令，主机收到应答指令后，如果发现是正确应答指令则继续发送其它的新数据，如果发现是错误应答指令，或者超时没有接收到任何应答指令，则继续重发，如果连续重发三次都是错误应答或者无应答，主机就进行报错处理。

上一节已经讲了从机，这节就讲主机的收发端程序实例。要教会大家四个知识点:

第一个: 为了保证串口中断接收的数据不丢失，在初始化时必须设置IP= 0x10，相当于把串口中断设置为最高优先级，这个时候，串口中断可以打断任何其他的中断服务函数，实现中断嵌套。

第二个: 主机端的收发端程序框架。包括重发，超时检测等等。

第三个: 主机的状态指示程序框架。可以指示待机，通讯中，超时出错三种状态。

第四个: 其实上一节的LED灯闪烁的时间里，我忘了加原子锁，不加原子锁的后果是，闪烁的时间有时候会不一致，所以这节多增加一个原子锁变量ucLedLock，再次感谢“红金龙吸味”关于原子锁的建议，真的很好用。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

显示和独立按键部分根据第29节的程序来改编，用朱兆祺51单片机学习板中的S1,S5,S9,S13作为独立按键。

一共有4个窗口。每个窗口显示一个参数。串口可以把当前设置的4个数据发送给从机。从机端可以用电脑的串口助手来模拟。

第一: 按键更改参数:

第8,7,6,5位数码管显示当前窗口，P-1代表第1个窗口，P-2代表第2个窗口，P-3代表第3个窗口，P-4代表第1个窗口。

第4,3,2,1位数码管显示当前窗口被设置的参数。范围是从0到9999。S1是加按键，按下此按键会依次增加当前窗口的参数。S5是减按键，按下此按键会依次减少当前窗口的参数。S9是切换窗口按键，按下此按键会依次循环切换不同的窗口。S13是启动发送数据和复位按键，当系统处于待机状态时，按下此按键会启动发送数据；当通讯超时蜂鸣器报警时，可以按下此键清除报警，返回到待机的状态。

第二：通过串口把更改的参数发送给从机。

波特率是：9600.

通讯协议：EB00 55 GG 00 02 XX XX CY

其中第1,2,3位EB00 55就是数据头

其中第4位GG就是数据类型。01代表更改参数1，02代表更改参数2，03代表更改参数3，04代表更改参数4，

其中第5,6位0002就是有效数据长度。高位在左，低位在右。

其中从第7，8位XXXX是被更改的参数。高位在左，低位在右。

第9位CY是累加和，前面所有字节的累加。

一个完整的通讯必须发送完4串数据，每串数据之间的间隔时间不能超过10秒钟，否则认为通讯超时主机会重发数据，如果连续三次都没有返回，则引发蜂鸣器报警。如果接收到得数据校验正确，主机继续发送新的一串数据，直到把4串数据发送完毕为止。

系统处于待机状态时，LED灯一直亮，

系统处于非待机状态时，LED灯闪烁，

系统处于出错状态时，LED灯闪烁，并且蜂鸣器间歇鸣叫报警。

通过电脑的串口助手来模拟从机，返回不同的应答

从机返回校验正确应答：eb 00 55 f5 00 00 35

从机返回校验出错应答：eb00 55 fa 00 00 3a

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
#define const_led_0_5s  200   //大概0.5秒的时间
#define const_led_1s    400   //大概1秒的时间
#define const_send_time_out   4000  //通讯超时出错的时间 大概10秒
#define const_rc_size  20  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define const_send_size  10  //串口发送数据的缓冲区数组大小
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口接收服务程序,在main函数里
```

```
void communication_service(void); //一发一收的通讯服务程序
void eusart_send(unsigned char ucSendData); //发送一个字节，内部自带每个字节之间的delay延时
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void status_service(void);  //状态显示的应用程序
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为状态指示灯 亮的时候表示待机状态.闪烁表示非待机状态，处于正在发送数据或者出错的状态
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucSendregBuf[const_send_size]; //发送的缓冲区数组
unsigned int  uiSendCnt=0;      //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;     //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char  ucSendCntLock=0; //串口计时器的原子锁
unsigned char ucRcType=0;  //数据类型
unsigned int  uiRcSize=0;  //数据长度
unsigned char ucRcCy=0;  //校验累加和
unsigned char ucLedLock=0; //原子锁
unsigned int  uiLedCnt=0;  //控制Led闪烁的延时计时器
unsigned int  uiSendTimeOutCnt=0; //用来识别接收数据超时的计时器
unsigned char ucSendTimeOutLock=0; //原子锁
unsigned char ucStatus=0; //当前状态变量 0代表待机 1代表正在通讯过程 2代表发送出错
unsigned char ucSendStep=0; //发送数据的过程步骤
unsigned char ucErrorCnt=0; //累计错误总数
unsigned char ucSendTotal=0; //记录当前已经发送了多少串数据
unsigned char ucReceiveStatus=0; //返回的数据状态 0代表待机 1代表校验正确 2代表校验出错
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
```

```
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd4Update=0; //窗口4更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
```

```
    delay_long(100);
    initial_peripheral();
    while(1)
    {
       key_service(); //按键服务的应用程序
       usart_service();  //串口接收服务程序
       communication_service(); //一发一收的通讯服务程序
       display_service(); //显示的窗口菜单服务程序
       status_service();  //状态显示的应用程序
    }
}
void communication_service(void) //一发一收的通讯服务程序
{
    unsigned int i;
    if(ucStatus==1)  //处于正在通讯的过程中
    {
        switch(ucSendStep)
            {
                case 0: //通讯过程0  发送一串数据
                 switch(ucSendTotal)  //根据当前已经发送到第几条数据来决定发送哪些参数
                                {
                                   case 0:   //发送参数1
                            ucSendregBuf[0]=0xeb;     //把准备发送的数据放入发送缓冲区
                            ucSendregBuf[1]=0x00;
                            ucSendregBuf[2]=0x55;
                            ucSendregBuf[3]=0x01;     //代表发送参数1
                            ucSendregBuf[4]=0x00;
                            ucSendregBuf[5]=0x02;     //代表发送2个字节的有效数据
                                                ucSendregBuf[6]=uiSetData1>>8;  //把int类型的参数分解成
两个字节的数据
                                                ucSendregBuf[7]=uiSetData1;
                                        break;
                                   case 1:  //发送参数2
                            ucSendregBuf[0]=0xeb;     //把准备发送的数据放入发送缓冲区
                            ucSendregBuf[1]=0x00;
                            ucSendregBuf[2]=0x55;
                            ucSendregBuf[3]=0x02;     //代表发送参数2
                            ucSendregBuf[4]=0x00;
                            ucSendregBuf[5]=0x02;     //代表发送2个字节的有效数据
                                                ucSendregBuf[6]=uiSetData2>>8;  //把int类型的参数分解成
两个字节的数据
                                                ucSendregBuf[7]=uiSetData2;
                                        break;
                                   case 2:  //发送参数3
                            ucSendregBuf[0]=0xeb;     //把准备发送的数据放入发送缓冲区
                            ucSendregBuf[1]=0x00;
                            ucSendregBuf[2]=0x55;
                            ucSendregBuf[3]=0x03;     //代表发送参数3
                            ucSendregBuf[4]=0x00;
                            ucSendregBuf[5]=0x02;     //代表发送2个字节的有效数据
```

```
                                        ucSendregBuf[6]=uiSetData3>>8;  //把int类型的参数分解成
两个字节的数据
                                        ucSendregBuf[7]=uiSetData3;
                                    break;
                                case 3:  //发送参数4
                            ucSendregBuf[0]=0xeb;    //把准备发送的数据放入发送缓冲区
                            ucSendregBuf[1]=0x00;
                            ucSendregBuf[2]=0x55;
                            ucSendregBuf[3]=0x04;    //代表发送参数4
                            ucSendregBuf[4]=0x00;
                            ucSendregBuf[5]=0x02;    //代表发送2个字节的有效数据
                                        ucSendregBuf[6]=uiSetData4>>8;  //把int类型的参数分解成
两个字节的数据
                                        ucSendregBuf[7]=uiSetData4;
                                    break;
                                }

                        ucSendregBuf[8]=0x00;
                        for(i=0;i<8;i++)  //最后一个字节是校验和，是前面所有字节累加，溢出部分不用我们管，系统会
有规律的自动处理
                        {
                          ucSendregBuf[8]=ucSendregBuf[8]+ucSendregBuf[i];
                        }
                        for(i=0;i<9;i++)
                        {
                            eusart_send(ucSendregBuf[i]);  //把一串完整的数据发送给下位机
                        }
                        ucSendTimeOutLock=1; //原子锁加锁
                        uiSendTimeOutCnt=0;  //超时计时器计时清零
                        ucSendTimeOutLock=0; //原子锁解锁
                                ucReceiveStatus=0;  //返回的数据状态清零
                                ucSendStep=1;  //切换到下一个步骤，等待返回的数据
                        break;
                    case 1: //通讯过程1  判断返回的指令
                            if(ucReceiveStatus==1)  //校验正确
                                {
                                    ucErrorCnt=0; //累计校验错误总数清零
                                   ucSendTotal++;  //累加当前发送了多少串数据
                                   if(ucSendTotal>=4) //已经发送完全部4串数据，结束
                                   {
                              ucStatus=0;  //切换到结束时的待机状态
                                   }
                                   else  //还没发送完4串数据，则继续发送下一串新数据
                                   {
                                            ucSendStep=0;  //返回上一个步骤，继续发送新数据
                                   }
                                }
                            else if(ucReceiveStatus==2||uiSendTimeOutCnt>const_send_time_out)  //校验出错或
者超时出错
                                {
```

```
                                ucErrorCnt++; //累计错误总数
                    if(ucErrorCnt>=3)  //累加重发次数3次以上，则报错
                                    {
                       ucStatus=2;  //切换到出错报警状态
                                    }
                                    else  //重发还没超过3次，继续返回重发
                                    {
                                              ucSendStep=0;  //返回上一个步骤，重发一次数据
                                    }
                                }
                          break;

           }

   }
}
void status_service(void)  //状态显示的应用程序
{
   if(ucStatus!=0) //处于非待机的状态,Led闪烁
   {
      if(uiLedCnt<const_led_0_5s)  //大概0.5秒
          {
             led_dr=1;  //前半秒亮
                 if(ucStatus==2)  //处于发送数据出错的状态，则蜂鸣器间歇鸣叫报警
                 {
             ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
             uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
             ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
                 }
          }
          else if(uiLedCnt<const_led_1s)  //大概1秒
          {
             led_dr=0; //前半秒灭
          }
          else
          {
                    ucLedLock=1; //原子锁加锁
             uiLedCnt=0; //延时计时器清零，让Led灯处于闪烁的反复循环中
                        ucLedLock=0; //原子锁解锁
          }

   }
   else  //处于待机状态，Led一直亮
   {
      led_dr=1;

   }
}
void usart_service(void)  //串口接收服务程序,在main函数里
{
```

```
    unsigned int i;

    if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
    {
        ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
        //下面的代码进入数据协议解析和数据处理的阶段
        uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
        while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
        {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
            {
                ucRcType=ucRcregBuf[uiRcMoveIndex+3];   //数据类型  一个字节
                uiRcSize=ucRcregBuf[uiRcMoveIndex+4];   //数据长度  两个字节
                uiRcSize=uiRcSize<<8;
                uiRcSize=uiRcSize+ucRcregBuf[uiRcMoveIndex+5];

                ucRcCy=ucRcregBuf[uiRcMoveIndex+6+uiRcSize];   //记录最后一个字节的校验
                ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=0;  //清零最后一个字节的累加和变量
                for(i=0;i<(3+1+2+uiRcSize);i++) //计算校验累加和
                {
ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=ucRcregBuf[uiRcMoveIndex+6+uiRcSize]+ucRcregBuf[i];
                }
                 if(ucRcCy==ucRcregBuf[uiRcMoveIndex+6+uiRcSize])  //如果一串数据校验正确，则进入以下
数据指令的判断
                 {
                    switch(ucRcType)   //根据不同的数据类型来做不同的数据处理
                    {
                          case 0xf5:   //返回的是正确的校验指令
                               ucReceiveStatus=1;//代表校验正确
                               break;

                          case 0xfa:   //返回的是错误的校验指令
                               ucReceiveStatus=2;//代表校验错误
                               break;
                     }
                  }
                  break;   //退出循环
            }
            uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
        }

        uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

    }

}
void eusart_send(unsigned char ucSendData) //发送一个字节，内部自带每个字节之间的delay延时
{
  ES = 0; //关串口中断
```

```
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示P--1窗口的数据
            if(ucWd1Update==1)  //窗口1要全部更新显示
   {
               ucWd1Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=12;  //第8位数码管显示P
               ucDigShow7=11;  //第7位数码管显示-
               ucDigShow6=1;   //第6位数码管显示1
               ucDigShow5=10;  //第5位数码管显示无
              //先分解数据
                       ucTemp4=uiSetData1/1000;
                       ucTemp3=uiSetData1%1000/100;
                       ucTemp2=uiSetData1%100/10;
                       ucTemp1=uiSetData1%10;

                          //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
               if(uiSetData1<1000)
                           {
                              ucDigShow4=10;  //如果小于1000，千位显示无
                           }
               else
                           {
                  ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                           }
               if(uiSetData1<100)
                           {
                  ucDigShow3=10;  //如果小于100，百位显示无
                           }
                           else
                           {
                  ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                           }
               if(uiSetData1<10)
                           {
                  ucDigShow2=10;  //如果小于10，十位显示无
                           }
                           else
                           {
                  ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
               }
```

```
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
        }
        break;
    case 2:  //显示P--2窗口的数据
        if(ucWd2Update==1)  //窗口2要全部更新显示
{
            ucWd2Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=2;  //第6位数码管显示2
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData2/1000;     //分解数据
                    ucTemp3=uiSetData2%1000/100;
                    ucTemp2=uiSetData2%100/10;
                    ucTemp1=uiSetData2%10;
            if(uiSetData2<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData2<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData2<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
         break;
    case 3:  //显示P--3窗口的数据
        if(ucWd3Update==1)  //窗口3要全部更新显示
{
            ucWd3Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=3;  //第6位数码管显示3
            ucDigShow5=10;   //第5位数码管显示无
```

```
                ucTemp4=uiSetData3/1000;      //分解数据
                ucTemp3=uiSetData3%1000/100;
                ucTemp2=uiSetData3%100/10;
                ucTemp1=uiSetData3%10;
            if(uiSetData3<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData3<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData3<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
         break;
     case 4:  //显示P--4窗口的数据
         if(ucWd4Update==1)  //窗口4要全部更新显示
{
            ucWd4Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=4;  //第6位数码管显示4
            ucDigShow5=10;   //第5位数码管显示无
                ucTemp4=uiSetData4/1000;      //分解数据
                ucTemp3=uiSetData4%1000/100;
                ucTemp2=uiSetData4%100/10;
                ucTemp1=uiSetData4%10;
            if(uiSetData4<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
```

```
                    }
                if(uiSetData4<100)
                    {
                   ucDigShow3=10;  //如果小于100，百位显示无
                    }
                    else
                    {
                   ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                    }
                if(uiSetData4<10)
                    {
                   ucDigShow2=10;  //如果小于10，十位显示无
                    }
                    else
                    {
                   ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                }
                ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
    }
              break;
           }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
```

```
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
      }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;    //触发4号键
     }
  }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
                  {
                     case 1:
                  uiSetData1++;
                                  if(uiSetData1>9999) //最大值是9999
                                  {
                                     uiSetData1=9999;
                                  }
                            ucWd1Update=1;  //窗口1更新显示
```

```
                    break;
                case 2:
            uiSetData2++;
                        if(uiSetData2>9999) //最大值是9999
                        {
                           uiSetData2=9999;
                        }
                    ucWd2Update=1;  //窗口2更新显示
                    break;
                case 3:
            uiSetData3++;
                        if(uiSetData3>9999) //最大值是9999
                        {
                           uiSetData3=9999;
                        }
                    ucWd3Update=1;  //窗口3更新显示
                    break;
                case 4:
            uiSetData4++;
                        if(uiSetData4>9999) //最大值是9999
                        {
                           uiSetData4=9999;
                        }
                    ucWd4Update=1;  //窗口4更新显示
                    break;
            }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
                {
                case 1:
            uiSetData1--;
                        if(uiSetData1>9999)
                        {
                           uiSetData1=0;  //最小值是0
                        }
                    ucWd1Update=1;  //窗口1更新显示
                    break;
                case 2:
            uiSetData2--;
                        if(uiSetData2>9999)
                        {
                           uiSetData2=0;  //最小值是0
                        }
                    ucWd2Update=1;  //窗口2更新显示
```

```
                        break;
                case 3:
            uiSetData3--;
                            if(uiSetData3>9999)
                            {
                               uiSetData3=0;  //最小值是0
                            }
                        ucWd3Update=1;  //窗口3更新显示
                           break;
                case 4:
            uiSetData4--;
                            if(uiSetData4>9999)
                            {
                               uiSetData4=0;  //最小值是0
                            }
                        ucWd4Update=1;  //窗口4更新显示
                           break;
            }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 切换窗口按键 对应朱兆祺学习板的S9键
        ucWd++;  //切换窗口
            if(ucWd>4)
            {
              ucWd=1;
            }
        switch(ucWd)  //在不同的窗口下，在不同的窗口下，更新显示不同的窗口
            {
                case 1:
                      ucWd1Update=1;  //窗口1更新显示
                         break;
                case 2:
                      ucWd2Update=1;  //窗口2更新显示
                         break;
                case 3:
                      ucWd3Update=1;  //窗口3更新显示
                         break;
                case 4:
                      ucWd4Update=1;  //窗口4更新显示
                         break;
            }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 启动发送数据和复位按键 对应朱兆祺学习板的S13键
```

```
        switch(ucStatus)  //在不同的状态下，进行不同的操作
        {
           case 0:  //处于待机状态，则启动发送数据
                ucErrorCnt=0; //累计错误总数清零
                ucSendTotal=0; //已经发送串数据总数清零
                                ucSendStep=0; //发送数据的过程步骤清零,返回开始的步骤待命
                ucStatus=1; //启动发送数据，1代表正在通讯过程
                break;
           case 1:  //处于正在通讯的过程
                break;
           case 2: //发送数据出错，比如中间超时没有接收到数据
                ucStatus=0; //切换回待机的状态
                break;
        }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

  }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
```

```
            break;
        case 4:  //显示第4位
            ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xef);
            break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
```

```
{
   unsigned char i;
   unsigned char ucTempData;
   dig_hc595_sh_dr=0;
   dig_hc595_st_dr=0;
   ucTempData=ucDigStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucDigStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   dig_hc595_st_dr=1;
   delay_short(1);
   dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
   dig_hc595_st_dr=0;
   dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
```

```
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;     //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void usart_receive(void) interrupt 4   //串口接收数据中断
{
    if(RI==1)
    {
        RI = 0;
         ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        if(ucSendCntLock==0)  //原子锁判断
        {
            ucSendCntLock=1; //加锁
            uiSendCnt=0;  //及时喂狗，虽然在定时中断那边此变量会不断累加，但是只要串口的数据还没发送完毕
，那么它永远也长不大,因为每个串口接收中断它都被清零。
            ucSendCntLock=0; //解锁
        }

    }
    else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
    {
        TI = 0;  //如果不是串口接收中断，那么必然是串口发送中断，及时清除发送中断的标志，否则一直发送中
断
    }

}
void T0_time(void) interrupt 1   //定时中断
```

```
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释一:
  * 此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建议.
  */
  if(ucSendCntLock==0)  //原子锁判断
  {
     ucSendCntLock=1; //加锁
     if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
     {
        uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个
中间没有串口数据过来
        ucSendLock=1;     //开自锁标志
     }
     ucSendCntLock=0; //解锁
  }
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  if(ucStatus!=0) //处于非待机的状态,Led闪烁
  {
     if(ucLedLock==0)//原子锁判断
         {
        uiLedCnt++; //Led闪烁计时器不断累加
         }
  }
  if(ucStatus==1) //处于正在通讯的状态,
  {
     if(ucSendTimeOutLock==0)  //原子锁判断
     {
         uiSendTimeOutCnt++;   //超时计时器累加
     }
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
```

```
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=1;  //点亮独立LED灯
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
/* 注释三:
* 为了保证串口中断接收的数据不丢失，必须设置IP = 0x10，相当于把串口中断设置为最高优先级，
* 这个时候，串口中断可以打断任何其他的中断服务函数实现嵌套，
*/
  IP =0x10;  //把串口中断设置为最高优先级，必须的。
  TH1=TL1=-(11059200L/12/32/9600);  //串口波特率为9600。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
```

```
    ucDigDot7=0;
    ucDigDot6=0;
    ucDigDot5=0;
    ucDigDot4=0;
    ucDigDot3=0;
    ucDigDot2=0;
    ucDigDot1=0;
    EA=1;       //开总中断
    ES=1;       //允许串口中断
    ET0=1;      //允许定时中断
    TR0=1;      //启动定时中断
}
```

复制代码

总结陈词:

前面花了大量篇幅详细地讲解了串口收发数据的程序框架，从下一节开始我讲解单片机掉电后数据保存的内容，欲知详情，请听下回分解-----利用AT24C02进行掉电后的数据保存。

（未完待续，下节更精彩，不要走开哦）

（未完待续，下节更精彩，不要走开哦）

第四十六节：利用AT24C02进行掉电后的数据保存。

开场白:

一个AT24C02可以存储256个字节，地址范围是(0至255)。利用AT24C02存储数据时，要教会大家六个知识点:

第一个: 单片机操作AT24C02的通讯过程也就是IIC的通讯过程， IIC通讯过程是一个要求一气呵成的通讯过程，中间不能被其它中断影响时序出错，因此在整个通讯过程中应该先关闭总中断，完成之后再开中断。

第二个: 在写入或者读取完一个字节之后，一定要加上一段延时时间。在11.0592M晶振的系统中，写入数据时经验值用delay_short(2000)，读取数据时经验值用delay_short(800)。否则在连续写入或者读取一串数据时容易丢失数据。如果一旦发现丢失数据，应该适当继续把这个时间延长，尤其是在写入数据时。

第三个: 如何初始化EEPROM数据的方法。系统第一次上电时，我们从EEPROM读取出来的数据有可能超出了范围，可能是ff。这个时候我们应该给它填入一个初始化的数据，这一步千万别漏了。

第四个: 在时序中，发送ACK确认信号时，要记得把数据线eeprom_sda_dr_s设置为输入的状态。对于51单片机来说，只要把eeprom_sda_dr_s=1就可以。而对于PIC或者AVR单片机来说，它们都是带方向寄存器的，就不能直接eeprom_sda_dr_s=1，而要直接修改方向寄存器，把它设置为输入状态。在本驱动程序中，我没有对ACK信号进行出错判断，因为我这么多年一直都是这样用也没出现过什么问题。

第五个: 提醒各位读者在硬件上应该注意的问题，单片机跟AT24C02通讯的2根IO口都要加上一个4.7K左右的上拉电阻。凡是在IIC通讯场合，都要加上拉电阻。AT24C02的WP引脚一定要接地，否则存不进数据。

第六个: 旧版的朱兆祺51学习板在硬件上有一个bug，AT24C02的第8个引脚VCC悬空了!!!，读者记得把它飞线连接到5V电源处。新版的朱兆祺51学习板已经改过来了。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

4个被更改后的参数断电后不丢失，数据可以保存，断电再上电后还是上一次最新被修改的数据。

显示和独立按键部分根据第29节的程序来改编，用朱兆祺51单片机学习板中的S1,S5,S9作为独立按键。

一共有4个窗口。每个窗口显示一个参数。

第8,7,6,5位数码管显示当前窗口，P-1代表第1个窗口，P-2代表第2个窗口，P-3代表第3个窗口，P-4代表第1个窗口。

第4,3,2,1位数码管显示当前窗口被设置的参数。范围是从0到9999。S1是加按键，按下此按键会依次增加当前窗口的参数。S5是减按键，按下此按键会依次减少当前窗口的参数。S9是切换窗口按键，按下此按键会依次循环切换不同的窗口。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void start24(void);  //开始位
void ack24(void);  //确认位
void stop24(void);  //停止位
unsigned char read24(void);  //读取一个字节的时序
void write24(unsigned char dd); //发送一个字节的时序
unsigned char read_eeprom(unsigned int address);   //从一个地址读取出一个字节数据
void write_eeprom(unsigned int address,unsigned char dd); //往一个地址存入一个字节数据
unsigned int read_eeprom_int(unsigned int address);   //从一个地址读取出一个int类型的数据
void write_eeprom_int(unsigned int address,unsigned int uiWriteData); //往一个地址存入一个int类型的数据
void T0_time(void);  //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit eeprom_scl_dr=P3^7;    //时钟线
sbit eeprom_sda_dr_sr=P3^6; //数据的输出线和输入线
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
```

```
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd4Update=0; //窗口4更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
```

```
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
      display_service(); //显示的窗口菜单服务程序
   }
}
//AT24C02驱动程序
void start24(void)  //开始位
{
    eeprom_sda_dr_sr=1;
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_sda_dr_sr=0;
        delay_short(15);
    eeprom_scl_dr=0;
}
void ack24(void)  //确认位时序
{
    eeprom_sda_dr_sr=1; //51单片机在读取数据之前要先置一，表示数据输入
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_scl_dr=0;
        delay_short(15);
//在本驱动程序中，我没有对ACK信号进行出错判断，因为我这么多年一直都是这样用也没出现过什么问题。
//有兴趣的朋友可以自己增加出错判断，不一定非要按我的方式去做。
}
void stop24(void)  //停止位
{
    eeprom_sda_dr_sr=0;
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_sda_dr_sr=1;
}
unsigned char read24(void)  //读取一个字节的时序
{
        unsigned char outdata,tempdata;
        outdata=0;
                eeprom_sda_dr_sr=1; //51单片机的IO口在读取数据之前要先置一，表示数据输入
        delay_short(2);
        for(tempdata=0;tempdata<8;tempdata++)
        {
            eeprom_scl_dr=0;
            delay_short(2);
            eeprom_scl_dr=1;
            delay_short(2);
            outdata<<=1;
            if(eeprom_sda_dr_sr==1)outdata++;
            eeprom_sda_dr_sr=1; //51单片机的IO口在读取数据之前要先置一，表示数据输入
```

```
            delay_short(2);
        }
    return(outdata);

}
void write24(unsigned char dd) //发送一个字节的时序
{
        unsigned char tempdata;
        for(tempdata=0;tempdata<8;tempdata++)
        {
                if(dd>=0x80)eeprom_sda_dr_sr=1;
                else eeprom_sda_dr_sr=0;
                dd<<=1;
                delay_short(2);
                eeprom_scl_dr=1;
                delay_short(4);
                eeprom_scl_dr=0;
        }
}
unsigned char read_eeprom(unsigned int address)   //从一个地址读取出一个字节数据
{
   unsigned char dd,cAddress;
   cAddress=address; //把低字节地址传递给一个字节变量。
/* 注释一:
  * IIC通讯过程是一个要求一气呵成的通讯过程，中间不能被其它中断影响时序出错，因此
  * 在整个通讯过程中应该先关闭总中断，完成之后再开中断。但是，这样就会引起另外一个新
  * 问题，如果关闭总中断的时间太长，会导致动态数码管不能及时均匀的扫描，在操作EEPROM时，
  * 数码管就会出现闪烁的现象，解决这个问题最好的办法就是在做项目中尽量不要用动态扫描数码管
  * 的方案，应该用静态显示的方案。那么程序上还有没有改善的方法？有的，下一节我会讲这个问题
  * 的改善方法。
  */
   EA=0; //禁止中断
   start24(); //IIC通讯开始
   write24(0xA0); //此字节包含读写指令和芯片地址两方面的内容。
                  //指令为写指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
   ack24(); //发送应答信号
   write24(cAddress); //发送读取的存储地址(范围是0至255)
   ack24(); //发送应答信号
   start24(); //开始
   write24(0xA1); //此字节包含读写指令和芯片地址两方面的内容。
                  //指令为读指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
   ack24(); //发送应答信号
   dd=read24(); //读取一个字节
   ack24(); //发送应答信号
   stop24();  //停止
/* 注释二:
  * 在写入或者读取完一个字节之后，一定要加上一段延时时间。在11.0592M晶振的系统中，
  * 写入数据时经验值用delay_short(2000)，读取数据时经验值用delay_short(800)。
  * 否则在连续写入或者读取一串数据时容易丢失数据。如果一旦发现丢失数据，
  * 应该适当继续把这个时间延长，尤其是在写入数据时。
```

```
  */
   delay_short(800);  //此处最关键，此处的延时时间一定要，而且要足够长，此处也是导致动态数码管闪烁的根本原因
   EA=1; //允许中断
   return(dd);
}
void write_eeprom(unsigned int address,unsigned char dd) //往一个地址存入一个字节数据
{
   unsigned char cAddress;
   cAddress=address; //把低字节地址传递给一个字节变量。
   EA=0; //禁止中断
   start24(); //IIC通讯开始
   write24(0xA0); //此字节包含读写指令和芯片地址两方面的内容。
                  //指令为写指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
   ack24(); //发送应答信号
   write24(cAddress);   //发送写入的存储地址(范围是0至255)
   ack24(); //发送应答信号
   write24(dd);  //写入存储的数据
   ack24(); //发送应答信号
   stop24();  //停止
   delay_short(2000);  //此处最关键，此处的延时时间一定要，而且要足够长，此处也是导致动态数码管闪烁的根本原因
   EA=1; //允许中断
}
unsigned int read_eeprom_int(unsigned int address)   //从一个地址读取出一个int类型的数据
{
   unsigned char ucReadDataH;
   unsigned char ucReadDataL;
   unsigned int  uiReadDate;
   ucReadDataH=read_eeprom(address);    //读取高字节
   ucReadDataL=read_eeprom(address+1);  //读取低字节
   uiReadDate=ucReadDataH;  //把两个字节合并成一个int类型数据
   uiReadDate=uiReadDate<<8;
   uiReadDate=uiReadDate+ucReadDataL;
   return uiReadDate;
}
void write_eeprom_int(unsigned int address,unsigned int uiWriteData) //往一个地址存入一个int类型的数据
{
   unsigned char ucWriteDataH;
   unsigned char ucWriteDataL;
   ucWriteDataH=uiWriteData>>8;
   ucWriteDataL=uiWriteData;
   write_eeprom(address,ucWriteDataH); //存入高字节
   write_eeprom(address+1,ucWriteDataL); //存入低字节
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示P--1窗口的数据
```

```
        if(ucWd1Update==1)  //窗口1要全部更新显示
{
           ucWd1Update=0;  //及时清零标志，避免一直进来扫描
           ucDigShow8=12;  //第8位数码管显示P
           ucDigShow7=11;  //第7位数码管显示-
           ucDigShow6=1;   //第6位数码管显示1
           ucDigShow5=10;  //第5位数码管显示无
          //先分解数据
                   ucTemp4=uiSetData1/1000;
                   ucTemp3=uiSetData1%1000/100;
                   ucTemp2=uiSetData1%100/10;
                   ucTemp1=uiSetData1%10;

                      //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
           if(uiSetData1<1000)
                       {
                          ucDigShow4=10;  //如果小于1000，千位显示无
                       }
           else
                       {
              ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                       }
           if(uiSetData1<100)
                       {
              ucDigShow3=10;  //如果小于100，百位显示无
                       }
                       else
                       {
              ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                       }
           if(uiSetData1<10)
                       {
              ucDigShow2=10;  //如果小于10，十位显示无
                       }
                       else
                       {
              ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
           }
           ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
         }
         break;
     case 2:  //显示P--2窗口的数据
         if(ucWd2Update==1)  //窗口2要全部更新显示
{
           ucWd2Update=0;  //及时清零标志，避免一直进来扫描
           ucDigShow8=12;  //第8位数码管显示P
           ucDigShow7=11;  //第7位数码管显示-
           ucDigShow6=2;   //第6位数码管显示2
           ucDigShow5=10;   //第5位数码管显示无
                   ucTemp4=uiSetData2/1000;     //分解数据
```

```
                ucTemp3=uiSetData2%1000/100;
                ucTemp2=uiSetData2%100/10;
                ucTemp1=uiSetData2%10;
            if(uiSetData2<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData2<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData2<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
          break;
     case 3:  //显示P--3窗口的数据
         if(ucWd3Update==1)  //窗口3要全部更新显示
{
            ucWd3Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=3;  //第6位数码管显示3
            ucDigShow5=10;   //第5位数码管显示无
                ucTemp4=uiSetData3/1000;     //分解数据
                ucTemp3=uiSetData3%1000/100;
                ucTemp2=uiSetData3%100/10;
                ucTemp1=uiSetData3%10;
            if(uiSetData3<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
```

```
            if(uiSetData3<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData3<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
         break;
      case 4:  //显示P--4窗口的数据
         if(ucWd4Update==1)  //窗口4要全部更新显示
{
            ucWd4Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=4;  //第6位数码管显示4
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData4/1000;     //分解数据
                    ucTemp3=uiSetData4%1000/100;
                    ucTemp2=uiSetData4%100/10;
                    ucTemp1=uiSetData4%10;
            if(uiSetData4<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData4<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData4<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
```

```
                            }
                            else
                            {
                    ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                }
                ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
    }
              break;
           }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
```

```
      uiKeyTimeCnt3++; //累加定时中断次数
      if(uiKeyTimeCnt3>const_key_time3)
      {
         uiKeyTimeCnt3=0;
         ucKeyLock3=1;  //自锁按键置位,避免一直触发
         ucKeySec=3;    //触发3号键
      }
   }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
                  {
                     case 1:
                  uiSetData1++;
                                 if(uiSetData1>9999) //最大值是9999
                                 {
                                    uiSetData1=9999;
                                 }
                           write_eeprom_int(0,uiSetData1); //存入EEPROM 由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd1Update=1;  //窗口1更新显示
                              break;
                     case 2:
                  uiSetData2++;
                                 if(uiSetData2>9999) //最大值是9999
                                 {
                                    uiSetData2=9999;
                                 }
                           write_eeprom_int(2,uiSetData2); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd2Update=1;  //窗口2更新显示
                              break;
                     case 3:
                  uiSetData3++;
                                 if(uiSetData3>9999) //最大值是9999
                                 {
                                    uiSetData3=9999;
                                 }
                           write_eeprom_int(4,uiSetData3); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd3Update=1;  //窗口3更新显示
                              break;
                     case 4:
                  uiSetData4++;
                                 if(uiSetData4>9999) //最大值是9999
                                 {
```

```
                                uiSetData4=9999;
                            }
                        write_eeprom_int(6,uiSetData4); //存入EEPROM,由于内部有延时函数，所以此处会引起数码管闪烁
                        ucWd4Update=1;  //窗口4更新显示
                            break;
                    }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;

        case 2:// 减按键 对应朱兆祺学习板的S5键
            switch(ucWd)  //在不同的窗口下，设置不同的参数
                    {
                       case 1:
                    uiSetData1--;
                                if(uiSetData1>9999)
                                {
                                   uiSetData1=0;  //最小值是0
                                }
                        write_eeprom_int(0,uiSetData1); //存入EEPROM,由于内部有延时函数，所以此处会引起数码管闪烁
                        ucWd1Update=1;  //窗口1更新显示
                            break;
                       case 2:
                    uiSetData2--;
                                if(uiSetData2>9999)
                                {
                                   uiSetData2=0;  //最小值是0
                                }
                        write_eeprom_int(2,uiSetData2); //存入EEPROM,由于内部有延时函数，所以此处会引起数码管闪烁
                        ucWd2Update=1;  //窗口2更新显示
                            break;
                       case 3:
                    uiSetData3--;
                                if(uiSetData3>9999)
                                {
                                   uiSetData3=0;  //最小值是0
                                }
                        write_eeprom_int(4,uiSetData3); //存入EEPROM,由于内部有延时函数，所以此处会引起数码管闪烁
                        ucWd3Update=1;  //窗口3更新显示
                            break;
                       case 4:
                    uiSetData4--;
                                if(uiSetData4>9999)
                                {
```

```
                                uiSetData4=0;  //最小值是0
                            }
                        write_eeprom_int(6,uiSetData4); //存入EEPROM,由于内部有延时函数，所以此处会引起数码管闪烁
                        ucWd4Update=1;  //窗口4更新显示
                            break;
                    }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 3:// 切换窗口按键 对应朱兆祺学习板的S9键
            ucWd++;  //切换窗口
                    if(ucWd>4)
                    {
                      ucWd=1;
                    }
            switch(ucWd)  //在不同的窗口下，在不同的窗口下，更新显示不同的窗口
                    {
                       case 1:
                            ucWd1Update=1;  //窗口1更新显示
                               break;
                       case 2:
                            ucWd2Update=1;  //窗口2更新显示
                               break;
                       case 3:
                            ucWd3Update=1;  //窗口3更新显示
                               break;
                       case 4:
                            ucWd4Update=1;  //窗口4更新显示
                               break;
                    }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;

      }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
```

```
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xfe);
            break;
    case 2:  //显示第2位
        ucDigShowTemp=dig_table[ucDigShow2];
            if(ucDigDot2==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xfd);
            break;
    case 3:  //显示第3位
        ucDigShowTemp=dig_table[ucDigShow3];
            if(ucDigDot3==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
            if(ucDigDot4==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
            if(ucDigDot5==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
            if(ucDigDot6==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
        ucDigShowTemp=dig_table[ucDigShow7];
            if(ucDigDot7==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
    }
```

```
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
       case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
```

```
    dig_hc595_sh_dr=0;     //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;     //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释三:
  * 此处多增加一个原子锁，作为中断与主函数共享数据的保护，实际上是借鉴了"红金龙吸味"关于原子锁的建议.
  */
  if(ucVoiceLock==0) //原子锁判断
  {
```

```
    if(uiVoiceCnt!=0)
    {
       uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
       beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

    }
    else
    {
       ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
       beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

    }
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
/* 注释四:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
```

```
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=0;    //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
/* 注释五:
* 如何初始化EEPROM数据的方法。在使用EEPROM时，这一步初始化很关键!
* 第一次上电时，我们从EEPROM读取出来的数据有可能超出了范围，可能是ff。
* 这个时候我们应该给它填入一个初始化的数据，这一步千万别漏了。另外，
* 由于int类型数据占用2个字节，所以以下4个数据挨着的地址是0,2,4,6.
*/
   uiSetData1=read_eeprom_int(0);  //读取uiSetData1，内部占用2个字节地址
   if(uiSetData1>9999)    //不在范围内
   {
       uiSetData1=0;    //填入一个初始化数据
       write_eeprom_int(0,uiSetData1); //存入uiSetData1，内部占用2个字节地址
   }
   uiSetData2=read_eeprom_int(2);  //读取uiSetData2，内部占用2个字节地址
   if(uiSetData2>9999)//不在范围内
   {
       uiSetData2=0;  //填入一个初始化数据
       write_eeprom_int(2,uiSetData2); //存入uiSetData2，内部占用2个字节地址
   }
   uiSetData3=read_eeprom_int(4);  //读取uiSetData3，内部占用2个字节地址
   if(uiSetData3>9999)//不在范围内
   {
       uiSetData3=0;  //填入一个初始化数据
       write_eeprom_int(4,uiSetData3); //存入uiSetData3，内部占用2个字节地址
   }
   uiSetData4=read_eeprom_int(6);  //读取uiSetData4，内部占用2个字节地址
   if(uiSetData4>9999)//不在范围内
   {
       uiSetData4=0;  //填入一个初始化数据
       write_eeprom_int(6,uiSetData4); //存入uiSetData4，内部占用2个字节地址
   }
}
```

复制代码

总结陈词:

IIC通讯过程是一个要求一气呵成的通讯过程，中间不能被其它中断影响时序出错，因此，在整个通讯过程中应该先

关闭总中断，完成之后再开中断。但是，这样就会引起另外一个新问题，如果关闭总中断的时间太长，会导致动态数码管不能及时均匀的扫描，在按键更改参数，内部操作EEPROM时，数码管就会出现短暂明显的闪烁现象，解决这个问题最好的办法就是在做项目中尽量不要用动态扫描数码管的方案，应该用静态显示的方案。那么在程序上还有没有改善这种现象的方法？当然有。欲知详情，请听下回分解-----操作AT24C02时，利用“一气呵成的定时器方式”改善数码管的闪烁现象。

（未完待续，下节更精彩，不要走开哦）

第四十七节：操作AT24C02时，利用“一气呵成的定时器延时”改善数码管的闪烁现象。

开场白：

上一节在按键更改参数时，会出现短暂明显的数码管闪烁现象。这节通过教大家使用新型延时函数可以有效的改善闪烁现象。要教会大家三个知识点：

第一个：如何编写一气呵成的定时器延时函数。

第二个：如何编写检查EEPROM芯片是否存在短路，虚焊或者芯片坏了的监控程序。

第三个：经过网友“cjseng”的提醒，我建议大家以后在用EEPROM芯片时，如果单片机IO口足够多，WP引脚应该专门接一个IO口，并且加一个上拉电阻，需要更改EEPROM存储数据时置低，其他任何一个时刻都置高，这样可以更加有效地保护EEPROM内部数据不会被意外更改。

具体内容，请看源代码讲解。

（1）硬件平台：

基于朱兆祺51单片机学习板。旧版的朱兆祺51学习板在硬件上有一个bug，AT24C02的第8个引脚VCC悬空了!!!，读者记得把它飞线连接到5V电源处。新版的朱兆祺51学习板已经改过来了。

（2）实现功能：

4个被更改后的参数断电后不丢失，数据可以保存，断电再上电后还是上一次最新被修改的数据。如果AT24C02短路，虚焊，或者坏了，系统可以检查出来，并且蜂鸣器会间歇性鸣叫报警。按更改参数按键时，数码管比上一节大大降低了闪烁现象。

显示和独立按键部分根据第29节的程序来改编，用朱兆祺51单片机学习板中的S1,S5,S9作为独立按键。

一共有4个窗口。每个窗口显示一个参数。

第8,7,6,5位数码管显示当前窗口，P-1代表第1个窗口，P-2代表第2个窗口，P-3代表第3个窗口，P-4代表第1个窗口。

第4,3,2,1位数码管显示当前窗口被设置的参数。范围是从0到9999。S1是加按键，按下此按键会依次增加当前窗口的参数。S5是减按键，按下此按键会依次减少当前窗口的参数。S9是切换窗口按键，按下此按键会依次循环切换不同的窗口。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_eeprom_1s    400   //大概1秒的时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
void delay_timer(unsigned int uiDelayTimerTemp); //一气呵成的定时器延时方式
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
```

```
void start24(void);  //开始位
void ack24(void);  //确认位
void stop24(void);  //停止位
unsigned char read24(void);  //读取一个字节的时序
void write24(unsigned char dd); //发送一个字节的时序
unsigned char read_eeprom(unsigned int address);   //从一个地址读取出一个字节数据
void write_eeprom(unsigned int address,unsigned char dd); //往一个地址存入一个字节数据
unsigned int read_eeprom_int(unsigned int address);   //从一个地址读取出一个int类型的数据
void write_eeprom_int(unsigned int address,unsigned int uiWriteData); //往一个地址存入一个int类型的数据
void T0_time(void);  //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void eeprom_alarm_service(void); //EEPROM出错报警
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit eeprom_scl_dr=P3^7;    //时钟线
sbit eeprom_sda_dr_sr=P3^6; //数据的输出线和输入线
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
```

```
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd2Update=0; //窗口2更新显示标志
unsigned char ucWd3Update=0; //窗口3更新显示标志
unsigned char ucWd4Update=0; //窗口4更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned int  uiSetData1=0;  //本程序中需要被设置的参数1
unsigned int  uiSetData2=0;  //本程序中需要被设置的参数2
unsigned int  uiSetData3=0;  //本程序中需要被设置的参数3
unsigned int  uiSetData4=0;  //本程序中需要被设置的参数4
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucDelayTimerLock=0; //原子锁
unsigned int  uiDelayTimer=0;
unsigned char ucCheckEeprom=0;  //检查EEPROM芯片是否正常
unsigned char ucEepromError=0; //EEPROM芯片是否正常的标志
unsigned char ucEepromLock=0;//原子锁
unsigned int  uiEepromCnt=0; //间歇性蜂鸣器报警的计时器
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
      display_service(); //显示的窗口菜单服务程序
          eeprom_alarm_service(); //EEPROM出错报警
   }
```

```
}
void eeprom_alarm_service(void) //EEPROM出错报警
{
  if(ucEepromError==1) //EEPROM出错
  {
      if(uiEepromCnt<const_eeprom_1s)  //大概1秒钟蜂鸣器响一次
      {
            ucEepromLock=1;  //原子锁加锁
        uiEepromCnt=0; //计时器清零
            ucEepromLock=0;  //原子锁解锁
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //蜂鸣器声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
      }
  }
}
//AT24C02驱动程序
void start24(void)  //开始位
{
    eeprom_sda_dr_sr=1;
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_sda_dr_sr=0;
        delay_short(15);
    eeprom_scl_dr=0;
}
void ack24(void)  //确认位时序
{
    eeprom_sda_dr_sr=1; //51单片机在读取数据之前要先置一，表示数据输入
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_scl_dr=0;
        delay_short(15);
//在本驱动程序中，我没有对ACK信号进行出错判断，因为我这么多年一直都是这样用也没出现过什么问题。
//有兴趣的朋友可以自己增加出错判断，不一定非要按我的方式去做。
}
void stop24(void)  //停止位
{
    eeprom_sda_dr_sr=0;
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_sda_dr_sr=1;
}
unsigned char read24(void)  //读取一个字节的时序
{
        unsigned char outdata,tempdata;
        outdata=0;
                eeprom_sda_dr_sr=1; //51单片机的IO口在读取数据之前要先置一，表示数据输入
        delay_short(2);
        for(tempdata=0;tempdata<8;tempdata++)
```

```
        {
            eeprom_scl_dr=0;
            delay_short(2);
            eeprom_scl_dr=1;
            delay_short(2);
            outdata<<=1;
            if(eeprom_sda_dr_sr==1)outdata++;
            eeprom_sda_dr_sr=1; //51单片机的IO口在读取数据之前要先置一，表示数据输入
            delay_short(2);
        }
    return(outdata);

}
void write24(unsigned char dd) //发送一个字节的时序
{
        unsigned char tempdata;
        for(tempdata=0;tempdata<8;tempdata++)
        {
                if(dd>=0x80)eeprom_sda_dr_sr=1;
                else eeprom_sda_dr_sr=0;
                dd<<=1;
                delay_short(2);
                eeprom_scl_dr=1;
                delay_short(4);
                eeprom_scl_dr=0;
        }
}
unsigned char read_eeprom(unsigned int address)   //从一个地址读取出一个字节数据
{
   unsigned char dd,cAddress;
   cAddress=address; //把低字节地址传递给一个字节变量。
   EA=0; //禁止中断
   start24(); //IIC通讯开始
   write24(0xA0); //此字节包含读写指令和芯片地址两方面的内容。
                  //指令为写指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
   ack24(); //发送应答信号
   write24(cAddress); //发送读取的存储地址(范围是0至255)
   ack24(); //发送应答信号
   start24(); //开始
   write24(0xA1); //此字节包含读写指令和芯片地址两方面的内容。
                  //指令为读指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
   ack24(); //发送应答信号
   dd=read24(); //读取一个字节
   ack24(); //发送应答信号
   stop24();  //停止
   EA=1; //允许中断
   delay_timer(2); //一气呵成的定时器延时方式，在延时的时候还可以动态扫描数码管
   return(dd);
}
void write_eeprom(unsigned int address,unsigned char dd) //往一个地址存入一个字节数据
```

```
{
   unsigned char cAddress;
   cAddress=address; //把低字节地址传递给一个字节变量。
   EA=0; //禁止中断
   start24(); //IIC通讯开始
   write24(0xA0); //此字节包含读写指令和芯片地址两方面的内容。
                  //指令为写指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
   ack24(); //发送应答信号
   write24(cAddress);   //发送写入的存储地址(范围是0至255)
   ack24(); //发送应答信号
   write24(dd);  //写入存储的数据
   ack24(); //发送应答信号
   stop24();  //停止
   EA=1; //允许中断
   delay_timer(4); //一气呵成的定时器延时方式，在延时的时候还可以动态扫描数码管
}
unsigned int read_eeprom_int(unsigned int address)   //从一个地址读取出一个int类型的数据
{
   unsigned char ucReadDataH;
   unsigned char ucReadDataL;
   unsigned int  uiReadDate;
   ucReadDataH=read_eeprom(address);    //读取高字节
   ucReadDataL=read_eeprom(address+1);  //读取低字节
   uiReadDate=ucReadDataH;  //把两个字节合并成一个int类型数据
   uiReadDate=uiReadDate<<8;
   uiReadDate=uiReadDate+ucReadDataL;
   return uiReadDate;
}
void write_eeprom_int(unsigned int address,unsigned int uiWriteData) //往一个地址存入一个int类型的数据
{
   unsigned char ucWriteDataH;
   unsigned char ucWriteDataL;
   ucWriteDataH=uiWriteData>>8;
   ucWriteDataL=uiWriteData;
   write_eeprom(address,ucWriteDataH); //存入高字节
   write_eeprom(address+1,ucWriteDataL); //存入低字节
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示P--1窗口的数据
            if(ucWd1Update==1)  //窗口1要全部更新显示
   {
               ucWd1Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=12;  //第8位数码管显示P
               ucDigShow7=11;  //第7位数码管显示-
               ucDigShow6=1;   //第6位数码管显示1
               ucDigShow5=10;  //第5位数码管显示无
              //先分解数据
```

```
                ucTemp4=uiSetData1/1000;
                ucTemp3=uiSetData1%1000/100;
                ucTemp2=uiSetData1%100/10;
                ucTemp1=uiSetData1%10;

                    //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
            if(uiSetData1<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData1<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData1<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
               ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
           }
           break;
        case 2:  //显示P--2窗口的数据
           if(ucWd2Update==1)  //窗口2要全部更新显示
{
               ucWd2Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=12;  //第8位数码管显示P
               ucDigShow7=11;  //第7位数码管显示-
               ucDigShow6=2;  //第6位数码管显示2
               ucDigShow5=10;   //第5位数码管显示无
                ucTemp4=uiSetData2/1000;     //分解数据
                ucTemp3=uiSetData2%1000/100;
                ucTemp2=uiSetData2%100/10;
                ucTemp1=uiSetData2%10;
            if(uiSetData2<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
```

```
                {
            ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                }
        if(uiSetData2<100)
                {
            ucDigShow3=10;  //如果小于100，百位显示无
                }
                else
                {
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                }
        if(uiSetData2<10)
                {
            ucDigShow2=10;  //如果小于10，十位显示无
                }
                else
                {
            ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
        }
        ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
      break;
    case 3:  //显示P--3窗口的数据
       if(ucWd3Update==1)  //窗口3要全部更新显示
{
        ucWd3Update=0;  //及时清零标志，避免一直进来扫描
        ucDigShow8=12;  //第8位数码管显示P
        ucDigShow7=11;  //第7位数码管显示-
        ucDigShow6=3;  //第6位数码管显示3
        ucDigShow5=10;   //第5位数码管显示无
              ucTemp4=uiSetData3/1000;     //分解数据
              ucTemp3=uiSetData3%1000/100;
              ucTemp2=uiSetData3%100/10;
              ucTemp1=uiSetData3%10;
        if(uiSetData3<1000)
                {
                   ucDigShow4=10;  //如果小于1000，千位显示无
                }
        else
                {
            ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                }
        if(uiSetData3<100)
                {
            ucDigShow3=10;  //如果小于100，百位显示无
                }
                else
                {
            ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                }
```

```
            if(uiSetData3<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
}
         break;
     case 4:  //显示P--4窗口的数据
         if(ucWd4Update==1)  //窗口4要全部更新显示
{
            ucWd4Update=0;  //及时清零标志，避免一直进来扫描
            ucDigShow8=12;  //第8位数码管显示P
            ucDigShow7=11;  //第7位数码管显示-
            ucDigShow6=4;  //第6位数码管显示4
            ucDigShow5=10;   //第5位数码管显示无
                    ucTemp4=uiSetData4/1000;     //分解数据
                    ucTemp3=uiSetData4%1000/100;
                    ucTemp2=uiSetData4%100/10;
                    ucTemp1=uiSetData4%10;
            if(uiSetData4<1000)
                        {
                           ucDigShow4=10;  //如果小于1000，千位显示无
                        }
            else
                        {
               ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                        }
            if(uiSetData4<100)
                        {
               ucDigShow3=10;  //如果小于100，百位显示无
                        }
                        else
                        {
               ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                        }
            if(uiSetData4<10)
                        {
               ucDigShow2=10;  //如果小于10，十位显示无
                        }
                        else
                        {
               ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
            }
            ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
 }
          break;
```

```
            }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;   //自锁按键置位,避免一直触发
        ucKeySec=1;     //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;   //自锁按键置位,避免一直触发
        ucKeySec=2;     //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;   //自锁按键置位,避免一直触发
        ucKeySec=3;     //触发3号键
     }
  }
```

```
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
                  {
                     case 1:
                  uiSetData1++;
                                  if(uiSetData1>9999) //最大值是9999
                                  {
                                     uiSetData1=9999;
                                  }
                           write_eeprom_int(0,uiSetData1); //存入EEPROM 由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd1Update=1;  //窗口1更新显示
                              break;
                     case 2:
                  uiSetData2++;
                                  if(uiSetData2>9999) //最大值是9999
                                  {
                                     uiSetData2=9999;
                                  }
                           write_eeprom_int(2,uiSetData2); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd2Update=1;  //窗口2更新显示
                              break;
                     case 3:
                  uiSetData3++;
                                  if(uiSetData3>9999) //最大值是9999
                                  {
                                     uiSetData3=9999;
                                  }
                           write_eeprom_int(4,uiSetData3); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd3Update=1;  //窗口3更新显示
                              break;
                     case 4:
                  uiSetData4++;
                                  if(uiSetData4>9999) //最大值是9999
                                  {
                                     uiSetData4=9999;
                                  }
                           write_eeprom_int(6,uiSetData4); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                           ucWd4Update=1;  //窗口4更新显示
                              break;
                  }
          ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
```

```
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
                {
                   case 1:
                uiSetData1--;
                                if(uiSetData1>9999)
                                {
                                   uiSetData1=0;  //最小值是0
                                }
                        write_eeprom_int(0,uiSetData1); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                        ucWd1Update=1;  //窗口1更新显示
                           break;
                   case 2:
                uiSetData2--;
                                if(uiSetData2>9999)
                                {
                                   uiSetData2=0;  //最小值是0
                                }
                        write_eeprom_int(2,uiSetData2); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                        ucWd2Update=1;  //窗口2更新显示
                           break;
                   case 3:
                uiSetData3--;
                                if(uiSetData3>9999)
                                {
                                   uiSetData3=0;  //最小值是0
                                }
                        write_eeprom_int(4,uiSetData3); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                        ucWd3Update=1;  //窗口3更新显示
                           break;
                   case 4:
                uiSetData4--;
                                if(uiSetData4>9999)
                                {
                                   uiSetData4=0;  //最小值是0
                                }
                        write_eeprom_int(6,uiSetData4); //存入EEPROM,由于内部有延时函数，所以此处会引
起数码管闪烁
                        ucWd4Update=1;  //窗口4更新显示
                           break;
                }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
```

```
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3:// 切换窗口按键 对应朱兆祺学习板的S9键
        ucWd++;  //切换窗口
                if(ucWd>4)
                {
                  ucWd=1;
                }
        switch(ucWd)  //在不同的窗口下，在不同的窗口下，更新显示不同的窗口
                {
                   case 1:
                         ucWd1Update=1;  //窗口1更新显示
                            break;
                   case 2:
                         ucWd2Update=1;  //窗口2更新显示
                            break;
                   case 3:
                         ucWd3Update=1;  //窗口3更新显示
                            break;
                   case 4:
                         ucWd4Update=1;  //窗口4更新显示
                            break;
                }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

  }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
```

```
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xfd);
            break;
    case 3:  //显示第3位
        ucDigShowTemp=dig_table[ucDigShow3];
            if(ucDigDot3==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
            if(ucDigDot4==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
            if(ucDigDot5==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
            if(ucDigDot6==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
        ucDigShowTemp=dig_table[ucDigShow7];
            if(ucDigDot7==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
        }
        dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
    case 8:  //显示第8位
        ucDigShowTemp=dig_table[ucDigShow8];
            if(ucDigDot8==1)
            {
                ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
```

```
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
```

```
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
```

```
    }
  }
  if(ucDelayTimerLock==0) //原子锁判断
  {
     if(uiDelayTimer>0)
         {
           uiDelayTimer--;    //一气呵成的定时器延时方式的计时器
         }

  }
  if(ucEepromError==1) //EEPROM出错
  {
      if(ucEepromLock==0)//原子锁判断
          {
             uiEepromCnt++;  //间歇性蜂鸣器报警的计时器
          }
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;    //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_timer(unsigned int uiDelayTimerTemp)
{
    ucDelayTimerLock=1; //原子锁加锁
    uiDelayTimer=uiDelayTimerTemp;
    ucDelayTimerLock=0; //原子锁解锁
/* 注释一:
  *延时等待，一直等到定时中断把它减到0为止.这种一气呵成的定时器方式，
```

```
  *可以在延时的时候动态扫描数码管，改善数码管的闪烁现象
  */
    while(uiDelayTimer!=0);  //一气呵成的定时器方式延时等待
}
void initial_myself(void)  //第一区 初始化单片机
{
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
/* 注释二:
  * 检查AT24C02芯片是否存在短路，虚焊，芯片坏了等不工作现象。
  * 在一个特定的地址里把数据读出来，如果发现不等于0x5a，则重新写入0x5a，再读出来
  * 判断是不是等于0x5a，如果不相等，则芯片有问题，出错报警提示。
  */
   ucCheckEeprom=read_eeprom(254); //判断AT24C02是否正常
   if(ucCheckEeprom!=0x5a)  //如果不等于特定内容。则重新写入数据再判断一次
   {
     write_eeprom(254,0x5a);  //重新写入标志数据
     ucCheckEeprom=read_eeprom(254); //判断AT24C02是否正常
         if(ucCheckEeprom!=0x5a)  //如果还是不等于特定数字，则芯片不正常
         {
            ucEepromError=1;  //表示AT24C02芯片出错报警
         }
   }
   uiSetData1=read_eeprom_int(0);  //读取uiSetData1，内部占用2个字节地址
   if(uiSetData1>9999)   //不在范围内
   {
       uiSetData1=0;   //填入一个初始化数据
       write_eeprom_int(0,uiSetData1); //存入uiSetData1，内部占用2个字节地址
   }
   uiSetData2=read_eeprom_int(2);  //读取uiSetData2，内部占用2个字节地址
   if(uiSetData2>9999)//不在范围内
   {
```

```
        uiSetData2=0;  //填入一个初始化数据
        write_eeprom_int(2,uiSetData2); //存入uiSetData2，内部占用2个字节地址
    }
    uiSetData3=read_eeprom_int(4);  //读取uiSetData3，内部占用2个字节地址
    if(uiSetData3>9999)//不在范围内
    {
        uiSetData3=0;  //填入一个初始化数据
        write_eeprom_int(4,uiSetData3); //存入uiSetData3，内部占用2个字节地址
    }
    uiSetData4=read_eeprom_int(6);  //读取uiSetData4，内部占用2个字节地址
    if(uiSetData4>9999)//不在范围内
    {
        uiSetData4=0;  //填入一个初始化数据
        write_eeprom_int(6,uiSetData4); //存入uiSetData4，内部占用2个字节地址
    }
}
```

复制代码

总结陈词:

下一节开始讲关于单片机驱动实时时钟芯片的内容，欲知详情，请听下回分解-----利用DS1302做一个实时时钟　。

（未完待续，下节更精彩，不要走开哦）

乐于分享，勇于质疑!

第四十八节：利用DS1302做一个实时时钟　。

开场白:

DS1302有两路独立电源输入，我们只要在其中一路电源上挂一个纽扣电池就可以实现掉电时钟继续跑的功能，纽扣电池作为备用电源必须比主电源的电压低一点。DS1302还给我们预留了一片RAM区，我们可以把一些数据存入到DS1302，只要DS1302的电池有电，那么它就相当于一个EEPROM。这个RAM区有什么用呢？因为RAM区的数据只要一掉电，所有的数据都会变成0x00或者0xff，也就是数据掉电会丢失，我们可以利用这个特点，可以在里面存入标志位数据，一旦发现这个数据改变了，就知道时钟的数据需要重新设置过，或者说明电池没电了。

在移植DS1302驱动程序中，有一个地方最容易出错，就是DS1302芯片的数据线DIO。我们编程时要特别留意这个IO口什么时候作为数据输入，什么时候作为数据输出，以便及时更改方向寄存器。对于51单片机，IO口在读取数据之前，要先置1。

这一节要教会大家六个知识点:

第一个: DS1302做实时时钟时，修改时间和读取时间的常见程序框架。

第二个: 如何编写监控备用电池电量耗尽的监控程序。

第三个: 在往DS1302写入数据修改时间之前，必须注意调整一下“日”的变量，因为每个月的最大天数是不一样的，有的一个月28天，有的一个月29天，有的一个月30天，有的一个月31天。

第四个: 本程序第一次出现了电平按键，跟之前讲的下降沿按键不一样，请留意我是何如用软件滤波的，以及它具体的实现代码。

第五个: 本程序第一次出现了一个按键按下去后，如果不松手就会触发两次事件，第一次是短按，第二次是长按3秒。请留意我是如何在之前的按键上略做修改就实现此功能的具体代码。

第六个: 继续加深了解按键与显示是如何紧密关联起来的程序框架。

具体内容，请看源代码讲解。

（1）　　硬件平台.

基于朱兆祺51单片机学习板。

旧版的朱兆祺51学习板在硬件上有一个bug，DS1302芯片附近的R43,R42两个电阻应该去掉，并且把R41的电阻换成0欧姆的电阻,或者直接短接起来。新版的朱兆祺51学习板已经改过来了。

（2）实现功能:

本程序有2两个窗口。

第1个窗口显示日期。显示格式“年-月-日”。注意中间有“-”分开。

第2个窗口显示时间。显示格式“时 分 秒”。注意中间没“-”，只有空格分开。

系统上电后，默认显示第2个窗口，实时显示动态的“时 分 秒”时间。此时按下S13按键不松手就会切换到显示日期的第1个窗口。松手后自动切换回第2个显示动态时间的窗口。

需要更改时间的时候，长按S9按键不松手超过3秒后，系统将进入修改时间的状态，切换到第1个日期窗口，并且显示“年”的两位数码管会闪烁，此时可以按S1或者S5加减按键修改年的参数，修改完年后，继续短按S9按键，会切换到“月”的参数闪烁状态，只要依次不断按下S9按键，就会依次切换年，月，日，时，分，秒的参数闪烁状态，最后修改完秒的参数后，系统会自动把我们修改设置的日期时间一次性写入DS1302芯片内部，达到修改日期时间的目的。

S13是电平变化按键，用来切换窗口的，专门用来查看当前日期。按下S13按键时显示日期窗口，松手后返回到显示实时时间的窗口。

本程序在使用过程中的注意事项:

（a）第一次上电时，蜂鸣器会报警，只要DS1302芯片的备用电池电量充足，这个时候断电再重启一次，就不会报警了。

(b)第一次上电时，时间没有走动，需要重新设置一下日期时间才可以。长按S9按键可以进入修改日期时间的状态。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_dpy_time_half  200  //数码管闪烁时间的半值
#define const_dpy_time_all   400  //数码管闪烁时间的全值 一定要比const_dpy_time_half 大
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
#define const_key_time17  1200  //长按超过3秒的时间
#define const_ds1302_0_5s  200   //大概0.5秒的时间
#define const_ds1302_sampling_time    360   //累计主循环次数的时间，每次刷新采样时钟芯片的时间
#define WRITE_SECOND    0x80    //DS1302内部的相关地址
#define WRITE_MINUTE    0x82
#define WRITE_HOUR      0x84
#define WRITE_DATE      0x86
#define WRITE_MONTH     0x88
#define WRITE_YEAR      0x8C
#define WRITE_CHECK     0xC2  //用来检查芯片的备用电池是否用完了的地址
#define READ_CHECK      0xC3  //用来检查芯片的备用电池是否用完了的地址
#define READ_SECOND     0x81
#define READ_MINUTE     0x83
#define READ_HOUR       0x85
#define READ_DATE       0x87
#define READ_MONTH      0x89
#define READ_YEAR       0x8D
#define WRITE_PROTECT   0x8E
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time(void);  //定时中断函数
```

```
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void ds1302_alarm_service(void); //ds1302出错报警
void ds1302_sampling(void); //ds1302采样程序，内部每秒钟采集更新一次
void Write1302 ( unsigned char addr, unsigned char dat );//修改时间的驱动
unsigned char Read1302 ( unsigned char addr );//读取时间的驱动
unsigned char bcd_to_number(unsigned char ucBcdTemp);  //BCD转原始数值
unsigned char number_to_bcd(unsigned char ucNumberTemp); //原始数值转BCD
//日调整 每个月份的日最大取值不同，有的最大28日，有的最大29日，有的最大30，有的最大31
unsigned char date_adjust(unsigned char ucYearTemp,unsigned char ucMonthTemp,unsigned char ucDateTemp);
//日调整
sbit SCLK_dr      =P1^3;
sbit DIO_dr_sr    =P1^4;
sbit DS1302_CE_dr =P1^5;
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit eeprom_scl_dr=P3^7;    //时钟线
sbit eeprom_sda_dr_sr=P3^6; //数据的输出线和输入线
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned int uiSampingCnt=0;   //采集Ds1302的计时器，每秒钟更新采集一次
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int uiKey4Cnt1=0;  //在软件滤波中，用到的变量
unsigned int uiKey4Cnt2=0;
unsigned char ucKey4Sr=1;  //实时反映按键的电平状态
unsigned char ucKey4SrRecord=0; //记录上一次按键的电平状态
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
```

```
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=2;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned char ucPart=0;//本程序的核心变量，局部显示变量。类似于二级菜单的变量。代表显示不同的局部。
unsigned char ucWd1Update=0; //窗口1更新显示标志
unsigned char ucWd2Update=1; //窗口2更新显示标志
unsigned char ucWd1Part1Update=0;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=0; //在窗口1中，局部2的更新显示标志
unsigned char ucWd1Part3Update=0; //在窗口1中，局部3的更新显示标志
unsigned char ucWd2Part1Update=0;  //在窗口2中，局部1的更新显示标志
unsigned char ucWd2Part2Update=0; //在窗口2中，局部2的更新显示标志
unsigned char ucWd2Part3Update=0; //在窗口2中，局部3的更新显示标志
unsigned char  ucYear=0;    //原始数据
unsigned char  ucMonth=0;
unsigned char  ucDate=0;
unsigned char  ucHour=0;
unsigned char  ucMinute=0;
unsigned char  ucSecond=0;
unsigned char  ucYearBCD=0;   //BCD码的数据
unsigned char  ucMonthBCD=0;
unsigned char  ucDateBCD=0;
unsigned char  ucHourBCD=0;
unsigned char  ucMinuteBCD=0;
unsigned char  ucSecondBCD=0;
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
unsigned char ucDelayTimerLock=0; //原子锁
unsigned int  uiDelayTimer=0;
unsigned char ucCheckDs1302=0;  //检查Ds1302芯片是否正常
unsigned char ucDs1302Error=0; //Ds1302芯片的备用电池是否用完了的报警标志
unsigned char ucDs1302Lock=0;//原子锁
unsigned int  uiDs1302Cnt=0; //间歇性蜂鸣器报警的计时器
unsigned char ucDpyTimeLock=0; //原子锁
unsigned int  uiDpyTimeCnt=0;  //数码管的闪烁计时器,放在定时中断里不断累加
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
```

```
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
          ds1302_sampling(); //ds1302采样程序，内部每秒钟采集更新一次
      display_service(); //显示的窗口菜单服务程序
          ds1302_alarm_service(); //ds1302出错报警
   }
}
/* 注释一:
  * 系统不用时时刻刻采集ds1302的内部数据，每隔一段时间更新采集一次就可以了。
  * 这个间隔时间应该小于或者等于1秒钟的时间，否则在数码管上看不到每秒钟的时间变化。
  */
void ds1302_sampling(void) //ds1302采样程序，内部每秒钟采集更新一次
{
   if(ucPart==0)  //当系统不是处于设置日期和时间的情况下
   {
      ++uiSampingCnt;  //累计主循环次数的时间
      if(uiSampingCnt>const_ds1302_sampling_time)  //每隔一段时间就更新采集一次Ds1302数据
          {
          uiSampingCnt=0;

          ucYearBCD=Read1302(READ_YEAR); //读取年
          ucMonthBCD=Read1302(READ_MONTH); //读取月
          ucDateBCD=Read1302(READ_DATE); //读取日
          ucHourBCD=Read1302(READ_HOUR); //读取时
          ucMinuteBCD=Read1302(READ_MINUTE); //读取分
          ucSecondBCD=Read1302(READ_SECOND); //读取秒
                  ucYear=bcd_to_number(ucYearBCD);  //BCD转原始数值
                  ucMonth=bcd_to_number(ucMonthBCD);  //BCD转原始数值
                  ucDate=bcd_to_number(ucDateBCD);  //BCD转原始数值
                  ucHour=bcd_to_number(ucHourBCD);  //BCD转原始数值
                  ucMinute=bcd_to_number(ucMinuteBCD);  //BCD转原始数值
```

```
                ucSecond=bcd_to_number(ucSecondBCD);  //BCD转原始数值
          ucWd2Update=1; //窗口2更新显示时间
          }
   }
}
//修改ds1302时间的驱动 ,注意，此处写入的是BCD码，
void Write1302 ( unsigned char addr, unsigned char dat )
{
     unsigned char i,temp;          //单片机驱动DS1302属于SPI通讯方式，根据我的经验，不用关闭中断
     DS1302_CE_dr=0;                                          //CE引脚为低，数据传送中止
     delay_short(1);
     SCLK_dr=0;                                               //清零时钟总线
     delay_short(1);
     DS1302_CE_dr = 1;                                        //CE引脚为高，逻辑控制有效
     delay_short(1);
                                                             //发送地址
     for ( i=0; i<8; i++ )                                   //循环8次移位
     {
        DIO_dr_sr = 0;
        temp = addr;
        if(temp&0x01)
        {
            DIO_dr_sr =1;
        }
        else
        {
            DIO_dr_sr =0;
        }
        delay_short(1);
        addr >>= 1;                                          //右移一位
        SCLK_dr = 1;
        delay_short(1);
        SCLK_dr = 0;
        delay_short(1);
     }
                                                             //发送数据
     for ( i=0; i<8; i++ )                                   //循环8次移位
     {
        DIO_dr_sr = 0;
        temp = dat;
        if(temp&0x01)
        {
            DIO_dr_sr =1;
        }
        else
        {
           DIO_dr_sr =0;
        }
        delay_short(1);
        dat >>= 1;                                           //右移一位
```

```
        SCLK_dr = 1;
        delay_short(1);
        SCLK_dr = 0;
        delay_short(1);
     }
     DS1302_CE_dr = 0;
     delay_short(1);
}
//读取Ds1302时间的驱动 ,注意, 此处读取的是BCD码,
unsigned char Read1302 ( unsigned char addr )
{
    unsigned char i,temp,dat1;
    DS1302_CE_dr=0;       //单片机驱动DS1302属于SPI通讯方式, 根据我的经验, 不用关闭中断
    delay_short(1);
    SCLK_dr=0;
    delay_short(1);
    DS1302_CE_dr = 1;
    delay_short(1);
                                                          //发送地址
    for ( i=0; i<8; i++ )                                 //循环8次移位
    {
       DIO_dr_sr = 0;
       temp = addr;
       if(temp&0x01)
       {
          DIO_dr_sr =1;
       }
       else
       {
          DIO_dr_sr =0;
       }
       delay_short(1);
       addr >>= 1;                                        //右移一位
       SCLK_dr = 1;
       delay_short(1);
       SCLK_dr = 0;
       delay_short(1);
    }

/* 注释二:
  * 51单片机IO口的特点, 在读取数据之前必须先输出高电平,
  * 如果是PIC,AVR等单片机, 这里应该把IO方向寄存器设置为输入
  */
   DIO_dr_sr =1;    //51单片机IO口的特点, 在读取数据之前必须先输出高电平,
   temp=0;
   for ( i=0; i<8; i++ )
   {
      temp>>=1;
      if(DIO_dr_sr==1)
      {
```

```
          temp=temp+0x80;
       }
           DIO_dr_sr =1;  //51单片机IO口的特点，在读取数据之前必须先输出高电平
       delay_short(1);
       SCLK_dr = 1;
       delay_short(1);
       SCLK_dr = 0;
       delay_short(1);
     }
     DS1302_CE_dr=0;
     delay_short(1);
     dat1=temp;
     return (dat1);
}
unsigned char bcd_to_number(unsigned char ucBcdTemp)  //BCD转原始数值
{
   unsigned char ucNumberResult=0;
   unsigned char ucBcdTemp10;
   unsigned char ucBcdTemp1;
   ucBcdTemp10=ucBcdTemp;
   ucBcdTemp10=ucBcdTemp10>>4;
   ucBcdTemp1=ucBcdTemp;
   ucBcdTemp1=ucBcdTemp1&0x0f;
   ucNumberResult=ucBcdTemp10*10+ucBcdTemp1;
   return ucNumberResult;
}
unsigned char number_to_bcd(unsigned char ucNumberTemp) //原始数值转BCD
{
   unsigned char ucBcdResult=0;
   unsigned char ucNumberTemp10;
   unsigned char ucNumberTemp1;
   ucNumberTemp10=ucNumberTemp;
   ucNumberTemp10=ucNumberTemp10/10;
   ucNumberTemp10=ucNumberTemp10<<4;
   ucNumberTemp10=ucNumberTemp10&0xf0;
   ucNumberTemp1=ucNumberTemp;
   ucNumberTemp1=ucNumberTemp1%10;
   ucBcdResult=ucNumberTemp10|ucNumberTemp1;
   return ucBcdResult;
}
//日调整 每个月份的日最大取值不同，有的最大28日，有的最大29日，有的最大30，有的最大31
unsigned char date_adjust(unsigned char ucYearTemp,unsigned char ucMonthTemp,unsigned char ucDateTemp)
//日调整
{
   unsigned char ucDayResult;
   unsigned int uiYearTemp;
   unsigned int uiYearYu;

   ucDayResult=ucDateTemp;
   switch(ucMonthTemp)  //根据不同的月份来修正不同的日最大值
```

```
   {
      case 2:  //二月份要计算是否是闰年
           uiYearTemp=2000+ucYearTemp;
           uiYearYu=uiYearTemp%4;
           if(uiYearYu==0) //闰年
           {
               if(ucDayResult>29)
               {
                  ucDayResult=29;
               }
           }
           else
           {
               if(ucDayResult>28)
               {
                  ucDayResult=28;
               }
           }
           break;
      case 4:
      case 6:
      case 9:
      case 11:
           if(ucDayResult>30)
           {
              ucDayResult=30;
           }
           break;
   }
   return ucDayResult;
}
void ds1302_alarm_service(void) //ds1302出错报警
{
    if(ucDs1302Error==1)  //备用电池的电量用完了报警提示
        {
           if(uiDs1302Cnt>const_ds1302_0_5s)  //大概0.5秒钟蜂鸣器响一次
           {
                   ucDs1302Lock=1;  //原子锁加锁
               uiDs1302Cnt=0; //计时器清零
                   ucDs1302Lock=0;  //原子锁解锁
               ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
               uiVoiceCnt=const_voice_short; //蜂鸣器声音触发，滴一声就停。
               ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt

           }
   }
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
```

```
{
    case 1:   //显示日期窗口的数据  数据格式 NN-YY-RR 年-月-日
        if(ucWd1Update==1)  //窗口1要全部更新显示
        {
           ucWd1Update=0;  //及时清零标志，避免一直进来扫描
           ucDigShow6=11;  //显示一杠"-"
           ucDigShow3=11;  //显示一杠"-"
           ucWd1Part1Update=1;  //局部年更新显示
           ucWd1Part2Update=1;  //局部月更新显示
           ucWd1Part3Update=1;  //局部日更新显示
        }
                  if(ucWd1Part1Update==1)//局部年更新显示
                  {
                     ucWd1Part1Update=0;
           ucTemp8=ucYear/10;  //年
           ucTemp7=ucYear%10;
           ucDigShow8=ucTemp8; //数码管显示实际内容
           ucDigShow7=ucTemp7;
                  }
                  if(ucWd1Part2Update==1)//局部月更新显示
                  {
                     ucWd1Part2Update=0;
           ucTemp5=ucMonth/10;  //月
           ucTemp4=ucMonth%10;
           ucDigShow5=ucTemp5; //数码管显示实际内容
           ucDigShow4=ucTemp4;
                  }
                  if(ucWd1Part3Update==1) //局部日更新显示
                  {
                     ucWd1Part3Update=0;
           ucTemp2=ucDate/10;  //日
           ucTemp1=ucDate%10;

           ucDigShow2=ucTemp2; //数码管显示实际内容
           ucDigShow1=ucTemp1;
                  }
         //数码管闪烁
        switch(ucPart)  //相当于二级菜单，根据局部变量的值，使对应的参数产生闪烁的动态效果。
        {
            case 0:  //都不闪烁
                 break;
            case 1:  //年参数闪烁
                 if(uiDpyTimeCnt==const_dpy_time_half)
                 {
                       ucDigShow8=ucTemp8; //数码管显示实际内容
                       ucDigShow7=ucTemp7;
                  }
                 else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                 {
```

```
                                    ucDpyTimeLock=1; //原子锁加锁
                        uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                            ucDpyTimeLock=0;  //原子锁解锁
                        ucDigShow8=10;    //数码管显示空，什么都不显示
                        ucDigShow7=10;
                    }
                    break;
                case 2:    //月参数闪烁
                    if(uiDpyTimeCnt==const_dpy_time_half)
                    {
                        ucDigShow5=ucTemp5; //数码管显示实际内容
                        ucDigShow4=ucTemp4;
                     }
                    else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                    {
                                    ucDpyTimeLock=1; //原子锁加锁
                        uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                            ucDpyTimeLock=0;  //原子锁解锁
                        ucDigShow5=10;    //数码管显示空，什么都不显示
                        ucDigShow4=10;
                    }
                   break;
                case 3:    //日参数闪烁
                    if(uiDpyTimeCnt==const_dpy_time_half)
                    {
                        ucDigShow2=ucTemp2; //数码管显示实际内容
                        ucDigShow1=ucTemp1;
                     }
                    else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                    {
                                    ucDpyTimeLock=1; //原子锁加锁
                        uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                            ucDpyTimeLock=0;  //原子锁解锁
                        ucDigShow2=10;    //数码管显示空，什么都不显示
                        ucDigShow1=10;
                    }
                   break;
            }
            break;
        case 2:    //显示时间窗口的数据  数据格式 SS FF MM 时 分 秒
            if(ucWd2Update==1)  //窗口2要全部更新显示
            {
               ucWd2Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow6=10;  //显示空
               ucDigShow3=10;  //显示空
               ucWd2Part3Update=1;  //局部时更新显示
               ucWd2Part2Update=1;  //局部分更新显示
               ucWd2Part1Update=1;  //局部秒更新显示
```

```
        }
                if(ucWd2Part1Update==1)//局部时更新显示
                {
                   ucWd2Part1Update=0;
           ucTemp8=ucHour/10;  //时
           ucTemp7=ucHour%10;
           ucDigShow8=ucTemp8; //数码管显示实际内容
           ucDigShow7=ucTemp7;
                 }
                 if(ucWd2Part2Update==1)//局部分更新显示
                 {
                   ucWd2Part2Update=0;
           ucTemp5=ucMinute/10;  //分
           ucTemp4=ucMinute%10;
           ucDigShow5=ucTemp5; //数码管显示实际内容
           ucDigShow4=ucTemp4;
                 }
                 if(ucWd2Part3Update==1) //局部秒更新显示
                 {
                   ucWd2Part3Update=0;
           ucTemp2=ucSecond/10;  //秒
           ucTemp1=ucSecond%10;

           ucDigShow2=ucTemp2; //数码管显示实际内容
           ucDigShow1=ucTemp1;
                 }
          //数码管闪烁
         switch(ucPart)  //相当于二级菜单，根据局部变量的值，使对应的参数产生闪烁的动态效果。
         {
             case 0:  //都不闪烁
                  break;
             case 1:  //时参数闪烁
                  if(uiDpyTimeCnt==const_dpy_time_half)
                  {
                        ucDigShow8=ucTemp8; //数码管显示实际内容
                        ucDigShow7=ucTemp7;
                   }
                  else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                  {
                                       ucDpyTimeLock=1; //原子锁加锁
                        uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                                              ucDpyTimeLock=0;  //原子锁解锁
                        ucDigShow8=10;    //数码管显示空，什么都不显示
                        ucDigShow7=10;
                  }
                  break;
             case 2:   //分参数闪烁
                  if(uiDpyTimeCnt==const_dpy_time_half)
                  {
```

```
                        ucDigShow5=ucTemp5; //数码管显示实际内容
                        ucDigShow4=ucTemp4;
                    }
                    else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                    {
                                        ucDpyTimeLock=1; //原子锁加锁
                        uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                                            ucDpyTimeLock=0;  //原子锁解锁
                        ucDigShow5=10;   //数码管显示空，什么都不显示
                        ucDigShow4=10;
                    }
                    break;
                case 3:   //秒参数闪烁
                    if(uiDpyTimeCnt==const_dpy_time_half)
                    {
                        ucDigShow2=ucTemp2; //数码管显示实际内容
                        ucDigShow1=ucTemp1;
                    }
                    else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                    {
                                        ucDpyTimeLock=1; //原子锁加锁
                        uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                                            ucDpyTimeLock=0;  //原子锁解锁
                        ucDigShow2=10;   //数码管显示空，什么都不显示
                        ucDigShow1=10;
                    }
                    break;
            }
            break;
     }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
```

```
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
/* 注释三:
  * 注意，此处把一个按键的短按和长按的功能都实现了。
  */
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //短按触发3号键
     }
  }
  else if(uiKeyTimeCnt3<const_key_time17)   //长按3秒
  {
     uiKeyTimeCnt3++; //累加定时中断次数
         if(uiKeyTimeCnt3==const_key_time17)  //等于3秒钟，触发17号长按按键
         {
            ucKeySec=17;    //长按3秒触发17号键
         }
  }
/* 注释四:
  * 注意，此处是电平按键的滤波抗干扰处理
  */
   if(key_sr4==1)  //对应朱兆祺学习板的S13键
   {
       uiKey4Cnt1=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiKey4Cnt2++; //类似独立按键去抖动的软件抗干扰处理
       if(uiKey4Cnt2>const_key_time4)
```

```
        {
            uiKey4Cnt2=0;
            ucKey4Sr=1;  //实时反映按键松手时的电平状态
        }
    }
    else
    {
        uiKey4Cnt2=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
        uiKey4Cnt1++;
        if(uiKey4Cnt1>const_key_time4)
        {
           uiKey4Cnt1=0;
           ucKey4Sr=0;  //实时反映按键按下时的电平状态
        }
    }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
               case 1:
                    switch(ucPart) //在不同的局部变量下，相当于二级菜单
                                   {
                                      case 1:  //年
                                           ucYear++;
                                                   if(ucYear>99)
                                                   {
                                                      ucYear=99;
                                                   }
                                      ucWd1Part1Update=1;  //更新显示
                                           break;
                                      case 2: //月
                                           ucMonth++;
                                                   if(ucMonth>12)
                                                   {
                                                      ucMonth=12;
                                                   }
                                      ucWd1Part2Update=1;  //更新显示

                                           break;
                                      case 3: //日
                                           ucDate++;
                                                   if(ucDate>31)
                                                   {
                                                      ucDate=31;
                                                   }
                                      ucWd1Part3Update=1;  //更新显示
```

```
                                        break;
                                    }
                        break;
                case 2:
                        switch(ucPart) //在不同的局部变量下，相当于二级菜单
                                    {
                                       case 1:  //时
                                            ucHour++;
                                                    if(ucHour>23)
                                                    {
                                                       ucHour=23;
                                                    }
                                            ucWd2Part1Update=1;  //更新显示

                                                    break;
                                       case 2: //分
                                            ucMinute++;
                                                    if(ucMinute>59)
                                                    {
                                                       ucMinute=59;
                                                    }
                                            ucWd2Part2Update=1;  //更新显示

                                                    break;
                                       case 3: //秒
                                            ucSecond++;
                                                    if(ucSecond>59)
                                                    {
                                                       ucSecond=59;
                                                    }
                                            ucWd2Part3Update=1;  //更新显示
                                                    break;
                                    }
                        break;

        }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
                case 1:
                        switch(ucPart) //在不同的局部变量下，相当于二级菜单
                                    {
                                       case 1:  //年
                                            ucYear--;
```

```
                                    if(ucYear>99)
                                    {
                                       ucYear=0;
                                    }
                              ucWd1Part1Update=1;  //更新显示
                                    break;
                                 case 2: //月
                                    ucMonth--;
                                    if(ucMonth<1)
                                    {
                                       ucMonth=1;
                                    }
                              ucWd1Part2Update=1;  //更新显示

                                    break;
                                 case 3: //日
                                    ucDate--;
                                    if(ucDate<1)
                                    {
                                       ucDate=1;
                                    }
                              ucWd1Part3Update=1;  //更新显示
                                    break;
                              }
        break;
case 2:
        switch(ucPart) //在不同的局部变量下，相当于二级菜单
                              {
                                 case 1:  //时
                                    ucHour--;
                                    if(ucHour>23)
                                    {
                                       ucHour=0;
                                    }
                              ucWd2Part1Update=1;  //更新显示

                                    break;
                                 case 2: //分
                                    ucMinute--;
                                    if(ucMinute>59)
                                    {
                                       ucMinute=0;
                                    }
                              ucWd2Part2Update=1;  //更新显示

                                    break;
                                 case 3: //秒
                                    ucSecond--;
                                    if(ucSecond>59)
                                    {
```

```
                                        ucSecond=0;
                                    }
                                ucWd2Part3Update=1;  //更新显示
                                    break;
                                }
                    break;

            }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 3://短按设置按键 对应朱兆祺学习板的S9键
            switch(ucWd)  //在不同的窗口下，设置不同的参数
            {
                case 1:
                    ucPart++;
                                if(ucPart>3)
                                {
                                   ucPart=1;
                                   ucWd=2; //切换到第二个窗口，设置时分秒
                                   ucWd2Update=1;  //窗口2更新显示
                                }
                            ucWd1Update=1;  //窗口1更新显示
                    break;
                case 2:
                            if(ucPart>0) //在窗口2的时候，要第一次激活设置时间，必须是长按3秒才可以
，这里短按激活不了第一次
                                {
                        ucPart++;
                                    if(ucPart>3)  //设置时间结束
                                   {
                                       ucPart=0;
/* 注释五:
  * 每个月份的天数最大值是不一样的，在写入ds1302时钟芯片内部数据前，应该做一次调整。
  * 有的月份最大28天，有的月份最大29天，有的月份最大30天，有的月份最大31天，
  */
                        ucDate=date_adjust(ucYear,ucMonth,ucDate); //日调整 避免日的数值在某个月份超
范围
                            ucYearBCD=number_to_bcd(ucYear);  //原始数值转BCD
                            ucMonthBCD=number_to_bcd(ucMonth); //原始数值转BCD
                              ucDateBCD=number_to_bcd(ucDate);  //原始数值转BCD
                            ucHourBCD=number_to_bcd(ucHour);  //原始数值转BCD
                            ucMinuteBCD=number_to_bcd(ucMinute);  //原始数值转BCD
                            ucSecondBCD=number_to_bcd(ucSecond);  //原始数值转BCD
                                        Write1302 (WRITE_PROTECT,0X00);          //禁止写保护
                        Write1302 (WRITE_YEAR,ucYearBCD);        //年修改
                        Write1302 (WRITE_MONTH,ucMonthBCD);      //月修改
                        Write1302 (WRITE_DATE,ucDateBCD);        //日修改
```

```
                        Write1302 (WRITE_HOUR,ucHourBCD);          //小时修改
                        Write1302 (WRITE_MINUTE,ucMinuteBCD);      //分钟修改
                        Write1302 (WRITE_SECOND,ucSecondBCD);      //秒位修改
                        Write1302 (WRITE_PROTECT,0x80);            //允许写保护
                                }
                                 ucWd2Update=1;  //窗口2更新显示
                              }
                break;

        }

        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 17://长按3秒设置按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 2:
                        if(ucPart==0) //处于非设置时间的状态下，要第一次激活设置时间，必须是长按
3秒才可以
                              {
                                  ucWd=1;
                    ucPart=1;  //进入到设置日期的状态下
                                  ucWd1Update=1;  //窗口1更新显示
                              }
                break;

        }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

  }

/* 注释六:
  * 注意，此处就是第一次出现的电平按键程序，跟以往的下降沿按键不一样。
  * ucKey4Sr是经过软件滤波处理后，直接反应IO口电平状态的变量.当电平发生
  * 变化时，就会切换到不同的显示界面，这里多用了一个ucKey4SrRecord变量
  * 记录上一次的电平状态，是为了避免一直刷新显示。
  */
  if(ucKey4Sr!=ucKey4SrRecord)  //说明S13的切换按键电平状态发生变化
  {
     ucKey4SrRecord=ucKey4Sr;//及时记录当前最新的按键电平状态  避免一直进来触发
         if(ucKey4Sr==1) //松手后切换到显示时间的窗口
         {
            ucWd=2;     //显示时分秒的窗口
```

```
                ucPart=0;  //进入到非设置时间的状态下
            ucWd2Update=1;  //窗口2更新显示
        }
        else  //按下去切换到显示日期的窗口
        {
            ucWd=1;   //显示年月日的窗口
                ucPart=0;  //进入到非设置时间的状态下
            ucWd1Update=1;  //窗口1更新显示
        }

  }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
                   if(ucDigDot4==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xf7);
               break;
      case 5:  //显示第5位
```

```
            ucDigShowTemp=dig_table[ucDigShow5];
                    if(ucDigDot5==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                    if(ucDigDot6==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                    if(ucDigDot7==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
```

```
        dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        dig_hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)dig_hc595_ds_dr=1;
        else dig_hc595_ds_dr=0;
        dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        dig_hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
```

```
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  if(ucDs1302Error>0) //EEPROM出错
  {
      if(ucDs1302Lock==0)//原子锁判断
          {
             uiDs1302Cnt++;  //间歇性蜂鸣器报警的计时器
          }
  }
  if(ucDpyTimeLock==0) //原子锁判断
  {
     uiDpyTimeCnt++;  //数码管的闪烁计时器
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
```

```
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
    ucDigDot8=0;   //小数点全部不显示
    ucDigDot7=0;
    ucDigDot6=0;
    ucDigDot5=0;
    ucDigDot4=0;
    ucDigDot3=0;
    ucDigDot2=0;
    ucDigDot1=0;
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
/* 注释七:
  * 检查ds1302芯片的备用电池电量是否用完了。
  * 在上电的时候，在一个特定的地址里把数据读出来，如果发现不等于0x5a，
  * 则是因为备用电池电量用完了，导致保存在ds1302内部RAM数据区的数据被更改，
  * 与此同时，应该重新写入一次0x5a，为下一次通电判断做准备
  */
    ucCheckDs1302=Read1302(READ_CHECK); //判断ds1302内部的数据是否被更改
    if(ucCheckDs1302!=0x5a)
    {
          Write1302 (WRITE_PROTECT,0X00);          //禁止写保护
```

```
        Write1302 (WRITE_CHECK,0x5a);              //重新写入标志数据,方便下一次更换新电池后的判断
        Write1302 (WRITE_PROTECT,0x80);            //允许写保护
            ucDs1302Error=1;  //表示ds1302备用电池没电了，报警提示更换新电池
    }
}
```

复制代码

总结陈词:

下一节开始讲单片机驱动温度传感器芯片的内容，欲知详情，请听下回分解-----利用DS18B20做一个温控器 。

第四十九节: 利用DS18B20做一个温控器 。

开场白:

DS18B20是一款常用的温度传感器芯片，它只占用单片机一根IO口，使用起来也特别方便。需要特别注意的是，正因为它只用一根IO口跟单片机通讯，因此读取一次温度值的通讯时间比较长，而且时序要求严格，在通讯期间不允许被单片机其它的中断干扰，因此在实际项目中，系统一旦选用了这款传感器芯片，就千万不要选用动态扫描数码管的显示方式。否则在关闭中断读取温度的时候，数码管的显示会有略微的“闪烁”现象。

DS18B20的测温范围是-55度至125度。在-10度至85度的温度范围内误差是+-0.5度，能满足大部分常用的测温要求。

这一节要教会大家三个知识点:

第一个: 大概了解一下DS18B20的驱动程序。

第二个: 做温控设备的时候，为了避免继电器在临界温度附近频繁跳动切换，应该设置一个缓冲温差。本程序的缓冲温差是2度。

第三个: 继续加深了解按键，显示，传感器它们三者是如何紧密关联起来的程序框架。

具体内容，请看源代码讲解。

（1） 硬件平台.

基于朱兆祺51单片机学习板。

（2）实现功能:

本程序只有1个窗口。这个窗口有2个局部显示。

第1个局部是第7,6,5位数码管，显示设定的温度。

第2个局部是第4,3,2,1位数码管，显示实际环境温度。其中第4位数码管显示正负符号位。

S1按键是加键，S5按键是减键。通过它们可以直接设置“设定温度”。

一个LED灯用来模拟工控的继电器。

当实际温度低于或者等于设定温度2度以下时，模拟继电器的LED灯亮。

当实际温度等于或者大于设定温度时，模拟继电器的LED灯灭。

当实际温度处于设定温度和设定温度减去2度的范围内，模拟继电器的LED维持现状，这个2度范围用来做缓冲温差，避免继电器在临界温度附近频繁跳动切换。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_ds18b20_sampling_time    180   //累计主循环次数的时间，每次刷新采样时钟芯片的时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
```

```
void T0_time(void);  //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void temper_control_service(void); //温控程序
void ds18b20_sampling(void); //ds18b20采样程序
void ds18b20_reset(); //复位ds18b20的时序
unsigned char ds_read_byte(void ); //读一字节
void ds_write_byte(unsigned char dat); //写一个字节
unsigned int get_temper();  //读取一次没有经过换算的温度数值
sbit dq_dr_sr=P2^6; //ds18b20的数据驱动线
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit led_dr=P3^5;  //LED灯，模拟工控中的继电器
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned int uiSampingCnt=0;   //采集Ds1302的计时器，每秒钟更新采集一次
unsigned char ucSignFlag=0; //正负符号。0代表正数，1代表负数，表示零下多少度。
unsigned long ulCurrentTemper=33; //实际温度
unsigned long ulSetTemper=26; //设定温度
unsigned int uiTemperTemp=0; //中间变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
```

```
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=1;  //因为本程序只有1个窗口，在实际项目中，此处的ucWd也可以省略不要
unsigned char ucWd1Part1Update=1;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=1; //在窗口1中，局部2的更新显示标志
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
      ds18b20_sampling(); //ds18b20采样程序
      temper_control_service(); //温控程序
      display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
  * 做温控设备的时候，为了避免继电器在临界温度附近频繁跳动切换，应该设置一个
  * 缓冲温差。本程序的缓冲温差是2度。
  */
void temper_control_service(void) //温控程序
{
   if(ucSignFlag==0) //是正数的前提下
   {
```

```
      if(ulCurrentTemper>=ulSetTemper)  //当实际温度大于等于设定温度时
      {
        led_dr=0; //模拟继电器的LED灯熄灭
      }
      else if(ulCurrentTemper<=(ulSetTemper-2))  //当实际温度小于等于设定温度2读以下时，这里的2是缓冲温差2度
      {
        led_dr=1; //模拟继电器的LED灯点亮
      }
   }
   else  //是负数，说明是零下多少度的情况下
   {
      led_dr=1; //模拟继电器的LED灯点亮
   }
}
void ds18b20_sampling(void) //ds18b20采样程序
{
      ++uiSampingCnt;  //累计主循环次数的时间
      if(uiSampingCnt>const_ds18b20_sampling_time)  //每隔一段时间就更新采集一次Ds18b20数据
          {
          uiSampingCnt=0;
          ET0=0;  //禁止定时中断
          uiTemperTemp=get_temper();  //读取一次没有经过换算的温度数值
          ET0=1; //开启定时中断
          if((uiTemperTemp&0xf800)==0xf800) //是负号
          {
                        ucSignFlag=1;
             uiTemperTemp=~uiTemperTemp;  //求补码
             uiTemperTemp=uiTemperTemp+1;
          }
          else //是正号
          {
                        ucSignFlag=0;
          }
          ulCurrentTemper=0; //把int数据类型赋给long类型之前要先清零
          ulCurrentTemper=uiTemperTemp;
          ulCurrentTemper=ulCurrentTemper*10; //为了先保留一位小数点，所以放大10倍，
          ulCurrentTemper=ulCurrentTemper>>4;  //往右边移动4位，相当于乘以0.0625. 此时保留了1位小数点，
          ulCurrentTemper=ulCurrentTemper+5;  //四舍五入
          ulCurrentTemper=ulCurrentTemper/10; //四舍五入后，去掉小数点
          ucWd1Part2Update=1; //局部2更新显示实时温度
          }
}
//ds18b20驱动程序
unsigned int get_temper()  //读取一次没有经过换算的温度数值
{
unsigned char temper_H;
unsigned char temper_L;
unsigned int ds18b20_data=0;
ds18b20_reset(); //复位ds18b20的时序
```

```
ds_write_byte(0xCC);
ds_write_byte(0x44);
ds18b20_reset(); //复位ds18b20的时序
ds_write_byte(0xCC);
ds_write_byte(0xBE);
temper_L=ds_read_byte();
temper_H=ds_read_byte();
ds18b20_data=temper_H;     //把两个字节合并成一个int数据类型
ds18b20_data=ds18b20_data<<8;
ds18b20_data=ds18b20_data|temper_L;
return ds18b20_data;
}
void ds18b20_reset() //复位ds18b20的时序
{
  unsigned char x;
  dq_dr_sr=1;
  delay_short(8);
  dq_dr_sr=0;
  delay_short(80);
  dq_dr_sr=1;
  delay_short(14);
  x=dq_dr_sr;
  delay_short(20);
}
void ds_write_byte(unsigned char date) //写一个字节
{
unsigned char  i;
for(i=0;i<8;i++)
{
  dq_dr_sr=0;
  dq_dr_sr=date&0x01;
  delay_short(5);
  dq_dr_sr=1;
  date=date>>1;
}
}
unsigned char ds_read_byte(void ) //读一字节
{
unsigned char i;
unsigned char date=0;
for(i=0;i<8;i++)
{
  dq_dr_sr=0;
  date=date>>1;
  dq_dr_sr=1;
  if(dq_dr_sr)
  {
     date=date|0x80;
  }
  delay_short(5);
```

```
}
return (date);
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //因为本程序只有1个窗口，在实际项目中，此处的ucWd也可以省略不要
   {
       case 1:
                       if(ucWd1Part1Update==1)//局部设定温度更新显示
                       {
                          ucWd1Part1Update=0;
                ucTemp8=10; //显示空
                          if(ulSetTemper>=100)
                          {
                   ucTemp7=ulSetTemper%1000/100; //显示设定温度的百位
                          }
                          else
                          {
                             ucTemp7=10; //显示空
                          }
                          if(ulSetTemper>=10)
                          {
                   ucTemp6=ulSetTemper%100/10; //显示设定温度的十位
                          }
                          else
                          {
                             ucTemp6=10; //显示空
                          }
                ucTemp5=ulSetTemper%10; //显示设定温度的个位
                ucDigShow8=ucTemp8; //数码管显示实际内容
                ucDigShow7=ucTemp7;
                ucDigShow6=ucTemp6;
                ucDigShow5=ucTemp5;
                       }
                       if(ucWd1Part2Update==1)//局部实际温度更新显示
                       {
                          if(ucSignFlag==0)  //正数
                          {
                   ucTemp4=10; //显示空
                          }
                          else  //负数，说明是零下多少度的情况下
                          {
                   ucTemp4=11; //显示负号-
                          }
                          if(ulCurrentTemper>=100)
                          {
                   ucTemp3=ulCurrentTemper%100/100; //显示实际温度的百位
                          }
                          else
                          {
```

```
                        ucTemp3=10; //显示空
                         }
                         if(u1CurrentTemper>=10)
                         {
                    ucTemp2=u1CurrentTemper%100/10;  //显示实际温度的十位
                         }
                         else
                         {
                    ucTemp2=10;  //显示空
                         }
                ucTemp1=u1CurrentTemper%10; //显示实际温度的个数位
                ucDigShow4=ucTemp4; //数码管显示实际内容
                ucDigShow3=ucTemp3;
                ucDigShow2=ucTemp2;
                ucDigShow1=ucTemp1;
                        }
            break;
      }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
```

```
    }
  }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd) //因为本程序只有1个窗口，在实际项目中，此处的ucWd也可以省略不要
          {
              case 1: //在窗口1下设置设定温度
                   ulSetTemper++;
                                  if(ulSetTemper>125)
                                  {
                                    ulSetTemper=125;
                                  }
                              ucWd1Part1Update=1; //更新显示设定温度
                   break;
          }
          ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

    case 2:// 减按键 对应朱兆祺学习板的S5键
          switch(ucWd) //因为本程序只有1个窗口，在实际项目中，此处的ucWd也可以省略不要
          {
               case 1: //在窗口1下设置设定温度
                    if(ulSetTemper>2)  //由于缓冲温差是2度，所以我人为规定最小允许设定的温度不能低于2度
                                    {
                                       ulSetTemper--;
                                    }
                          ucWd1Part1Update=1; //更新显示设定温度
                    break;

          }
          ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

  }

}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
```

```
{
   case 1:  //显示第1位
        ucDigShowTemp=dig_table[ucDigShow1];
                if(ucDigDot1==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfe);
            break;
   case 2:  //显示第2位
        ucDigShowTemp=dig_table[ucDigShow2];
                if(ucDigDot2==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfd);
            break;
   case 3:  //显示第3位
        ucDigShowTemp=dig_table[ucDigShow3];
                if(ucDigDot3==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
   case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
   case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
   case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
   case 7:  //显示第7位
```

```
            ucDigShowTemp=dig_table[ucDigShow7];
                    if(ucDigDot7==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
```

```
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
```

```
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  led_dr=0;//此处的LED灯模拟工控中的继电器
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
```

```
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=0;    //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

下一节开始讲单片机采集模拟信号的内容，欲知详情，请听下回分解-----利用ADC0832采集电压的模拟信号

第五十节：利用ADC0832采集电压信号，用平均法和区间法进行软件滤波处理。

开场白:
ADC0832是一款常用的8位AD采样芯片，通过它可以把外部的模拟电压信号转换成数字信号，然后给单片机进行换算，显示等处理。
这一节要教会大家五个知识点:
第一个: 分辨率的算法。有些书上说8位AD最高分辨可达到256级(0xff+1)，当输入电压是0---5V时，电压精度为19.53mV(5000mV除以256)，我认为这种说法是错误的。8位AD的最高分辨率应该是255级(0xff)，当输入电压是0---5V时，电压精度为19.61mV(5000mV除以255)。
第二个: 用求平均值的滤波法，可以使AD采样的数据更加圆滑，去除小毛刺。
第三个: 用区间滤波法，在一些干扰很大的场合，可以避免末尾小数点的数据频繁跳动。
第四个: 如何使系统可以采集到更高的电压。由于ADC0832直接采集的电压最大不能超过5V，如果要采集的最大电压是25V该怎么办？我们只要在外部多增加1个10K的电阻和1个40K的电阻组成分压电路，把25V分压成5V，然后再让ADC0832采样，这时采样到的数据只要乘以5的系数，就可以得到超过5V的实际电压。选择分压电阻时，阻值尽量不要太小，一般要10K级别以上，阻值大一点，对被采样的系统干扰影响就越小。
第五个: 如何有效保护AD通道口。我在一些电压不稳定的工控场合，一般是在AD通道口对负极反接一个瞬变二极管SA5.0A。当电压超过5V时，瞬变二极管会导通吸收掉多余的能量，把电压降下来，避免AD通道口烧坏。
具体内容，请看源代码讲解。
（1） 硬件平台.
基于朱兆祺51单片机学习板。
（2）实现功能:
本程序有2个局部显示。
第1个局部是第8,7,6,5位数码管，显示没有经过滤波处理的实际电压值。此时能观察到未经滤波的数据不太稳定，末尾

小数点数据会有跳动的现象
第2个局部是第4,3,2,1位数码管，显示经过平均法，区间法滤波的实际电压值。此时能观察到经过滤波后的数据很稳定，没有跳动的现象
系统保留3位小数点。手动调节可调电阻时，可以看到显示的数据在变化。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time(void);  //定时中断函数
void ad_sampling_service(void); //AD采样与处理的服务程序
sbit led_dr=P3^5;  //LED灯
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit adc0832_clk_dr     = P1^2;  // 定义adc0832的引脚
sbit adc0832_cs_dr      = P1^0;
sbit adc0832_data_sr_dr = P1^1;
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Part1Update=1;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=1; //在窗口1中，局部2的更新显示标志
```

```
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
unsigned char ucAD=0;   //AD值
unsigned char ucCheckAD=0; //用来做校验对比的AD值
unsigned long ulTemp=0;  //参与换算的中间变量
unsigned long ulTempFilterV=0; //参与换算的中间变量
unsigned long ulBackupFilterV=5000;  //备份最新采样数据的中间变量
unsigned char ucSamplingCnt=0; //统计采样的次数  本程序采样8次后求平均值
unsigned long ulV=0; //未经滤波处理的实时电压值
unsigned long ulFilterV=0; //经过滤波后的实时电压值
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      ad_sampling_service(); //AD采样与处理的服务程序
      display_service(); //显示的窗口菜单服务程序
   }
}
void ad_sampling_service(void) //AD采样与处理的服务程序
{
    unsigned char i;
    ucAD=0;   //AD值
    ucCheckAD=0; //用来做校验对比的AD值
    /* 片选信号置为低电平 */
```

```
adc0832_cs_dr = 0;
    /* 第一个脉冲，开始位 */
    adc0832_data_sr_dr = 1;
    adc0832_clk_dr  = 0;
delay_short(1);
    adc0832_clk_dr  = 1;
    /* 第二个脉冲，选择通道 */
    adc0832_data_sr_dr = 1;
    adc0832_clk_dr  = 0;
    adc0832_clk_dr  = 1;
    /* 第三个脉冲，选择通道 */
    adc0832_data_sr_dr = 0;
    adc0832_clk_dr  = 0;
    adc0832_clk_dr  = 1;
/* 数据线输出高电平 */
    adc0832_data_sr_dr = 1;
delay_short(2);
    /* 第一个下降沿 */
    adc0832_clk_dr  = 1;
    adc0832_clk_dr  = 0;
delay_short(1);
    /* AD值开始送出 */
    for (i = 0; i < 8; i++)
    {
    ucAD <<= 1;
            adc0832_clk_dr = 1;
            adc0832_clk_dr = 0;
            if (adc0832_data_sr_dr==1)
            {
        ucAD |= 0x01;
            }
    }
    /* 用于校验的AD值开始送出 */
    for (i = 0; i < 8; i++)
    {
    ucCheckAD >>= 1;
            if (adc0832_data_sr_dr==1)
            {
       ucCheckAD |= 0x80;
            }
            adc0832_clk_dr = 1;
            adc0832_clk_dr = 0;
    }

    /* 片选信号置为高电平 */
    adc0832_cs_dr = 1;
    if(ucCheckAD==ucAD)  //检验相等
    {

        ulTemp=0;  //把char类型数据赋值给long类型数据之前，必须先清零
```

```
        ulTemp=ucAD; //把char类型数据赋值给long类型数据,参与乘除法运算的数据，为了避免运算结果溢出，我都用long类型
/* 注释一:
* 因为保留3为小数点，这里的5000代表5.000V。ulTemp/255代表分辨率.
* 有些书上说8位AD最高分辩可达到256级(0xff+1)，我认为这种说法是错误的。
* 8位AD最高分辩应该是255级(0xff)，所以这里除以255，而不是256.
*/
        ulTemp=5000*ulTemp/255;  //进行电压换算
        ulV=ulTemp; //得到未经滤波处理的实时电压值
        ucWd1Part1Update=1; //局部更新显示未经滤波处理的电压
                ulTempFilterV=ulTempFilterV+ulTemp;  //累加8次后求平均值
        ucSamplingCnt++;  //统计已经采样累计的次数
                if(ucSamplingCnt>=8)
                {
/* 注释二:
* 求平均值滤波法，为了得到的数据更加圆滑，去除小毛刺。
* 向右边移动3位相当于除以8。
*/
                    ulTempFilterV=ulTempFilterV>>3; //求平均值滤波法
/* 注释三:
* 以下区间滤波法，为了避免末尾小数点的数据频繁跳动。
* 这里的20用于区间滤波法的正负偏差，这里的20代表0.020V。
* 意思是只要最近采集到的数据在正负0.020V偏差范围内，就不更新。
*/
                    if(ulBackupFilterV>=20)  //最近备份的上一次数据大于等于0.02V的情况下
                    {
                       if(ulTempFilterV<(ulBackupFilterV-20)||ulTempFilterV>(ulBackupFilterV+20)) //在正负0.020V偏差范围外，更新
                          {
                              ulBackupFilterV=ulTempFilterV;  //备份最新采样的数据,方便下一次对比判断
                       ulFilterV=ulTempFilterV; //得到经过滤波处理的实时电压值
                       ucWd1Part2Update=1; //局部更新显示经过滤波处理的电压
                                 }
                    }
                    else   //最近备份的上一次数据小于0.02V的情况下
                    {
                       if(ulTempFilterV>(ulBackupFilterV+20))  //在正0.020V偏差范围外，更新
                       {
                              ulBackupFilterV=ulTempFilterV;  //备份最新采样的数据,方便下一次对比判断
                       ulFilterV=ulTempFilterV; //得到经过滤波处理的实时电压值
                       ucWd1Part2Update=1; //局部更新显示经过滤波处理的电压
                              }

                    }
                    ucSamplingCnt=0;  //清零，为下一轮采样滤波作准备。
                    ulTempFilterV=0;
                }

        }
}
```

```
void display_service(void) //显示的窗口菜单服务程序
{
                        if(ucWd1Part1Update==1)//未经滤波处理的实时电压更新显示
                        {
                           ucWd1Part1Update=0;
                ucTemp8=u1V%10000/1000;  //显示电压值个位
                ucTemp7=u1V%1000/100;    //显示电压值小数点后第1位
                ucTemp6=u1V%100/10;      //显示电压值小数点后第2位
                ucTemp5=u1V%10;          //显示电压值小数点后第3位
                ucDigShow8=ucTemp8; //数码管显示实际内容
                ucDigShow7=ucTemp7;
                ucDigShow6=ucTemp6;
                ucDigShow5=ucTemp5;
                        }
                        if(ucWd1Part2Update==1)//经过滤波处理后的实时电压更新显示
                        {
                             ucWd1Part2Update=0;
                ucTemp4=u1FilterV%10000/1000;  //显示电压值个位
                ucTemp3=u1FilterV%1000/100;    //显示电压值小数点后第1位
                ucTemp2=u1FilterV%100/10;      //显示电压值小数点后第2位
                ucTemp1=u1FilterV%10;          //显示电压值小数点后第3位
                ucDigShow4=ucTemp4; //数码管显示实际内容
                ucDigShow3=ucTemp3;
                ucDigShow2=ucTemp2;
                ucDigShow1=ucTemp1;
                        }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
```

```
                if(ucDigDot3==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
        ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
        }
        dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
    case 8:  //显示第8位
        ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
}
ucDisplayDriveStep++;
if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
{
```

```
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
```

```
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
```

```
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  led_dr=0;//LED灯默认关闭
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=1;   //显示未经过滤波电压的小数点
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=1;  //显示经过滤波后电压的小数点
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节用区间滤波法虽然可以解决小数点后面的数据出现频繁跳动的现象，但是也存在一个小问题，就是精度受到了影响，比如我们设置的正负偏差是0.02V，那就意味着系统存在0.02V的误差。有没有更好的办法解决这个问题？如果系统的末尾数据一直不断处于频繁跳动中，那么只能牺牲一点精度，我认为用区间法已经是最好的解决办法了，但是经过本次实验，我观察到未经过滤波处理的数据只是偶尔跳动，并非频繁跳动，所以下一节我会给大家介绍一种不用牺牲精度，又可以很好滤波的方法。欲知详情，请听下回分解-----利用ADC0832采集电压信号，用连续N次一致性的方法进行滤波处理。

（未完待续，下节更精彩，不要走开哦）

第五十一节：利用ADC0832采集电压信号，用连续N次一致性的方法进行滤波处理。

开场白:

连续判断N次一致性的滤波法，是为了避免末尾小数点的数据偶尔跳动。这种滤波方法的原理跟我在按键扫描中去抖动的原理是一模一样的，被我频繁地应用在大量的工控项目中。

这一节要教会大家一个知识点：连续判断N次一致性的滤波法。

具体原理：当某个采样变量发生变化时，有两种可能，一种可能是外界的一个瞬间干扰。另一种可能是变量确实发生变化。为了有效去除干扰，当发现变量有变化时，我会连续采集N次，如果连续N次都是一致的结果，我才认为不是干扰。如果中间只要出现一次不一致，我会马上把计数器清零，这一步是精华，很关键。

具体内容，请看源代码讲解。

（1）硬件平台.

基于朱兆祺51单片机学习板。

（2）实现功能:

本程序有2个局部显示。
第1个局部是第8,7,6,5位数码管，显示没有经过滤波处理的实际电压值。此时能观察到未经滤波的数据不太稳定，末尾小数点数据会有跳动的现象
第2个局部是第4,3,2,1位数码管，显示经过特定算法滤波后的实际电压值。此时能观察到经过滤波后的数据很稳定，没有跳动的现象。而且显示的电压值跟未经过滤波的电压值几乎是完全一致，不会出现上一节用区间滤波法所留下的0.02V误差问题。
系统保留3位小数点。手动调节可调电阻时，可以看到显示的数据在变化。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_N   8  //连续判断N次一致性滤波方法中，N的取值
#define const_voice_short  40   //蜂鸣器短叫的持续时间
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time(void);  //定时中断函数
void ad_sampling_service(void); //AD采样与处理的服务程序
sbit led_dr=P3^5;  //LED灯
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit adc0832_clk_dr     = P1^2;  // 定义adc0832的引脚
sbit adc0832_cs_dr      = P1^0;
sbit adc0832_data_sr_dr = P1^1;
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
```

```
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Part1Update=1;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=1; //在窗口1中，局部2的更新显示标志
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp6=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
unsigned char ucAD=0;   //AD值
unsigned char ucCheckAD=0; //用来做校验对比的AD值
unsigned long ulTemp=0;  //参与换算的中间变量
unsigned long ulTempFilterV=0; //参与换算的中间变量
unsigned long ulBackupFilterV=5000;  //备份最新采样数据的中间变量
unsigned char ucSamplingCnt=0; //记录连续N次采样的计数器
unsigned long ulV=0; //未经滤波处理的实时电压值
unsigned long ulFilterV=0; //经过滤波后的实时电压值
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      ad_sampling_service(); //AD采样与处理的服务程序
      display_service(); //显示的窗口菜单服务程序
   }
}
void ad_sampling_service(void) //AD采样与处理的服务程序
{
```

```
unsigned char i;
ucAD=0;    //AD值
ucCheckAD=0; //用来做校验对比的AD值
/* 片选信号置为低电平 */
adc0832_cs_dr = 0;
    /* 第一个脉冲，开始位 */
    adc0832_data_sr_dr = 1;
    adc0832_clk_dr  = 0;
delay_short(1);
    adc0832_clk_dr  = 1;
    /* 第二个脉冲，选择通道 */
    adc0832_data_sr_dr = 1;
    adc0832_clk_dr  = 0;
    adc0832_clk_dr  = 1;
    /* 第三个脉冲，选择通道 */
    adc0832_data_sr_dr = 0;
    adc0832_clk_dr  = 0;
    adc0832_clk_dr  = 1;
/* 数据线输出高电平 */
    adc0832_data_sr_dr = 1;
delay_short(2);
    /* 第一个下降沿 */
    adc0832_clk_dr  = 1;
    adc0832_clk_dr  = 0;
delay_short(1);
    /* AD值开始送出 */
    for (i = 0; i < 8; i++)
    {
    ucAD <<= 1;
            adc0832_clk_dr = 1;
            adc0832_clk_dr = 0;
            if (adc0832_data_sr_dr==1)
            {
        ucAD |= 0x01;
            }
    }
    /* 用于校验的AD值开始送出 */
    for (i = 0; i < 8; i++)
    {
    ucCheckAD >>= 1;
            if (adc0832_data_sr_dr==1)
            {
       ucCheckAD |= 0x80;
            }
            adc0832_clk_dr = 1;
            adc0832_clk_dr = 0;
    }

    /* 片选信号置为高电平 */
    adc0832_cs_dr = 1;
```

```
        if(ucCheckAD==ucAD)   //检验相等
        {

            ulTemp=0;   //把char类型数据赋值给long类型数据之前，必须先清零
            ulTemp=ucAD; //把char类型数据赋值给long类型数据,参与乘除法运算的数据，为了避免运算结果溢出
，我都用long类型
/* 注释一:
* 因为保留3为小数点，这里的5000代表5.000V。ulTemp/255代表分辨率.
* 有些书上说8位AD最高分辨可达到256级(0xff+1)，我认为这种说法是错误的。
* 8位AD最高分辨应该是255级(0xff)，所以这里除以255，而不是256.
*/
            ulTemp=5000*ulTemp/255;   //进行电压换算
            ulV=ulTemp; //得到未经滤波处理的实时电压值
            ucWd1Part1Update=1; //局部更新显示未经滤波处理的电压
/* 注释二:
* 以下连续判断N次一致性的滤波法，为了避免末尾小数点的数据偶尔跳动。
* 这种滤波方法的原理跟我在按键扫描中的去抖动原理是一模一样的，被我频繁
* 地应用在大量的工控项目中。
* 具体原理: 当某个采样变量发生变化时，有两种可能，一种可能是外界的一个瞬间干扰。
* 另一种可能是变量确实发生变化。为了有效去除干扰，当发现变量有变化时，
* 我会连续采集N次，如果连续N次都是一致的结果，我才认为不是干扰。如果中间
* 只要出现一次不一致，我会马上把计数器清零，这一步是精华，很关键。
*
*/
                      if(ulTempFilterV!=ulTemp) //发现变量有变化
                      {
                         ucSamplingCnt++;     //计数器累加
                               if(ucSamplingCnt>const_N)  //如果连续N次都是一致的，则认为不是干扰。确实有
数据需要更新显示。这里的const_N取值是8
                                  {
                                     ucSamplingCnt=0;
                                     ulTempFilterV=ulTemp;    //及时保存更新了的数据，方便下一次有新数据对比做
准备
                     ulFilterV=ulTempFilterV; //得到经过滤波处理的实时电压值
                     ucWd1Part2Update=1; //局部更新显示经过滤波处理的电压
                                  }
                        }
                     else
                     {
                          ucSamplingCnt=0;  //只要出现一次不一致，我会马上把计数器清零，这一步是精华，很
关键。
                     }

        }
}
void display_service(void) //显示的窗口菜单服务程序
{
                         if(ucWd1Part1Update==1)//未经滤波处理的实时电压更新显示
                         {
                            ucWd1Part1Update=0;
```

```
                ucTemp8=u1V%10000/1000;   //显示电压值个位
                ucTemp7=u1V%1000/100;     //显示电压值小数点后第1位
                ucTemp6=u1V%100/10;       //显示电压值小数点后第2位
                ucTemp5=u1V%10;           //显示电压值小数点后第3位
                ucDigShow8=ucTemp8; //数码管显示实际内容
                ucDigShow7=ucTemp7;
                ucDigShow6=ucTemp6;
                ucDigShow5=ucTemp5;
                        }
                        if(ucWd1Part2Update==1)//经过滤波处理后的实时电压更新显示
                        {
                             ucWd1Part2Update=0;
                ucTemp4=u1FilterV%10000/1000;   //显示电压值个位
                ucTemp3=u1FilterV%1000/100;     //显示电压值小数点后第1位
                ucTemp2=u1FilterV%100/10;       //显示电压值小数点后第2位
                ucTemp1=u1FilterV%10;           //显示电压值小数点后第3位
                ucDigShow4=ucTemp4; //数码管显示实际内容
                ucDigShow3=ucTemp3;
                ucDigShow2=ucTemp2;
                ucDigShow1=ucTemp1;
                        }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
```

```
            break;
        case 4:  //显示第4位
            ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xef);
            break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
```

```
{
   unsigned char i;
   unsigned char ucTempData;
   dig_hc595_sh_dr=0;
   dig_hc595_st_dr=0;
   ucTempData=ucDigStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucDigStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)dig_hc595_ds_dr=1;
         else dig_hc595_ds_dr=0;
         dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         dig_hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   dig_hc595_st_dr=1;
   delay_short(1);
   dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
   dig_hc595_st_dr=0;
   dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
```

```
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
```

```
}
void initial_myself(void)  //第一区 初始化单片机
{
  led_dr=0;//LED灯默认关闭
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=1;   //显示未经过滤波电压的小数点
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=1;  //显示经过滤波后电压的小数点
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

在单片机AD采样的系统中，我常用的滤波方法有求平均值法，区间法，连续判断N次一致性这三种方法。读者可以根据不同的系统特点选择对应的滤波方法，有一些要求高的系统还可以把三种滤波方法混合在一起用。关于AD采样的知识到本节已经讲完，下一节会讲什么新内容呢？欲知详情，请听下回分解-----return语句鲜为人知的用法。

（未完待续，下节更精彩，不要走开哦）

第五十二节：程序后续升级修改的利器，return语句鲜为人知的用法。

开场白:

return语句经常用在带参数返回的函数中，字面上理解就是返回的意思，因此很多单片机初学者很容易忽略了return语句还有中断强行退出的功能。利用这个强行退出的功能，在项目后续程序的升级修改上很方便，还可以有效减少if语句的嵌套层数，使程序阅读起来很简洁。这一节要教大家return语句三个鲜为人知的用法:

第一个鲜为人知的用法: 在空函数里，可以插入很多个return语句，不仅仅是一个。

第二个鲜为人知的用法: return语句可以有效较少程序里条件判断语句的嵌套层数。

第三个鲜为人知的用法: return语句本身已经包含了类似break语句的功能,不管当前处于几层的内部循环嵌套，只要遇到return语句都可以强行退出全部循环，并且直接退出当前子程序，不执行当前子程序后面的任何语句，这个功能实在是太强大，太铁腕了。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

本程序实现的功能跟第三十九节是一摸一样的，唯一的差别就是在第三十九节的基础上，插入了几个return语句，用新的return语句替代原来的条件和循环判断语句。

波特率是: 9600 。

通讯协议: EB 00 55 XX YY

加无效填充字节后,上位机实际上应该发送: 00 EB 00 55 XX YY

其中第1位00是无效填充字节，防止由于硬件原因丢失第一个字节。

其中第2,3,4位EB 00 55就是数据头
后2位XX YY就是有效数据
任意时刻，单片机从电脑“串口调试助手”上位机收到的一串数据中，只要此数据中包含关键字EB 00 55 ，并且此关键字后面两个字节的数据XX YY 分别为01 02,那么蜂鸣器鸣叫一声表示接收的数据头和有效数据都是正确的。
也就是说，当在 串口助手往单片机发送十六进制数据串： eb 00 55 01 02 时，会听到蜂鸣器”滴”的一声。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释一:
* 以下函数说明了，在空函数里，可以插入很多个return语句。
* 用return语句非常便于后续程序的升级修改。
*/
void usart_service(void)  //串口服务程序,在main函数里
{

//     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //原来的语句，现在被两个return语句替代了
//     {
       if(uiSendCnt<const_receive_time)  //延时还没超过规定时间，直接退出本程序，不执行return后的任何语句。
           {
              return;  //强行退出本子程序，不执行以下任何语句
           }
           if(ucSendLock==0)  //不是最新一次接收到串口数据，直接退出本程序，不执行return后的任何语句。
           {
```

```
                return;  //强行退出本子程序，不执行以下任何语句
            }
/* 注释二:
* 以上两条return语句就相当于原来的一条if(uiSendCnt>=const_receive_time&&ucSendLock==1)语句。
* 用了return语句后，就明显减少了一个if嵌套。
*/
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
                    //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
//            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5)) //原来的语句，现在被两个return语句
替代了
            while(1) //死循环可以被以下return或者break语句中断，return本身已经包含了break语句功能。
            {
               if(uiRcregTotal<5)  //串口接收到的数据太少
                           {
                              uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据
                                  return;  //强行退出while(1)循环嵌套，直接退出本程序，不执行以下任何语
句
                           }
                           if(uiRcMoveIndex>(uiRcregTotal-5)) //数组缓冲区的数据已经处理完
                           {
                              uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据
                                  return;  //强行退出while(1)循环嵌套，直接退出本程序，不执行以下任何语
句
                           }
/* 注释三:
* 以上两条return语句就相当于原来的一条while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))语句。
* 以上两个return语句的用法，同时说明了return本身已经包含了break语句功能,不管当前处于几层的内部循环嵌套,
* 都可以强行退出循环，并且直接退出本程序。
*/
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
               {
                  if(ucRcregBuf[uiRcMoveIndex+3]==0x01&&ucRcregBuf[uiRcMoveIndex+4]==0x02)  //有效数据01
02的判断
                  {
                       uiVoiceCnt=const_voice_short; //蜂鸣器发出声音，说明数据头和有效数据都接收正确
                  }
                  break;   //退出while(1)循环
               }
               uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
            }

            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

//     }

}
void T0_time(void) interrupt 1    //定时中断
{
```

```
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
         uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
     ucSendLock=1;      //开自锁标志
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
```

```
            {
               ; //一个分号相当于执行一条空语句
            }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

我在第一节就告诉读者了，搞单片机开发如果不会C语言的指针也没关系，不会影响做项目。我本人平时做项目时，也很少用指针，只有在三种场合下我才会用指针，因为在这三种场合下，用了指针感觉程序阅读起来更加清爽了。所以，指针还是有它独到的好处，有哪三种好处？欲知详情，请听下回分解-----指针的第一大好处，让一个函数可以封装多个相当于return语句返回的参数。

（未完待续，下节更精彩，不要走开哦）

第五十三节：指针的第一大好处，让一个函数可以封装多个相当于return语句返回的参数。

开场白:

当我们想把某种算法通过一个函数来实现的时候，如果不会指针，那么只有两种方法。

第1种：用不带参数返回的空函数。这是最原始的做法，也是我当年刚毕业就开始做项目的时候经常用的方法。它完全依靠全局变量作为函数的输入和输出口。我们要用到这个函数，就要把参与运算的变量直接赋给对应的输入全局变量，调用一次函数之后，再找到对应的输出变量，这些输出变量就是我们要的结果。这种方法的缺点是阅读不直观，封装性不强，没有面对用户的输入输出接口。

第2种：用return返回参数和带输入形参的函数，这种方法已经具备了完整的输入和输出性能，比第1种方法直观多了。但是这种方法有它的局限性，因为return只能返回一个变量，如果要用在返回多个输出结果的函数中，就无能为力了，这时候该怎么办？就必须用指针了，也就是我下面讲到的第3种方法。

这一节要教大家一个知识点：通过指针，让函数可以返回多个变量。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

通过电脑串口调试助手，往单片机发送EB 00 55 XX YY 指令，其中EB 00 55是数据头， XX是被除数，YY是除数。单片机收到指令后就会返回6个数据，最前面两个数据是第1种运算方式的商和余数，中间两个数据是第2种运算方式的商

和余数，最后两个数据是第3种运算方式的商和余数。
比如电脑发送: EB 00 55 08 02
单片机就返回: 04 00 04 00 04 00 (04是商，00是余数)
串口程序的接收部分请参考第39节。串口程序的发送部分请参考第42节。
波特率是: 9600 。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void chu_fa_yun_suan_1(void);//第1种方法 求商和余数
unsigned char get_shang_2(unsigned char ucBeiChuShuTemp,unsigned char ucChuShuTemp); //第2种方法 求商
unsigned char get_yu_2(unsigned char ucBeiChuShuTemp,unsigned char ucChuShuTemp); //第2种方法 求余数
void chu_fa_yun_suan_3(unsigned char ucBeiChuShuTemp,unsigned char ucChuShuTemp,unsigned char *p_ucShangTemp,unsigned char *p_ucYuTemp);//第3种方法 求商和余数
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucBeiChuShu_1=0;  //第1种方法中的被除数
unsigned char ucChuShu_1=1;     //第1种方法中的除数
unsigned char ucShang_1=0;      //第1种方法中的商
unsigned char ucYu_1=0;         //第1种方法中的余数
unsigned char ucBeiChuShu_2=0;  //第2种方法中的被除数
unsigned char ucChuShu_2=1;     //第2种方法中的除数
unsigned char ucShang_2=0;      //第2种方法中的商
unsigned char ucYu_2=0;         //第2种方法中的余数
unsigned char ucBeiChuShu_3=0;  //第3种方法中的被除数
unsigned char ucChuShu_3=1;     //第3种方法中的除数
unsigned char ucShang_3=0;      //第3种方法中的商
unsigned char ucYu_3=0;         //第3种方法中的余数
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
    {
```

```
        usart_service();  //串口服务程序
    }
}
/* 注释一:
* 第1种方法，用不带参数返回的空函数，这是最原始的做法，也是我当年刚毕业
* 就开始做项目的时候经常用的方法。它完全依靠全局变量作为函数的输入和输出口。
* 我们要用到这个函数，就要把参与运算的变量直接赋给对应的输入全局变量，
* 调用一次函数之后，再找到对应的输出变量，这些输出变量就是我们要的结果。
* 在本函数中，被除数ucBeiChuShu_1和除数ucChuShu_1就是输入全局变量，
* 商ucShang_1和余数ucYu_1就是输出全局变量。这种方法的缺点是阅读不直观，
* 封装性不强，没有面对用户的输入输出接口，
*/
void chu_fa_yun_suan_1(void)//第1种方法 求商和余数
{
    if(ucChuShu_1==0) //如果除数为0，则商和余数都为0
    {
        ucShang_1=0;
            ucYu_1=0;
    }
    else
    {
        ucShang_1=ucBeiChuShu_1/ucChuShu_1;  //求商
        ucYu_1=ucBeiChuShu_1%ucChuShu_1;  //求余数
    }
}
/* 注释二:
* 第2种方法，用return返回参数和带输入形参的函数，这种方法已经具备了完整的输入和输出性能，
* 比第1种方法直观多了。但是这种方法有它的局限性，因为return只能返回一个变量，
* 如果要用在返回多个输出结果的函数中，就无能为力了。比如本程序，就不能同时输出
* 商和余数，只能分两个函数来做。如果要在一个函数中同时输出商和余数，该怎么办?
* 这个时候就必须用指针了，也就是我下面讲到的第3种方法。
*/
unsigned char get_shang_2(unsigned char ucBeiChuShuTemp,unsigned char ucChuShuTemp) //第2种方法 求商
{
    unsigned char ucShangTemp;
    if(ucChuShuTemp==0) //如果除数为0，则商为0
    {
        ucShangTemp=0;
    }
    else
    {
        ucShangTemp=ucBeiChuShuTemp/ucChuShuTemp;  //求商
    }
    return ucShangTemp; //返回运算后的结果 商
}
unsigned char get_yu_2(unsigned char ucBeiChuShuTemp,unsigned char ucChuShuTemp) //第2种方法 求余数
{
    unsigned char ucYuTemp;
    if(ucChuShuTemp==0) //如果除数为0，则余数为0
    {
```

```
      ucYuTemp=0;
   }
   else
   {
      ucYuTemp=ucBeiChuShuTemp%ucChuShuTemp;    //求余数
   }
   return ucYuTemp; //返回运算后的结果 余数
}
/* 注释三:
* 第3种方法，用带指针的函数，就可以顺心所欲，不受return的局限，想输出多少个
* 运算结果都可以，赞一个！在本函数中，ucBeiChuShuTemp和ucChuShuTemp是输入变量，
* 它们不是指针，所以不具备输出接口属性。*p_ucShangTemp和*p_ucYuTemp是输出变量，
* 因为它们是指针，所以具备输出接口属性。
*/
void chu_fa_yun_suan_3(unsigned char ucBeiChuShuTemp,unsigned char ucChuShuTemp,unsigned char
*p_ucShangTemp,unsigned char *p_ucYuTemp)//第3种方法 求商和余数
{
   if(ucChuShuTemp==0) //如果除数为0，则商和余数都为0
   {
      *p_ucShangTemp=0;
          *p_ucYuTemp=0;
   }
   else
   {
      *p_ucShangTemp=ucBeiChuShuTemp/ucChuShuTemp;  //求商
      *p_ucYuTemp=ucBeiChuShuTemp%ucChuShuTemp;  //求余数
   }
}
void usart_service(void)  //串口服务程序,在main函数里
{

     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
               {
                  //第1种运算方法，依靠全局变量
                  ucBeiChuShu_1=ucRcregBuf[uiRcMoveIndex+3]; //被除数
                  ucChuShu_1=ucRcregBuf[uiRcMoveIndex+4];  //除数
                                    chu_fa_yun_suan_1(); //调用一次空函数就出结果了，结果保存在ucShang_1和
ucYu_1全局变量中
                                    eusart_send(ucShang_1); //把运算结果返回给上位机观察
                                    eusart_send(ucYu_1);//把运算结果返回给上位机观察
                  //第2种运算方法，依靠两个带return语句的返回函数
                  ucBeiChuShu_2=ucRcregBuf[uiRcMoveIndex+3]; //被除数
```

```
                    ucChuShu_2=ucRcregBuf[uiRcMoveIndex+4];  //除数
                    ucShang_2=get_shang_2(ucBeiChuShu_2,ucChuShu_2); //第2种方法 求商
                    ucYu_2=get_yu_2(ucBeiChuShu_2,ucChuShu_2); //第2种方法 求余数
                                    eusart_send(ucShang_2); //把运算结果返回给上位机观察
                                    eusart_send(ucYu_2);//把运算结果返回给上位机观察
                    //第3种运算方法，依靠指针
                    ucBeiChuShu_3=ucRcregBuf[uiRcMoveIndex+3]; //被除数
                    ucChuShu_3=ucRcregBuf[uiRcMoveIndex+4];  //除数
/* 注释四:
* 注意，由于商和余数是指针形参，我们代入的变量必须带地址符号& 。比如&ucShang_3和&ucYu_3。
* 因为我们是把变量的地址传递进去的。
*/
                                    chu_fa_yun_suan_3(ucBeiChuShu_3,ucChuShu_3,&ucShang_3,&ucYu_3);//第3种
方法 求商和余数
                                    eusart_send(ucShang_3); //把运算结果返回给上位机观察
                                    eusart_send(ucYu_3);//把运算结果返回给上位机观察
                    break;   //退出循环
                }
                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
            }

            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

        }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目
来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这
个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
```

```
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                    //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
```

```
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节讲了指针的第一大好处，它的第二大好处是什么？欲知详情，请听下回分解-----指针的第二大好处，指针作为数组在函数内部的化身。

第五十四节：指针的第二大好处，指针作为数组在函数中的输入接口。

开场白:

如果不会指针，当我们想把一个数组的数据传递进某个函数内部的时候，只能通过全局变量的方式，这种方法的缺点是阅读不直观，封装性不强，没有面对用户的输入接口。

针对以上问题，这一节要教大家一个知识点：通过指针，为函数增加一个数组输入接口。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

把5个随机数据按从大到小排序，用冒泡法来排序。

通过电脑串口调试助手，往单片机发送EB 00 55 08 06 09 05 07 指令，其中EB 00 55是数据头，08 06 09 05 07 是参与排序的5个随机原始数据。单片机收到指令后就会返回13个数据，最前面5个数据是第1种方法的排序结果，中间3个数据EE EE EE是第1种和第2种的分割线，为了方便观察，没实际意义。最后5个数据是第2种方法的排序结果.

比如电脑发送: EB 00 55 08 06 09 05 07

单片机就返回: 09 08 07 06 05 EE EE EE 09 08 07 06 05

串口程序的接收部分请参考第39节。串口程序的发送部分请参考第42节。

波特率是: 9600 。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_array_size  5  //参与排序的数组大小
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
```

```
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void big_to_small_sort_1(void);//第1种方法 把一个数组从大小小排序
void big_to_small_sort_2(unsigned char *p_ucInputBuffer);//第2种方法 把一个数组从大小小排序
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucUsartBuffer[const_array_size];  //从串口接收到的需要排序的原始数据
unsigned char ucGlobalBuffer_1[const_array_size]; //第1种方法，参与具体排序算法的全局变量数组
unsigned char ucGlobalBuffer_2[const_array_size]; //第2种方法，参与具体排序算法的全局变量数组
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释一:
* 第1种方法，用不带输入输出接口的空函数，这是最原始的做法，它完全依靠
* 全局变量作为函数的输入和输出口。我们要用到这个函数，就要把参与运算
* 的变量直接赋给对应的输入全局变量，调用一次函数之后，再找到对应的
* 输出全局变量，这些输出全局变量就是我们要的结果。
* 在本函数中，ucGlobalBuffer_1[const_array_size]既是输入全局变量，也是输出全局变量，
* 这种方法的缺点是阅读不直观，封装性不强，没有面对用户的输入输出接口，
*/
void big_to_small_sort_1(void)//第1种方法 把一个数组从大小小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
/* 注释二:
* 以下就是著名的 冒泡法排序。这个方法几乎所有的C语言大学教材都讲过了。大家在百度上可以直接
* 搜索到它的工作原理和详细的讲解步骤，我就不再详细讲解了。
*/
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-
i)
          {
```

```
            if(ucGlobalBuffer_1[const_array_size-1-k]>ucGlobalBuffer_1[const_array_size-1-1-k])  //后一个与前一个数据两两比较
                {
                    ucTemp=ucGlobalBuffer_1[const_array_size-1-1-k];     //通过一个中间变量实现两个数据交换
            ucGlobalBuffer_1[const_array_size-1-1-k]=ucGlobalBuffer_1[const_array_size-1-k];
            ucGlobalBuffer_1[const_array_size-1-k]=ucTemp;
                }

        }
   }
}
/* 注释三:
* 第2种方法，为了改进第1种方法的用户体验，用指针为函数增加一个输入接口。
* 为什么要用指针？因为C语言的函数中，数组不能直接用来做函数的形参，只能用指针作为数组的形参。
* 比如，你不能这样写一个函数void big_to_small_sort_2(unsigned char a[5])，否则编译就会出错不通过。
* 在本函数中，*p_ucInputBuffer指针就是输入接口，而输出接口仍然是全局变量数组ucGlobalBuffer_2。
* 这种方法由于为函数多增加了一个数组输入接口，已经比第1种方法更加直观了。
*/
void big_to_small_sort_2(unsigned char *p_ucInputBuffer)//第2种方法 把一个数组从大小小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
   for(i=0;i<const_array_size;i++)
   {
      ucGlobalBuffer_2[i]=p_ucInputBuffer[i];  //参与排序算法之前，先把输入接口的数据全部搬移到全局变量数组中。
   }
   //以下就是著名的 冒泡法排序。详细讲解请找百度。
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-i)
          {
            if(ucGlobalBuffer_2[const_array_size-1-k]>ucGlobalBuffer_2[const_array_size-1-1-k])  //后一个与前一个数据两两比较
                {
                    ucTemp=ucGlobalBuffer_2[const_array_size-1-1-k];     //通过一个中间变量实现两个数据交换
            ucGlobalBuffer_2[const_array_size-1-1-k]=ucGlobalBuffer_2[const_array_size-1-k];
            ucGlobalBuffer_2[const_array_size-1-k]=ucTemp;
                }

        }
   }
}
void usart_service(void)  //串口服务程序,在main函数里
{
    unsigned char i=0;
```

```
    if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
    {
        ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
        //下面的代码进入数据协议解析和数据处理的阶段
        uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
        while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
        {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x55)  //数据头eb 00 55的判断
            {
                        for(i=0;i<const_array_size;i++)
                        {
                ucUsartBuffer[i]=ucRcregBuf[uiRcMoveIndex+3+i]; //从串口接收到的需要被排序的原始数据
                        }
                //第1种运算方法，依靠全局变量
                        for(i=0;i<const_array_size;i++)
                        {
                           ucGlobalBuffer_1[i]=ucUsartBuffer[i];  //把需要被排列的数据放进输入全局变量数组
                        }
                big_to_small_sort_1(); //调用一次空函数就出结果了，结果还是保存在ucGlobalBuffer_1全局变量数组中
                for(i=0;i<const_array_size;i++)
                        {
                          eusart_send(ucGlobalBuffer_1[i]);  ////把用第1种方法排序后的结果返回给上位机观察
                        }
                        eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为第1种方法与第2种方法的分割线
                        eusart_send(0xee);
                        eusart_send(0xee);
                //第2种运算方法，依靠指针为函数增加一个数组的输入接口
                        //通过指针输入接口，直接把ucUsartBuffer数组的首地址传址进去，排序后输出的结果还是保存在ucGlobalBuffer_2全局变量数组中
                big_to_small_sort_2(ucUsartBuffer);
                for(i=0;i<const_array_size;i++)
                        {
                          eusart_send(ucGlobalBuffer_2[i]);  //把用第2种方法排序后的结果返回给上位机观察
                        }
                break;    //退出循环
            }
            uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
        }

        uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

    }
```

```
}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
```

```
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
    EA=1;      //开总中断
    ES=1;      //允许串口中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

第2种方法通过指针，为函数增加了一个数组输入接口，已经比第1种纯粹用全局变量的方法直观多了，但是还有一个小小的遗憾，因为它的输出排序结果仍然要靠全局变量。为了让函数更加完美，我们能不能为函数再增加一个输出接口？当然可以。欲知详情，请听下回分解-----指针的第三大好处，指针作为数组在函数中的输出接口。

第五十五节: 指针的第三大好处，指针作为数组在函数中的输出接口。

开场白:

上一节介绍的第2种方法，由于为函数多增加了一个数组输入接口，已经比第1种方法更加直观了，但是由于只有输入接口，没有输出接口，输出接口仍然要靠全局变量数组，所以还是有一个小小的遗憾，这节介绍的第3种方法就是为了改变这个遗憾，为数组在函数中多增加一个输出接口，这样，函数既有输入接口，又有输出接口，这样的函数才算完美直观。这一节要教大家一个知识点: 通过指针，为函数增加一个数组输出接口。

具体内容，请看源代码讲解。

(1) 硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:
把5个随机数据按从大到小排序，用冒泡法来排序。
通过电脑串口调试助手，往单片机发送EB 00 55 08 06 09 05 07 指令，其中EB 00 55是数据头，08 06 09 05 07 是参与排序的5个随机原始数据。单片机收到指令后就会返回13个数据，最前面5个数据是第2种方法的排序结果，中间3个数据EE EE EE是第2种和第3种的分割线，为了方便观察，没实际意义。最后5个数据是第3种方法的排序结果.
比如电脑发送: EB 00 55 08 06 09 05 07
单片机就返回: 09 08 07 06 05 EE EE EE 09 08 07 06 05
串口程序的接收部分请参考第39节。串口程序的发送部分请参考第42节。
波特率是: 9600 。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_array_size  5  //参与排序的数组大小
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void big_to_small_sort_2(unsigned char *p_ucInputBuffer);//第2种方法 把一个数组从大到小排序
void big_to_small_sort_3(unsigned char *p_ucInputBuffer,unsigned char *p_ucOutputBuffer);//第3种方法 把一个数组从大到小排序
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucUsartBuffer[const_array_size];  //从串口接收到的需要排序的原始数据
unsigned char ucGlobalBuffer_2[const_array_size]; //第2种方法，参与具体排序算法的全局变量数组
unsigned char ucGlobalBuffer_3[const_array_size]; //第3种方法，用来接收输出接口数据的全局变量数组
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释一:
* 第2种方法，为了改进第1种方法的用户体验，用指针为函数增加一个输入接口。
* 为什么要用指针？因为C语言的函数中，数组不能直接用来做函数的形参，只能用指针作为数组的形参。
* 比如，你不能这样写一个函数void big_to_small_sort_2(unsigned char a[5])，否则编译就会出错不通过。
```

```
* 在本函数中，*p_ucInputBuffer指针就是输入接口，而输出接口仍然是全局变量数组ucGlobalBuffer_2。
* 这种方法由于为函数多增加了一个数组输入接口，已经比第1种方法更加直观了，但是由于只有输入接口，
* 没有输出接口，输出接口仍然要靠全局变量，所以还是有点小遗憾，以下第3种方法就是为了改变这个遗憾。
*/
void big_to_small_sort_2(unsigned char *p_ucInputBuffer)//第2种方法 把一个数组从大到小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
   for(i=0;i<const_array_size;i++)
   {
      ucGlobalBuffer_2[i]=p_ucInputBuffer[i];  //参与排序算法之前，先把输入接口的数据全部搬移到全局变量
数组中。
   }
   //以下就是著名的 冒泡法排序。详细讲解请找百度。
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-
i)
          {
             if(ucGlobalBuffer_2[const_array_size-1-k]>ucGlobalBuffer_2[const_array_size-1-1-k])  //后一
个与前一个数据两两比较
                 {
                     ucTemp=ucGlobalBuffer_2[const_array_size-1-1-k];     //通过一个中间变量实现两个数据
交换
             ucGlobalBuffer_2[const_array_size-1-1-k]=ucGlobalBuffer_2[const_array_size-1-k];
             ucGlobalBuffer_2[const_array_size-1-k]=ucTemp;
                 }

          }
   }
}
/* 注释二:
* 第3种方法，为了改进第2种方法的用户体验，用指针为函数多增加一个数组输出接口。
* 这样，函数的数组既有输入接口，又有输出接口，已经堪称完美了。
* 本程序中*p_ucInputBuffer输入接口，*p_ucOutputBuffer是输出接口。
*/
void big_to_small_sort_3(unsigned char *p_ucInputBuffer,unsigned char *p_ucOutputBuffer)//第3种方法 把一
个数组从大到小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
   unsigned char ucBuffer_3[const_array_size]; //第3种方法，参与具体排序算法的局部变量数组
   for(i=0;i<const_array_size;i++)
   {
      ucBuffer_3[i]=p_ucInputBuffer[i];  //参与排序算法之前，先把输入接口的数据全部搬移到局部变量数组中
。
   }
   //以下就是著名的 冒泡法排序。详细讲解请找百度。
```

```
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-i)
          {
             if(ucBuffer_3[const_array_size-1-k]>ucBuffer_3[const_array_size-1-1-k])  //后一个与前一个数据两两比较
                 {
                     ucTemp=ucBuffer_3[const_array_size-1-1-k];      //通过一个中间变量实现两个数据交换
             ucBuffer_3[const_array_size-1-1-k]=ucBuffer_3[const_array_size-1-k];
             ucBuffer_3[const_array_size-1-k]=ucTemp;
                 }

          }
   }
   for(i=0;i<const_array_size;i++)
   {
      p_ucOutputBuffer[i]=ucBuffer_3[i];  //参与排序算法之后，把运算结果的数据全部搬移到输出接口中，方便外面程序调用
   }
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x55)  //数据头eb 00 55的判断
                {
                                    for(i=0;i<const_array_size;i++)
                                    {
                    ucUsartBuffer[i]=ucRcregBuf[uiRcMoveIndex+3+i]; //从串口接收到的需要被排序的原始数据
                                    }
                    //第2种运算方法，依靠指针为函数增加一个数组的输入接口
                                    //通过指针输入接口，直接把ucUsartBuffer数组的首地址传址进去，排序后输出的结果还是保存在ucGlobalBuffer_2全局变量数组中
                    big_to_small_sort_2(ucUsartBuffer);
                    for(i=0;i<const_array_size;i++)
                                    {
                                      eusart_send(ucGlobalBuffer_2[i]);  //把用第2种方法排序后的结果返回给上位机观察
                                    }
                                    eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为第2种方法与第3种方法的分割线
```

```
                                eusart_send(0xee);
                                eusart_send(0xee);
                    //第3种运算方法，依靠指针为函数增加一个数组的输出接口
                                //通过指针输出接口，排序运算后的结果直接从这个输出口中导出到
ucGlobalBuffer_3数组中
                    big_to_small_sort_3(ucUsartBuffer,ucGlobalBuffer_3);    //ucUsartBuffer是输入的数组
，ucGlobalBuffer_3是接收排序结果的数组
                    for(i=0;i<const_array_size;i++)
                                {
                                  eusart_send(ucGlobalBuffer_3[i]);  //把用第3种方法排序后的结果返回给
上位机观察
                                }
                    break;    //退出循环
                }
                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
            }

            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

        }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目
来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)    //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这
个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                //串口接收数据中断
{
   if(RI==1)
   {
```

```
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
```

```
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```
复制代码

总结陈词:

通过本节程序的讲解，一部分细心的读者可能会发现一个规律，其实所谓指针作为数组在函数中的输入接口和输出接口，输入接口的指针跟输出接口的指针在语法上没有任何区别，我没有用到C语言中专门的关键词去限定某个指针是输入，某个指针是输出，因此，这个告诉我们什么道理？指针在函数的接口中，天生就是既可以做输入，也可以是做输出，它是双向性的，不像普通的函数变量形参只能做输入。发现了这个秘密，我们可不可以把本节程序中的输入接口和输出接口合并成一个输入输出接口呢？当然可以。欲知详情，请听下回分解-----指针的第四大好处，指针作为数组在函数中的输入输出接口。

第五十六节：指针的第四大好处，指针作为数组在函数中的输入输出接口。

开场白:

通过前面几个章节的学习，我们知道指针在函数的接口中，天生就是既可以做输入，也可以是做输出，它是双向性的，类似全局变量的特点。我们根据实际项目的情况，在必要的时候可以直接把输入接口和输出接口合并在一起，这种方法的缺点是没有把输入和输出分开，没有那么直观。但是优点也是很明显的，就是比较省程序ROM容量和数据RAM容量，而且运行效率也比较快。这一节要教大家一个知识点：指针作为数组在函数中输入输出接口的特点。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

把5个随机数据按从大到小排序，用冒泡法来排序。

通过电脑串口调试助手，往单片机发送EB 00 55 08 06 09 05 07 指令，其中EB 00 55是数据头，08 06 09 05 07 是参与排序的5个随机原始数据。单片机收到指令后就会返回13个数据，最前面5个数据是第3种方法的排序结果，中间3个数据EE EE EE是第3种和第4种的分割线，为了方便观察，没实际意义。最后5个数据是第4种方法的排序结果.

比如电脑发送: EB 00 55 08 06 09 05 07

单片机就返回: 09 08 07 06 05 EE EE EE 09 08 07 06 05

串口程序的接收部分请参考第39节。串口程序的发送部分请参考第42节。

波特率是: 9600 。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_array_size  5  //参与排序的数组大小
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void big_to_small_sort_3(unsigned char *p_ucInputBuffer,unsigned char *p_ucOutputBuffer);//第3种方法 把一个数组从大到小排序
```

```
void big_to_small_sort_4(unsigned char *p_ucInputAndOutputBuffer);//第4种方法 把一个数组从大到小排序
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;      //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;     //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucUsartBuffer[const_array_size];  //从串口接收到的需要排序的原始数据
unsigned char ucGlobalBuffer_3[const_array_size]; //第3种方法，用来接收输出接口数据的全局变量数组
unsigned char ucGlobalBuffer_4[const_array_size]; //第4种方法，用来输入和输出接口数据的全局变量数组
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释一:
* 第3种方法，为了改进第2种方法的用户体验，用指针为函数多增加一个数组输出接口。
* 这样，函数的数组既有输入接口，又有输出接口，已经堪称完美了。
* 本程序中*p_ucInputBuffer输入接口，*p_ucOutputBuffer是输出接口。
*/
void big_to_small_sort_3(unsigned char *p_ucInputBuffer,unsigned char *p_ucOutputBuffer)//第3种方法 把一个数组从大到小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
   unsigned char ucBuffer_3[const_array_size]; //第3种方法，参与具体排序算法的局部变量数组
   for(i=0;i<const_array_size;i++)
   {
      ucBuffer_3[i]=p_ucInputBuffer[i];  //参与排序算法之前，先把输入接口的数据全部搬移到局部变量数组中。
   }
   //以下就是著名的 冒泡法排序。详细讲解请找百度。
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-i)
          {
             if(ucBuffer_3[const_array_size-1-k]>ucBuffer_3[const_array_size-1-1-k])  //后一个与前一个数据两两比较
                 {
                     ucTemp=ucBuffer_3[const_array_size-1-1-k];     //通过一个中间变量实现两个数据交换
             ucBuffer_3[const_array_size-1-1-k]=ucBuffer_3[const_array_size-1-k];
             ucBuffer_3[const_array_size-1-k]=ucTemp;
                 }
```

```
        }
    }
    for(i=0;i<const_array_size;i++)
    {
       p_ucOutputBuffer[i]=ucBuffer_3[i];  //参与排序算法之后，把运算结果的数据全部搬移到输出接口中，方便外面程序调用
    }
}
/* 注释二:
* 第4种方法.指针在函数的接口中，天生就是既可以做输入，也可以是做输出，它是双向性的，类似全局变量的特点。
* 我们可以根据实际项目的情况，在必要的时候可以直接把输入接口和输出接口合并在一起，
* 这种方法的缺点是没有把输入和输出分开，没有那么直观。但是优点也是很明显的，就是比较
* 省程序ROM容量和数据RAM容量，而且运行效率也比较快。现在介绍给大家。
* 本程序的*p_ucInputAndOutputBuffer是输入输出接口。
*/
void big_to_small_sort_4(unsigned char *p_ucInputAndOutputBuffer)//第4种方法 把一个数组从大到小排序
{
    unsigned char i;
    unsigned char k;
    unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
    //以下就是著名的 冒泡法排序。详细讲解请找百度。
    for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
    {
       for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-i)
          {
             if(p_ucInputAndOutputBuffer[const_array_size-1-k]>p_ucInputAndOutputBuffer[const_array_size-1-1-k])  //后一个与前一个数据两两比较
                 {
                     ucTemp=p_ucInputAndOutputBuffer[const_array_size-1-1-k];     //通过一个中间变量实现两个数据交换
             p_ucInputAndOutputBuffer[const_array_size-1-1-k]=p_ucInputAndOutputBuffer[const_array_size-1-k];
             p_ucInputAndOutputBuffer[const_array_size-1-k]=ucTemp;
                 }

          }
    }
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
            {
```

```
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x55)  //数据头eb 00 55的判断
                {
                                        for(i=0;i<const_array_size;i++)
                                        {
                    ucUsartBuffer[i]=ucRcregBuf[uiRcMoveIndex+3+i]; //从串口接收到的需要被排序的原始数据
                                        }
                  //第3种运算方法，依靠指针为函数增加一个数组的输出接口
                                        //通过指针输出接口，排序运算后的结果直接从这个输出口中导出到ucGlobalBuffer_3数组中
                  big_to_small_sort_3(ucUsartBuffer,ucGlobalBuffer_3);    //ucUsartBuffer是输入的数组，ucGlobalBuffer_3是接收排序结果的数组
                  for(i=0;i<const_array_size;i++)
                                        {
                                          eusart_send(ucGlobalBuffer_3[i]);  //把用第3种方法排序后的结果返回给上位机观察
                                        }
                                        eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为第2种方法与第3种方法的分割线
                                        eusart_send(0xee);
                                        eusart_send(0xee);
                  //第4种运算方法，依靠一个指针作为函数的输入输出接口。
                                        //通过这个指针输入输出接口，ucGlobalBuffer_4数组既是输入数组，也是输出数组，排序运算后的结果直接存放在它本身，类似于全局变量的特点。
                                        for(i=0;i<const_array_size;i++)
                                        {
                    ucGlobalBuffer_4[i]=ucUsartBuffer[i]; //把需要被排序的原始数据传递给接收输入输出数组ucGlobalBuffer_4,
                                        }
                  big_to_small_sort_4(ucGlobalBuffer_4);
                  for(i=0;i<const_array_size;i++)
                                        {
                                          eusart_send(ucGlobalBuffer_4[i]);  //把用第4种方法排序后的结果返回给上位机观察
                                        }
                  break;    //退出循环
                }
                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
          }

          uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

    }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
```

```
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;     //开自锁标志
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;   //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
            ; //一个分号相当于执行一条空语句
```

```
        }
    }
}
void delay_short(unsigned int uiDelayShort)
{
    unsigned int i;
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
    EA=1;      //开总中断
    ES=1;      //允许串口中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

通过本章的学习，我们知道指针在函数接口中的双向性，这个双向性是一把双刃剑，既给我们带来便捷，也给我们在以下两个场合中带来隐患。

第一个场合: 当需要把输入接口和输出接口分开时，我们希望输入接口的参数不要被意外改变，改变的仅仅只能是输出接口的数据。但是指针的双向性，就有可能导致我们在写函数内部代码的时候一不小心改变而没有发觉。

第二个场合: 如果是一个现成封装好的函数直接给我们调用，当我们发现是指针作为接口的时候，我们就不敢确定这个接口是输入接口，还是输出接口，或者是输入输出接口，我们传递进去的参数可能会更改，除非用之前进行数据备份，否则是没有安全感可言的。

有没有办法巧妙的解决以上两个问题？当然有。欲知详情，请听下回分解-----为指针加上紧箍咒const，避免意外修改了只做输入接口的数据。

（未完待续，下节更精彩，不要走开哦）

第五十七节: 为指针加上紧箍咒const，避免意外修改了只做输入接口的数据。

开场白:

通过上一节的学习，我们知道指针在函数接口中具有双向性，这个双向性是一把双刃剑，既给我们带来便捷，也给我们带来隐患。这一节要教大家以下知识点:

凡是做输入接口的指针，都应该加上const标签来标识,它可以让原来双向性的接口变成了单向性接口，它有两个好处:

第一个: 如果你是用别人已经封装好的函数，你发现接口指针带了const标签，就足以说明这个指针只能做输入接口，你用了它，不用担心输入数据被修改。

第二个：如果是你自己写的函数，你在输入接口处的指针加了const标签，它可以预防你在写函数内部代码时不小心修改了输入接口的数据。比如，你试着在函数内部更改带const标签的输入接口数据，当你点击编译时，会编译不过，出现错误提示：error C183: unmodifiable lvalue。这就是一道防火墙啊！
具体内容，请看源代码讲解。
（1）硬件平台：
基于朱兆祺51单片机学习板。
（2）实现功能：
我只是把第55节中凡是输入接口数据的指针都加了const关键字标签，其它代码内容没变。
把5个随机数据按从大到小排序，用冒泡法来排序。
通过电脑串口调试助手，往单片机发送EB 00 55 08 06 09 05 07 指令，其中EB 00 55是数据头，08 06 09 05 07 是参与排序的5个随机原始数据。单片机收到指令后就会返回13个数据，最前面5个数据是第3种方法的排序结果，中间3个数据EE EE EE是第3种和第4种的分割线，为了方便观察，没实际意义。最后5个数据是第4种方法的排序结果.
比如电脑发送：EB 00 55 08 06 09 05 07
单片机就返回：09 08 07 06 05 EE EE EE 09 08 07 06 05
波特率是：9600 。
（3）源代码讲解如下：

```
#include "REG52.H"
#define const_array_size  5  //参与排序的数组大小
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void big_to_small_sort_2(const unsigned char *p_ucInputBuffer);//第2种方法 把一个数组从大到小排序
void big_to_small_sort_3(const unsigned char *p_ucInputBuffer,unsigned char *p_ucOutputBuffer);//第3种方法 把一个数组从大到小排序
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucUsartBuffer[const_array_size];  //从串口接收到的需要排序的原始数据
unsigned char ucGlobalBuffer_2[const_array_size]; //第2种方法，参与具体排序算法的全局变量数组
unsigned char ucGlobalBuffer_3[const_array_size]; //第3种方法，用来接收输出接口数据的全局变量数组
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
```

```
   }
}
void big_to_small_sort_2(const unsigned char *p_ucInputBuffer)//第2种方法 把一个数组从大到小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
   for(i=0;i<const_array_size;i++)
   {
      ucGlobalBuffer_2[i]=p_ucInputBuffer[i];  //参与排序算法之前，先把输入接口的数据全部搬移到全局变量数组中。
   }
   //以下就是著名的 冒泡法排序。详细讲解请找百度。
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-i)
          {
             if(ucGlobalBuffer_2[const_array_size-1-k]>ucGlobalBuffer_2[const_array_size-1-1-k])  //后一个与前一个数据两两比较
                 {
                     ucTemp=ucGlobalBuffer_2[const_array_size-1-1-k];     //通过一个中间变量实现两个数据交换
             ucGlobalBuffer_2[const_array_size-1-1-k]=ucGlobalBuffer_2[const_array_size-1-k];
             ucGlobalBuffer_2[const_array_size-1-k]=ucTemp;
                 }

          }
   }
}
/* 注释一:
* 凡是做输入接口的指针，都应该加上const标签来标识,它可以让原来双向性的接口变成了单向性接口，它有两个好处:
* 第一个: 如果你是用别人已经封装好的函数，你发现接口指针带了const标签，就足以说明
* 这个指针只能做输入接口，你用了它，不用担心输入数据被修改。
* 第二个: 如果是你自己写的函数，你在输入接口处的指针加了const标签，它可以预防你在写函数内部代码时
* 不小心修改了输入接口的数据。比如，你试着在以下函数最后的地方加一条更改输入接口数据的指令，
* 当你点击编译时，会编译不过，出现错误提示: error C183: unmodifiable lvalue。
*/
void big_to_small_sort_3(const unsigned char *p_ucInputBuffer,unsigned char *p_ucOutputBuffer)//第3种方法 把一个数组从大到小排序
{
   unsigned char i;
   unsigned char k;
   unsigned char ucTemp; //在两两交换数据的过程中，用于临时存放交换的某个变量
   unsigned char ucBuffer_3[const_array_size]; //第3种方法，参与具体排序算法的局部变量数组
   for(i=0;i<const_array_size;i++)
   {
      ucBuffer_3[i]=p_ucInputBuffer[i];  //参与排序算法之前，先把输入接口的数据全部搬移到局部变量数组中。
```

```
   }
   //以下就是著名的 冒泡法排序。详细讲解请找百度。
   for(i=0;i<(const_array_size-1);i++)  //冒泡的次数是(const_array_size-1)次
   {
      for(k=0;k<(const_array_size-1-i);k++) //每次冒泡的过程中，需要两两比较的次数是(const_array_size-1-
i)
          {
             if(ucBuffer_3[const_array_size-1-k]>ucBuffer_3[const_array_size-1-1-k])  //后一个与前一个数
据两两比较
                 {
                     ucTemp=ucBuffer_3[const_array_size-1-1-k];      //通过一个中间变量实现两个数据交换
             ucBuffer_3[const_array_size-1-1-k]=ucBuffer_3[const_array_size-1-k];
             ucBuffer_3[const_array_size-1-k]=ucTemp;
                 }

          }
   }
   for(i=0;i<const_array_size;i++)
   {
      p_ucOutputBuffer[i]=ucBuffer_3[i];  //参与排序算法之后，把运算结果的数据全部搬移到输出接口中，方便
外面程序调用
   }
/* 注释二:
* 以下这条是企图修改输入接口数据的指令，如果不屏蔽，编译的时候就会出错提醒: error C183: unmodifiable
lvalue?
*/
   //p_ucInputBuffer[0]=0;  //修改输入接口数据的指令
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
               {
                                  for(i=0;i<const_array_size;i++)
                                  {
                    ucUsartBuffer[i]=ucRcregBuf[uiRcMoveIndex+3+i]; //从串口接收到的需要被排序的原始数
据
                                  }
                  //第2种运算方法，依靠指针为函数增加一个数组的输入接口
                                  //通过指针输入接口，直接把ucUsartBuffer数组的首地址传址进去，排序后输
出的结果还是保存在ucGlobalBuffer_2全局变量数组中
                  big_to_small_sort_2(ucUsartBuffer);
```

```
                for(i=0;i<const_array_size;i++)
                                {
                                  eusart_send(ucGlobalBuffer_2[i]);  //把用第2种方法排序后的结果返回给
上位机观察
                                }
                                eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为第
2种方法与第3种方法的分割线
                                eusart_send(0xee);
                                eusart_send(0xee);
                //第3种运算方法，依靠指针为函数增加一个数组的输出接口
                                //通过指针输出接口，排序运算后的结果直接从这个输出口中导出到
ucGlobalBuffer_3数组中
                big_to_small_sort_3(ucUsartBuffer,ucGlobalBuffer_3);   //ucUsartBuffer是输入的数组
，ucGlobalBuffer_3是接收排序结果的数组
                for(i=0;i<const_array_size;i++)
                                {
                                  eusart_send(ucGlobalBuffer_3[i]);  //把用第3种方法排序后的结果返回给
上位机观察
                                }
                break;   //退出循环
              }
              uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
           }

           uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

     }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目
来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这
个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
```

```
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                    //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
```

```
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

通过前面几节的学习，我们知道了指针在函数接口中的输入输出用途，以及const关键字的作用。下一节将要讲指针的第五大好处。欲知详情，请听下回分解-----指针的第五大好处，指针在众多数组中的中转站作用。

第五十八节: 指针的第五大好处，指针在众多数组中的中转站作用。

开场白:

单个变量数据之间可以通过一条指令任意自由赋值转移，但是数组之间不能通过一条指令直接赋值转移，必须用for等循环指令挨个把数组的数据一个一个来赋值转移，如果一个 函数中，有很多数组需要赋值转移，那就非常麻烦了，要用很多for语句，耗时。还好C语言里有个指针，它可以非常高效地来切换我们所需要的数组，起到很好的中转站作用。这一节要教大家一个知识点: 指针在众多数组中的中转站作用。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

在第57节的串口收发程序基础上修改。在串口接收函数中，以下代码有略微修改:

while(uiRcregTotal>=4&&uiRcMoveIndex<=(uiRcregTotal-4))//注意，这里是4，不是上一节的5，因为只有eb 00 55 xx这4个数据

通过上位机来调用下位机对应的数组数据。

通过电脑串口调试助手，往单片机发送EB 00 55 XX 指令，其中EB 00 55是数据头，XX的取值范围是0x01 至 0x05，每个不同的值代表调用下位机不同的数组数据。0x01调用第1组数据，0x02调用第2组数据，0x05调用第5组数据。

第1组: 11 12 13 14 15

第2组: 21 22 23 24 25

第3组: 31 32 33 34 35

第4组: 41 42 43 44 45

第5组: 51 52 53 54 55

下位机返回21个数据，前面5个是第1种不带指针函数返回的数据。中间5个是第2种不带指针函数返回的数据。最后5个是第3种带指针函数返回的数据。期间2组EE EE EE是各函数返回的数据分割线，为了方便观察，没实际意义。

比如电脑发送: EB 0055 02

单片机就返回: 21 2223 24 25 EE EE EE 21 22 23 24 25 EE EE EE 21 22 23 24 25

波特率是: 9600 。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_array_size  5  //参与排序的数组大小
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  10  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时
```

```
间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void send_array_1(unsigned char ucArraySec);  //第1种函数，不带指针
void send_array_2(unsigned char ucArraySec);  //第2种函数，不带指针
void send_array_3(unsigned char ucArraySec);  //第3种函数，带指针
void eusart_send(unsigned char ucSendData);
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
const unsigned char array_0x01[]={0x11,0x12,0x13,0x14,0x15}; //第1个常量数组
const unsigned char array_0x02[]={0x21,0x22,0x23,0x24,0x25}; //第2个常量数组
const unsigned char array_0x03[]={0x31,0x32,0x33,0x34,0x35}; //第3个常量数组
const unsigned char array_0x04[]={0x41,0x42,0x43,0x44,0x45}; //第4个常量数组
const unsigned char array_0x05[]={0x51,0x52,0x53,0x54,0x55}; //第5个常量数组
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释一:
* 第1种函数，内部不带指针，根据上位机相关的指令，
* 直接返回对应的数组。由于不带指针，因此多用了5个for循环来搬运数组。
* 比较耗程序ROM容量，也不够简洁清晰。
*/
void send_array_1(unsigned char ucArraySec)
{
   unsigned int i;
   switch(ucArraySec)
   {
      case 1:  //直接返回第1个常量数组
               for(i=0;i<5;i++)
                   {
                      eusart_send(array_0x01[i]);
                   }
               break;
      case 2:  //直接返回第2个常量数组
```

```
                for(i=0;i<5;i++)
                    {
                       eusart_send(array_0x02[i]);
                    }
                break;
        case 3:  //直接返回第3个常量数组
                for(i=0;i<5;i++)
                    {
                       eusart_send(array_0x03[i]);
                    }
                break;
        case 4:  //直接返回第4个常量数组
                for(i=0;i<5;i++)
                    {
                       eusart_send(array_0x04[i]);
                    }
                break;
        case 5:  //直接返回第5个常量数组
                for(i=0;i<5;i++)
                    {
                       eusart_send(array_0x05[i]);
                    }
                break;
    }

}
/* 注释二:
* 第2种函数，内部不带指针，根据上位机相关的指令，
* 先转移对应的数组放到一个中间变量数组，然后发送数组。
* 由于不带指针，因此多用了6个for循环来搬运数组。
* 跟第1种函数一样，比较耗程序ROM容量，也不够简洁清晰。
*/
void send_array_2(unsigned char ucArraySec)  //第2种函数，不带指针
{
    unsigned int i;
    unsigned char array_temp[5]; //临时中间数组
    switch(ucArraySec)
    {
        case 1:  //直接返回第1个常量数组
                for(i=0;i<5;i++)
                    {
                           array_temp[i]=array_0x01[i]; //先挨个把对应的数组数据转移到中间数组里
                    }
                break;
        case 2:  //直接返回第2个常量数组
                for(i=0;i<5;i++)
                    {
                           array_temp[i]=array_0x02[i]; //先挨个把对应的数组数据转移到中间数组里
                    }
                break;
```

```
        case 3:  //直接返回第3个常量数组
                 for(i=0;i<5;i++)
                     {
                            array_temp[i]=array_0x03[i]; //先挨个把对应的数组数据转移到中间数组里
                     }
                 break;
        case 4:  //直接返回第4个常量数组
                 for(i=0;i<5;i++)
                     {
                            array_temp[i]=array_0x04[i]; //先挨个把对应的数组数据转移到中间数组里
                     }
                 break;
        case 5:  //直接返回第5个常量数组
                 for(i=0;i<5;i++)
                     {
                            array_temp[i]=array_0x05[i]; //先挨个把对应的数组数据转移到中间数组里
                     }
                 break;
    }
    for(i=0;i<5;i++)
    {
            eusart_send(array_temp[i]);  //把临时存放在中间数组的数据全部发送出去
    }
}
/* 注释三:
* 第3种函数，内部带指针，根据上位机相关的指令，
* 先把对应的数组首地址传递给一个中间指针，然后再通过
* 指针把整个数组的数据发送出去，由于带指针，切换转移数组的数据非常快，
* 只需传递一下首地址给指针就可以，非常高效，整个函数只用了1个for循环。
* 跟前面第1,2种函数相比，更加节省程序容量，处理速度更加快，更加简洁。
*/
void send_array_3(unsigned char ucArraySec)  //第3种函数，带指针
{
    unsigned int i;
    unsigned char *p_array; //临时中间指针，作为数组的中转站，非常高效
    switch(ucArraySec)
    {
       case 1:  //直接返回第1个常量数组
                p_array=array_0x01;  //把数组的首地址传递给指针，一个指令就可以，不用for来挨个搬移数
据，高效!
                break;
       case 2:  //直接返回第2个常量数组
                p_array=array_0x02;  //把数组的首地址传递给指针，一个指令就可以，不用for来挨个搬移数
据，高效!
                break;
       case 3:  //直接返回第3个常量数组
                p_array=array_0x03;  //把数组的首地址传递给指针，一个指令就可以，不用for来挨个搬移数
据，高效!
                break;
       case 4:  //直接返回第4个常量数组
```

```
                    p_array=array_0x04;  //把数组的首地址传递给指针，一个指令就可以，不用for来挨个搬移数
据，高效!
                break;
      case 5:  //直接返回第5个常量数组
                    p_array=array_0x05;  //把数组的首地址传递给指针，一个指令就可以，不用for来挨个搬移数
据，高效!
                break;
   }
   for(i=0;i<5;i++)
   {
           eusart_send(p_array[i]);  //通过指针把数组的数据全部发送出去
   }
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     unsigned char ucWhichArray;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcregTotal>=4&&uiRcMoveIndex<=(uiRcregTotal-4)) //注意，这里是4，不是上一节的5，因为
只有eb 00 55 xx这4个数据
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
                {
                    ucWhichArray=ucRcregBuf[uiRcMoveIndex+3]; //上位机需要返回的某个数组
                    send_array_1(ucWhichArray); //第1种函数返回数组的5个数据，不带指针
                    eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为分割线
                    eusart_send(0xee);
                    eusart_send(0xee);
                    send_array_2(ucWhichArray); //第2种函数返回数组的5个数据，不带指针
                    eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为分割线
                    eusart_send(0xee);
                    eusart_send(0xee);
                    send_array_3(ucWhichArray); //第3种函数返回数组的5个数据，带指针
                    break;   //退出循环
                }
                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
            }

            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

     }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
```

```
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
      uiSendCnt++;    //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;     //开自锁标志
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
        ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;   //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
```

```
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

通过前面几节的学习，基本上讲完了我平时用指针的所有心得体会。

下一节开始讲新内容。在前面一些章节中，我提到为了防止中断函数把某些共享数据破坏，在主函数中更改某个数据变量时，应该先关闭中断，修改完后再打开中断；我也提到了网友“红金龙吸味”关于原子锁的建议。经过这段时间的思考和总结，我发现不管是关中断开中断，还是原子锁，其实本质上都是程序在多进程中临界点的数据处理，原子锁在程序员中有个专用名词叫互斥量，而我引以为豪的状态机程序框架，主函数的switch语句，外加一个定时中断，本质上就是2个独立进程在不断切换并行运行。我觉得这个临界点处理的知识很重要，也很容易忽略，所以我决定专门用两节内容来讲讲这方面的知识应用。欲知详情，请听下回分解-----关中断和开中断在多进程临界点的应用。

第五十九节: 串口程序第40,44,45节中存在一个bug，特此紧急公告。

经过网友“intech2008”的提醒，在我之前发表的第40,44,45节串口接收程序中，在计算检验和的地方，存在一个不容易发觉的bug。

原来的是:

```
  for(i=0;i<(3+1+2+uiRcSize);i++) //计算校验累加和
  {
      ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=ucRcregBuf[uiRcMoveIndex+6+uiRcSize]+ucRcregBuf[i];
```

```
  }
```
应该改成:
```
  for(i=0;i<(3+1+2+uiRcSize);i++) //计算校验累加和
  {
ucRcregBuf[uiRcMoveIndex+6+uiRcSize]=ucRcregBuf[uiRcMoveIndex+6+uiRcSize]+ucRcregBuf[uiRcMoveIndex+i];
  }
```
由于本连载技术文章在各大论坛发布和被转载，我没法做到处处提醒，不得不专门用一节内容来告知各位读者。

下节预告-----关中断和开中断在多进程临界点的应用。

（未完待续，下节更精彩，不要走开哦）

第六十节：用关中断和互斥量来保护多线程共享的全局变量。

开场白:

在前面一些章节中，我提到为了防止中断函数把某些共享数据破坏，在主函数中更改某个数据变量时，应该先关闭中断，修改完后再打开中断；我也提到了网友“红金龙吸味”关于原子锁的建议。经过这段时间的思考和总结，我发现不管是关中断开中断，还是原子锁，其实本质上都是程序在多进程中临界点的数据处理，原子锁有个专用名词叫互斥量，而我引以为豪的状态机程序框架，主函数的switch语句，外加一个定时中断，本质上就是2个独立进程在不断切换并行运行。

为什么要保护多线程共享的全局变量？因为，多个线程同时访问同一个全局变量，如果都是读取操作，则不会出现问题。如果一个线程负责改变此变量的值，而其他线程负责同时读取变量内容，则不能保证读取到的数据是经过写线程修改后的。

这一节要教大家一个知识点：如何用关中断和互斥量来保护多线程共享的全局变量。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

在第5节的基础上略作修改，让蜂鸣器在前面3秒发生一次短叫报警，在后面6秒发生一次长叫报警，如此反复循环。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_time_3s 1332   //3秒钟的时间需要的定时中断次数
#define const_time_6s 2664   //6秒钟的时间需要的定时中断次数
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_voice_long   200  //蜂鸣器长叫的持续时间
void initial_myself();
void initial_peripheral();
void delay_long(unsigned int uiDelaylong);
void led_flicker();
void alarm_run();
void T0_time();  //定时中断函数
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned char ucAlarmStep=0; //报警的步骤变量
unsigned int  uiTimeAlarmCnt=0; //报警统计定时中断次数的延时计数器
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucLock=0;      //互斥量，俗称原子锁
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      alarm_run();   //报警器定时报警
```

```
   }
}
/* 注释一:
* 保护多线程共享全局变量的原理:
* 多个线程同时访问同一个全局变量，如果都是读取操作，则不会出现问题。如果一个线程负责改变此变量的值，
* 而其他线程负责同时读取变量内容，则不能保证读取到的数据是经过写线程修改后的。
* 鸿哥的基本程序框架都是两线程为主，一个是main函数线程，一个是定时函数线程。
*/
void alarm_run() //报警器的应用程序
{

  switch(ucAlarmStep)
  {
     case 0:
           if(uiTimeAlarmCnt>=const_time_3s) //时间到
           {
/* 注释二:
* 用关中断来保护多线程共享的全局变量:
* 因为uiTimeAlarmCnt和uiVoiceCnt都是unsigned int类型，本质上是由两个字节组成。
* 在C语言中uiTimeAlarmCnt=0和uiVoiceCnt=const_voice_short看似一条指令，
* 实际上经过编译之后它不只一条汇编指令。由于另外一个定时中断线程里也会对这个变量
* 进行判断和操作，如果不禁止定时中断或者采取其它措施，定时函数往往会在主函数还没有
* 结束操作共享变量前就去访问或处理这个共享变量，这就会引起冲突，导致系统运行异常。
*/
              ET0=0;  //禁止定时中断
              uiTimeAlarmCnt=0; //时间计数器清零
              uiVoiceCnt=const_voice_short;  //蜂鸣器短叫
                         ET0=1; //开启允许定时中断
              ucAlarmStep=1; //切换到下一个步骤
           }
           break;
     case 1:
           if(uiTimeAlarmCnt>=const_time_6s) //时间到
           {
/* 注释三:
* 用互斥量来保护多线程共享的全局变量:
* 我觉得，在这种场合，用互斥量比前面用关中断的方法更加好。
* 因为一旦关闭了定时中断，整个中断函数就会在那一刻停止运行了，
* 而加一个互斥量，既能保护全局变量，又能让定时中断函数正常运行，
* 真是一举两得。
*/
                    ucLock=1;  //互斥量加锁。 俗称原子锁
              uiTimeAlarmCnt=0; //时间计数器清零
              uiVoiceCnt=const_voice_long;  //蜂鸣器长叫
                    ucLock=0; //互斥量解锁。  俗称原子锁
              ucAlarmStep=0; //返回到上一个步骤
           }
           break;
  }
}
```

```
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断

  if(ucLock==0) //互斥量判断
  {
     if(uiTimeAlarmCnt<0xffff)  //设定这个条件，防止uiTimeAlarmCnt超范围。
     {
         uiTimeAlarmCnt++;  //报警的时间计数器，累加定时中断的次数，
     }
     if(uiVoiceCnt!=0)
     {
         uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
     }
     else
     {
         ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
         beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
     }
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
}
void initial_peripheral() //第二区 初始化外围
{
  EA=1;     //开总中断
  ET0=1;    //允许定时中断
  TR0=1;    //启动定时中断
```

}
复制代码
总结陈词:
从下一节开始我准备用几章节的内容来讲常用的数学运算程序。这些程序经常要用在计算器，工控，以及高精度的仪器仪表等领域。C语言的语法中不是已经提供了+,-,*,/这些运算符号吗？为什么还要专门写算法程序？因为那些运算符只能进行简单的运算，一旦数据超过了unsigned long（4个字节）的范围就会出错。而这种大数据算法的程序是什么样的？欲知详情，请听下回分解----大数据的加法运算。
（未完待续，下节更精彩，不要走开哦）

第六十一节：组合BCD码，非组合BCD码，以及数值三者之间的相互转换和关系。

开场白:
本来这一节打算讲大数据的加法运算的，但是考虑大数据运算的基础是非组合BCD码，所以多增加一节讲BCD码的内容。
计算机中的BCD码，经常使用的有两种格式，即组合BCD码，非组合BCD码。
组合BCD码，是将两位十进制数，存放在一个字节中，例如：十进制数51的存放格式是0101 0001。
非组合BCD码，是将一个字节的低四位编码表示十进制数的一位，而高4位都为0。例如：十进制数51的占用了两个字节的空间，存放格式为：00000101 00000001。
这一节要教大家两个知识点:
第一个：如何编写组合BCD码，非组合BCD码，以及数值三者之间的相互转换函数。
第二个：通过转换函数的编写，重温前面几节所讲到的指针用法。
具体内容，请看源代码讲解。
（1）硬件平台:
基于朱兆祺51单片机学习板。
（2）实现功能:
波特率是：9600 。
通过电脑串口调试助手模拟上位机，往单片机发送EB 00 55 XX YY YY … YY YY 指令，其中EB 00 55是数据头，XX是指令类型。YY是具体的数据。
指令类型01代表发送的是数值，需要转成组合BCD码和非组合BCD码，并且返回上位机显示。
指令类型02代表发送的是组合BCD码，需要转成数值和非组合BCD码，并且返回上位机显示。
指令类型03代表发送的是非组合BCD码，需要转成数值和组合BCD码，并且返回上位机显示。
返回上位机的数据中，中间3个数据EE EE EE是分割线，为了方便观察，没实际意义。
例如：十进制的数据52013140，它的十六进制数据是03 19 A8 54。
（a）上位机发送数据：eb 00 55 01 03 19 a8 54
单片机返回：52 01 31 40 EE EE EE 05 02 00 01 03 01 04 00
（b）上位机发送组合BCD码：eb 00 55 02 52 01 31 40
单片机返回：03 19 A8 54 EE EE EE 05 02 00 01 03 01 04 00
（c）发送非组合BCD码：eb 00 55 03 05 02 00 01 03 01 04 00
单片机返回：03 19 A8 54 EE EE EE 52 01 31 40
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
/* 注释一:
* 注意，此处的const_rc_size是20，比之前章节的缓冲区稍微改大了一点。
*/
#define const_rc_size  20  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
```

```
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void number_to_BCD4(const unsigned char *p_ucNumber,unsigned char *p_ucBCD_bit4);//把数值转换成组合BCD码
void number_to_BCD8(const unsigned char *p_ucNumber,unsigned char *p_ucBCD_bit8);//把数值转换成非组合
BCD码
void BCD4_to_number(const unsigned char *p_ucBCD_bit4,unsigned char *p_ucNumber); //组合BCD码转成数值
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char *p_ucBCD_bit8); //组合BCD码转成非组合
BCD码
void BCD8_to_number(const unsigned char *p_ucBCD_bit8,unsigned char *p_ucNumber); //非组合BCD码转成数值
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char *p_ucBCD_bit4); //非组合BCD码转成组合
BCD码
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
/* 注释二:
* 注意，本程序规定数值的最大范围是0至99999999
* 数组中的数据。高位在数组下标大的方向，低位在数组下标小的方向。
*/
unsigned char ucBufferNumber[4]; //数值，用4个字节表示long类型的数值
unsigned char ucBufferBCB_bit4[4]; //组合BCD码
unsigned char ucBufferBCB_bit8[8]; //非组合BCD码
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
void number_to_BCD4(const unsigned char *p_ucNumber,unsigned char *p_ucBCD_bit4)//把数值转换成组合BCD码
{
   unsigned long ulNumberTemp=0;
   unsigned char ucTemp=0;
   ulNumberTemp=p_ucNumber[3];  //把4个字节的数值合并成一个long类型数据
   ulNumberTemp=ulNumberTemp<<8;
   ulNumberTemp=ulNumberTemp+p_ucNumber[2];
   ulNumberTemp=ulNumberTemp<<8;
   ulNumberTemp=ulNumberTemp+p_ucNumber[1];
   ulNumberTemp=ulNumberTemp<<8;
   ulNumberTemp=ulNumberTemp+p_ucNumber[0];
   p_ucBCD_bit4[3]=ulNumberTemp%100000000/10000000;
   p_ucBCD_bit4[3]=p_ucBCD_bit4[3]<<4; //前半4位存第8位组合BCD码
   ucTemp=ulNumberTemp%10000000/1000000;
   p_ucBCD_bit4[3]=p_ucBCD_bit4[3]+ucTemp; //后半4位存第7位组合BCD码
```

```
    p_ucBCD_bit4[2]=ulNumberTemp%1000000/100000;
    p_ucBCD_bit4[2]=p_ucBCD_bit4[2]<<4; //前半4位存第6位组合BCD码
    ucTemp=ulNumberTemp%100000/10000;
    p_ucBCD_bit4[2]=p_ucBCD_bit4[2]+ucTemp;//后半4位存第5位组合BCD码
    p_ucBCD_bit4[1]=ulNumberTemp%10000/1000;
    p_ucBCD_bit4[1]=p_ucBCD_bit4[1]<<4; //前半4位存第4位组合BCD码
    ucTemp=ulNumberTemp%1000/100;
    p_ucBCD_bit4[1]=p_ucBCD_bit4[1]+ucTemp;//后半4位存第3位组合BCD码
    p_ucBCD_bit4[0]=ulNumberTemp%100/10;
    p_ucBCD_bit4[0]=p_ucBCD_bit4[0]<<4; //前半4位存第2位组合BCD码
    ucTemp=ulNumberTemp%10;
    p_ucBCD_bit4[0]=p_ucBCD_bit4[0]+ucTemp;//后半4位存第1位组合BCD码
}
void number_to_BCD8(const unsigned char *p_ucNumber,unsigned char *p_ucBCD_bit8)//把数值转换成非组合
BCD码
{
    unsigned long ulNumberTemp=0;
    ulNumberTemp=p_ucNumber[3];  //把4个字节的数值合并成一个long类型数据
    ulNumberTemp=ulNumberTemp<<8;
    ulNumberTemp=ulNumberTemp+p_ucNumber[2];
    ulNumberTemp=ulNumberTemp<<8;
    ulNumberTemp=ulNumberTemp+p_ucNumber[1];
    ulNumberTemp=ulNumberTemp<<8;
    ulNumberTemp=ulNumberTemp+p_ucNumber[0];
    p_ucBCD_bit8[7]=ulNumberTemp%100000000/10000000;//一个字节8位存储第8位非组合BCD码
    p_ucBCD_bit8[6]=ulNumberTemp%10000000/1000000;//一个字节8位存储第7位非组合BCD码
    p_ucBCD_bit8[5]=ulNumberTemp%1000000/100000;//一个字节8位存储第6位非组合BCD码
    p_ucBCD_bit8[4]=ulNumberTemp%100000/10000;//一个字节8位存储第5位非组合BCD码
    p_ucBCD_bit8[3]=ulNumberTemp%10000/1000;//一个字节8位存储第4位非组合BCD码
    p_ucBCD_bit8[2]=ulNumberTemp%1000/100;//一个字节8位存储第3位非组合BCD码
    p_ucBCD_bit8[1]=ulNumberTemp%100/10;//一个字节8位存储第2位非组合BCD码
    p_ucBCD_bit8[0]=ulNumberTemp%10;//一个字节8位存储第1位非组合BCD码
}
void BCD4_to_number(const unsigned char *p_ucBCD_bit4,unsigned char *p_ucNumber) //组合BCD码转成数值
{
    unsigned long ulTmep;
    unsigned long ulSum;
    ulSum=0;  //累加和数值清零
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[3];
    ulTmep=ulTmep>>4;  //把组合BCD码第8位分解出来
    ulTmep=ulTmep*10000000;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[3];
    ulTmep=ulTmep&0x0000000f;  //把组合BCD码第7位分解出来
    ulTmep=ulTmep*1000000;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[2];
```

```
    ulTmep=ulTmep>>4;  //把组合BCD码第6位分解出来
    ulTmep=ulTmep*100000;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[2];
    ulTmep=ulTmep&0x0000000f;  //把组合BCD码第5位分解出来
    ulTmep=ulTmep*10000;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[1];
    ulTmep=ulTmep>>4;  //把组合BCD码第4位分解出来
    ulTmep=ulTmep*1000;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[1];
    ulTmep=ulTmep&0x0000000f;  //把组合BCD码第3位分解出来
    ulTmep=ulTmep*100;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[0];
    ulTmep=ulTmep>>4;  //把组合BCD码第2位分解出来
    ulTmep=ulTmep*10;
    ulSum=ulSum+ulTmep; //累加各位数值
    ulTmep=0;
    ulTmep=p_ucBCD_bit4[0];
    ulTmep=ulTmep&0x0000000f;  //把组合BCD码第1位分解出来
    ulTmep=ulTmep*1;
    ulSum=ulSum+ulTmep; //累加各位数值
    //以上代码非常有规律，有兴趣的读者也可以自己想办法把它压缩成一个for循环的函数，可以极大节省容量。
    p_ucNumber[3]=ulSum>>24;  //把long类型数据分解成4个字节
    p_ucNumber[2]=ulSum>>16;
    p_ucNumber[1]=ulSum>>8;
    p_ucNumber[0]=ulSum;
}
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char *p_ucBCD_bit8) //组合BCD码转成非组合
BCD码
{
    unsigned char ucTmep;
    ucTmep=p_ucBCD_bit4[3];
    p_ucBCD_bit8[7]=ucTmep>>4;    //把组合BCD码第8位分解出来
    p_ucBCD_bit8[6]=ucTmep&0x0f;  //把组合BCD码第7位分解出来
    ucTmep=p_ucBCD_bit4[2];
    p_ucBCD_bit8[5]=ucTmep>>4;    //把组合BCD码第6位分解出来
    p_ucBCD_bit8[4]=ucTmep&0x0f;  //把组合BCD码第5位分解出来
    ucTmep=p_ucBCD_bit4[1];
    p_ucBCD_bit8[3]=ucTmep>>4;    //把组合BCD码第4位分解出来
    p_ucBCD_bit8[2]=ucTmep&0x0f;  //把组合BCD码第3位分解出来
    ucTmep=p_ucBCD_bit4[0];
    p_ucBCD_bit8[1]=ucTmep>>4;    //把组合BCD码第2位分解出来
    p_ucBCD_bit8[0]=ucTmep&0x0f;  //把组合BCD码第1位分解出来
```

```
}
void BCD8_to_number(const unsigned char *p_ucBCD_bit8,unsigned char *p_ucNumber) //非组合BCD码转成数值
{
   unsigned long ulTmep;
   unsigned long ulSum;
   ulSum=0;  //累加和数值清零
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[7];
   ulTmep=ulTmep*10000000;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[6];
   ulTmep=ulTmep*1000000;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[5];
   ulTmep=ulTmep*100000;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[4];
   ulTmep=ulTmep*10000;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[3];
   ulTmep=ulTmep*1000;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[2];
   ulTmep=ulTmep*100;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[1];
   ulTmep=ulTmep*10;
   ulSum=ulSum+ulTmep; //累加各位数值
   ulTmep=0;
   ulTmep=p_ucBCD_bit8[0];
   ulTmep=ulTmep*1;
   ulSum=ulSum+ulTmep; //累加各位数值
   //以上代码非常有规律，有兴趣的读者也可以自己想办法把它压缩成一个for循环的函数，可以极大节省容量。
   p_ucNumber[3]=ulSum>>24;  //把long类型数据分解成4个字节
   p_ucNumber[2]=ulSum>>16;
   p_ucNumber[1]=ulSum>>8;
   p_ucNumber[0]=ulSum;
}
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char *p_ucBCD_bit4) //非组合BCD码转成组合
BCD码
{
   unsigned char ucTmep;
   ucTmep=p_ucBCD_bit8[7];    //把非组合BCD码第8位分解出来
   p_ucBCD_bit4[3]=ucTmep<<4;
```

```
    p_ucBCD_bit4[3]=p_ucBCD_bit4[3]+p_ucBCD_bit8[6];     //把非组合BCD码第7位分解出来
    ucTmep=p_ucBCD_bit8[5];     //把非组合BCD码第6位分解出来
    p_ucBCD_bit4[2]=ucTmep<<4;
    p_ucBCD_bit4[2]=p_ucBCD_bit4[2]+p_ucBCD_bit8[4];     //把非组合BCD码第5位分解出来
    ucTmep=p_ucBCD_bit8[3];     //把非组合BCD码第4位分解出来
    p_ucBCD_bit4[1]=ucTmep<<4;
    p_ucBCD_bit4[1]=p_ucBCD_bit4[1]+p_ucBCD_bit8[2];     //把非组合BCD码第3位分解出来
    ucTmep=p_ucBCD_bit8[1];     //把非组合BCD码第2位分解出来
    p_ucBCD_bit4[0]=ucTmep<<4;
    p_ucBCD_bit4[0]=p_ucBCD_bit4[0]+p_ucBCD_bit8[0];     //把非组合BCD码第1位分解出来

}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcregTotal>=5&&uiRcMoveIndex<=(uiRcregTotal-5))
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
                {
                     switch(ucRcregBuf[uiRcMoveIndex+3])  //根据命令类型来进行不同的处理
                                    {
                                       case 1:  //接收到的是数值，需要转成组合BCD码和非组合BCD码
                                for(i=0;i<4;i++)
                                {
                                    ucBufferNumber[3-i]=ucRcregBuf[uiRcMoveIndex+4+i]; //从串口接收到的数据
，注意，高位在数组下标大的方向
                                }
                                number_to_BCD4(ucBufferNumber,ucBufferBCB_bit4);//把数值转换成组合BCD码
                                number_to_BCD8(ucBufferNumber,ucBufferBCB_bit8);//把数值转换成非组合BCD码
                                for(i=0;i<4;i++)
                                {
                                   eusart_send(ucBufferBCB_bit4[3-i]);  ////把组合BCD码返回给上位机观察，注
意，高位在数组下标大的方向
                                }
                                eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为分割线
                                eusart_send(0xee);
                                eusart_send(0xee);
                                for(i=0;i<8;i++)
                                {
                                   eusart_send(ucBufferBCB_bit8[7-i]);  ////把非组合BCD码返回给上位机观察
，注意，高位在数组下标大的方向
                                }
                                        break;
                                    case 2:  //接收到的是组合BCD码，需要转成数值和非组合BCD码
```

```
                        for(i=0;i<4;i++)
                        {
                            ucBufferBCB_bit4[3-i]=ucRcregBuf[uiRcMoveIndex+4+i]; //从串口接收到的组合BCD码，注意，高位在数组下标大的方向
                        }
                        BCD4_to_number(ucBufferBCB_bit4,ucBufferNumber); //组合BCD码转成数值
                        BCD4_to_BCD8(ucBufferBCB_bit4,ucBufferBCB_bit8); //组合BCD码转成非组合BCD码
                        for(i=0;i<4;i++)
                        {
                           eusart_send(ucBufferNumber[3-i]);  ////把数值返回给上位机观察，注意，高位在数组下标大的方向
                        }
                        eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为分割线
                        eusart_send(0xee);
                        eusart_send(0xee);
                        for(i=0;i<8;i++)
                        {
                           eusart_send(ucBufferBCB_bit8[7-i]);  ////把非组合BCD码返回给上位机观察，注意，高位在数组下标大的方向
                        }
                                    break;
                                case 3:  //接收到的是非组合BCD码，需要转成数值和组合BCD码
                        for(i=0;i<8;i++)
                        {
                            ucBufferBCB_bit8[7-i]=ucRcregBuf[uiRcMoveIndex+4+i]; //从串口接收到的非组合BCD码，注意，高位在数组下标大的方向
                        }
                        BCD8_to_number(ucBufferBCB_bit8,ucBufferNumber); //非组合BCD码转成数值
                        BCD8_to_BCD4(ucBufferBCB_bit8,ucBufferBCB_bit4); //非组合BCD码转成组合BCD码
                        for(i=0;i<4;i++)
                        {
                           eusart_send(ucBufferNumber[3-i]);  ////把数值返回给上位机观察
                        }
                        eusart_send(0xee);  //为了方便上位机观察，多发送3个字节ee ee ee作为分割线，注意，高位在数组下标大的方向
                        eusart_send(0xee);
                        eusart_send(0xee);
                        for(i=0;i<4;i++)
                        {
                           eusart_send(ucBufferBCB_bit4[3-i]);  ////把组合BCD码返回给上位机观察，注意，高位在数组下标大的方向
                        }
                                    break;
                                }
                    break;    //退出循环
                }
                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
            }

            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据
```

```
    }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                  //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
```

```
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

有了这一节非组合BCD的基础知识，下一节就开始讲大数据的算法程序。这些算法程序经常要用在计算器，工控，以及高精度的仪器仪表等领域。C语言的语法中不是已经提供了+,-,*,/这些运算符号吗？为什么还要专门写算法程序？因为那些运算符只能进行简单的运算，一旦数据超过了unsigned long（4个字节）的范围就会出错。而这种大数据算法的程序是什么样的？欲知详情，请听下回分解----大数据的加法运算。

（未完待续，下节更精彩，不要走开哦）

第六十二节：大数据的加法运算。

开场白:

直接用C语言的“+”运算符进行加法运算时，“被加数”，“加数”，“和”，这三个数据的最大范围是unsigned long 类型，也就是数据最大范围是4个字节，十进制的范围是0至4294967295。一旦超过了这个范围，则运算会出错。

因此，当进行大数据加法运算时，我们要额外编程序，实现大数据的算法。其实这种算法并不难，就是我们在小学里学的四则运算算法。

我们先要弄清楚一个新的概念。不考虑小数点的情况下，数据有两种表现形式。一种是常用的变量形式，另外一种是上一节讲到的BCD码数组形式。变量的最大范围有限，而BCD码数组的形式是无限的，正因为这个特点，所以我们可以进行大数据运算。

这一节要教大家两个知识点:

第一个: 如何通过用for循环语句改写上一节的组合BCD码跟非组合BCD码的转换函数。

第二个: 如何编写涉及到大数据加法运算的算法程序函数，同时也复习了指针的用途。

第三个: 如何在串口程序中通过关键字来截取所需要的数据。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是: 9600 。

通过电脑串口调试助手模拟上位机，往单片机发送组合BCD码的被加数和加数。单片机把组合BCD码的运算结果返回到上位机。最大范围4位，从0到9999，如果超范围则返回EE EE EE报错。往单片机发送的数据格式: EB 00 55 XX XX 0d 0a YY YY 0d 0a指令，其中EB 00 55是数据头，XX 是被加数，可以是1个字节，也可以是2个字节。YY是加数，可以是1个字节，也可以是2个字节。0d 0a是固定的结束标志。

例如:

（a）1234+5678=6912

上位机发送数据: eb 00 55 12 34 0d 0a 56 78 0d 0a

单片机返回: 69 12

（b）9999+56=10055 超过4位的9999，所以报错

上位机发送数据: eb 00 55 99 99 0d 0a 56 0d 0a

单片机返回: EE EE EE 表示出错了

（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 本系统中，规定最大运算位数是4位。
* 由于STC89C52单片机的RAM只有256个，也就是说系统的变量数最大
* 不能超过256个，如果超过了这个极限，编译器就会报错。如果这个算法
* 移植到stm32或者PIC等RAM比较大的单片机上，那么就可以把这个运算位数
* 设置得更加大一点。
*/
#define  BCD4_MAX     2  //本系统中，规定的组合BCD码最大字节数，一个字节包含2位，因此4位有效运算数
#define  BCD8_MAX    (BCD4_MAX*2)  //本系统中，规定的非组合BCD码最大字节数，一个字节包含1位，因此4位有效运算数
#define const_rc_size  30  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define uchar unsigned char    //方便移植平台
#define ulong unsigned long   //方便移植平台
//如果在VC的平台模拟此算法，则都定义成int类型，如下:
//#define uchar int
//#define ulong int
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
```

```
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt);
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt);
void ClearAllData(uchar ucARRAY_MAX,uchar *destData);
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX);
uchar AddData(const uchar *destData,const uchar *sourceData,uchar *resultData);
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;      //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;     //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucDataBCD4_1[BCD4_MAX]; //接收到的第1个数组合BCD码数组形式  这里是指被加数
unsigned char ucDataBCD4_cnt_1=0;  //接收到的第1个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_2[BCD4_MAX]; //接收到的第2个数组合BCD码数组形式  这里是指加数
unsigned char ucDataBCD4_cnt_2=0;  //接收到的第2个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_3[BCD4_MAX]; //接收到的第3个数组合BCD码数组形式  这里是指和
unsigned char ucDataBCD4_cnt_3=0;  //接收到的第3个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_1[BCD8_MAX]; //接收到的第1个数非组合BCD码数组形式   这里是指被加数
unsigned char ucDataBCD8_cnt_1=0;  //接收到的第1个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_2[BCD8_MAX]; //接收到的第2个数非组合BCD码数组形式   这里是指加数
unsigned char ucDataBCD8_cnt_2=0;  //接收到的第2个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_3[BCD8_MAX]; //接收到的第3个数非组合BCD码数组形式   这里是指和
unsigned char ucDataBCD8_cnt_3=0;  //接收到的第3个数非组合BCD码数组的有效数据长度
unsigned char ucResultFlag=11; //运算结果标志，10代表计算结果超出范围出错，11代表正常。
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释二:
* 组合BCD码转成非组合BCD码。
* 这里的变量ucBCD4_cnt代表组合BCD码的有效字节数.
* 这里的变量*p_ucBCD8_cnt代表经过转换后，非组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt)
{
```

```
    unsigned char ucTmep;
    unsigned char i;
    for(i=0;i<BCD8_MAX;i++)    //先把即将保存转换结果的缓冲区清零
    {
        p_ucBCD_bit8[i]=0;
    }
    *p_ucBCD8_cnt=ucBCD4_cnt*2; //转换成非组合BCD码后的有效数据长度
    for(i=0;i<ucBCD4_cnt;i++)
    {
        ucTmep=p_ucBCD_bit4[ucBCD4_cnt-1-i];
        p_ucBCD_bit8[ucBCD4_cnt*2-i*2-1]=ucTmep>>4;
        p_ucBCD_bit8[ucBCD4_cnt*2-i*2-2]=ucTmep&0x0f;
    }
}
/* 注释三:
* 非组合BCD码转成组合BCD码。
* 这里的变量ucBCD8_cnt代表非组合BCD码的有效字节数.
* 这里的变量*p_ucBCD4_cnt代表经过转换后，组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了非组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt)
{
    unsigned char ucTmep;
    unsigned char i;
    unsigned char ucBCD4_cnt;
    for(i=0;i<BCD4_MAX;i++)    //先把即将保存转换结果的缓冲区清零
    {
        p_ucBCD_bit4[i]=0;
    }
    ucBCD4_cnt=(ucBCD8_cnt+1)/2; //非组合BCD码转化成组合BCD码的有效数,这里+1避免非组合数据长度是奇数位
    *p_ucBCD4_cnt=ucBCD4_cnt; //把转换后的结果付给接口指针的数据，可以对外输出结果
    for(i=0;i<ucBCD4_cnt;i++)
    {
        ucTmep=p_ucBCD_bit8[ucBCD4_cnt*2-1-i*2];     //把非组合BCD码第8位分解出来
        p_ucBCD_bit4[ucBCD4_cnt-1-i]=ucTmep<<4;
        p_ucBCD_bit4[ucBCD4_cnt-1-i]=p_ucBCD_bit4[ucBCD4_cnt-1-i]+p_ucBCD_bit8[ucBCD4_cnt*2-2-i*2];
//把非组合BCD码第7位分解出来
    }

}
/* 注释四:
*函数介绍: 清零数组的全部数组数据
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*输入输出参数: *destData--被清零的数组。
*/
void ClearAllData(uchar ucARRAY_MAX,uchar *destData)
{
```

```
  uchar i;
  for(i=0;i<ucARRAY_MAX;i++)
  {
     destData[i]=0;
  }
}
/* 注释五:
*函数介绍: 获取数组的有效长度
*输入参数: *destData--被获取的数组。
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*返回值  : 返回数组的有效长度。比如58786这个数据的有效长度是5
*电子开发者作者: 吴坚鸿
*/
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX)
{
  uchar i;
  uchar DataLength=ucARRAY_MAX;
  for(i=0;i<ucARRAY_MAX;i++)
  {
      if(0!=destData[ucARRAY_MAX-1-i])
          {
             break;
          }
          else
          {
             DataLength--;
          }
  }
  return DataLength;
}
/* 注释六:
*函数介绍: 两个数相加
*输入参数:
*(1)*destData--被加数的数组。
*(2)*sourceData--加数的数组。
*(3)*resultData--和的数组。注意，调用本函数前，必须先把这个数组清零
*返回值  : 10代表计算结果超出范围出错，11代表正常。
*/
uchar AddData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar addResult=11; //开始默认返回的运算结果是正常
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
uchar carryData=0;  //进位
uchar maxCnt=0; //最大位数
uchar resultTemp=0; //存放临时运算结果的中间变量
//为什么不在本函数内先把resultData数组清零？因为后面章节中的乘法运算中要用到此函数实现连加功能。
//因此如果纯粹实现加法运算时，在调用本函数之前，必须先在外面把和的数组清零，否则会计算出错。
destCnt=GetDataLength(destData,BCD8_MAX);   //获取被加数的有效位数
```

```
sourceCnt=GetDataLength(sourceData,BCD8_MAX);  //获取加数的有效位数
if(destCnt>=sourceCnt)  //找出两个运算数据中最大的有效位数
{
   maxCnt=destCnt;
}
else
{
   maxCnt=sourceCnt;
}
for(i=0;i<maxCnt;i++)
{
   resultTemp=destData[i]+sourceData[i]+carryData; //按位相加
   resultData[i]=resultTemp%10;   //截取最低位存放进保存结果的数组
   carryData=resultTemp/10;     //存放进位
}
resultData[i]=carryData;
if((maxCnt==BCD8_MAX)&&(carryData==1))  //如果数组的有效位是最大值并且最后的进位是1，则计算溢出报错
{
  ClearAllData(BCD8_MAX,resultData);
  addResult=10;  //报错
}
return addResult;
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     unsigned char k=0;
         unsigned char ucGetDataStep=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcMoveIndex<uiRcregTotal)  //说明还没有把缓冲区的数据读取完
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
               {

                                    i=0;
                                    ucGetDataStep=0;
                   ucDataBCD4_cnt_1=0;  //第1个数组合BCD码数组的有效数据长度
                   ucDataBCD4_cnt_2=0;  //第2个数组合BCD码数组的有效数据长度
                                    ClearAllData(BCD4_MAX,ucDataBCD4_1);  //清零第1个参与运算的数据
                                    ClearAllData(BCD4_MAX,ucDataBCD4_2);  //清零第2个参与运算的数据
                   //以下while循环是通过关键字0x0d 0x0a来截取第1个和第2个参与运算的数据。
                                    while(i<(BCD8_MAX+4))//这里+4是因为有2对0x0d 0x0a结尾特殊符号，一个共
4个字节
                                    {
                                           if(ucGetDataStep==0)//步骤0，相当于我平时用的case 0,获取第1个
```

```
数，在这里是指被加数
                                        {
if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a) //结束标志
                                                {
                                for(k=0;k<ucDataBCD4_cnt_1;k++) //提取第1个参与运算的数组数据
                                                        {
ucDataBCD4_1[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的数组数据与实际存储的数组数据的下标方向是相反的
                                                        }

                                                        i=i+2; //跳过 0x0d 0x0a 这两个字节，进行下一轮的关键字提取
                                                   ucGetDataStep=1;  //切换到下一个关键字提取的步骤
                                }
                                                else
                                                {
                                                        i++;
                                                        ucDataBCD4_cnt_1++;  //统计第1个有效数据的长度
                                }

                                        }
                                    else if(ucGetDataStep==1) //步骤1，相当于我平时用的case 1,获取第2个参与运行的数，在这里是加数
                                        {
if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a) //结束标志
                                                {
                                for(k=0;k<ucDataBCD4_cnt_2;k++) //提取第2个参与运算的数组数据
                                                        {
ucDataBCD4_2[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的数组数据与实际存储的数组数据的下标方向是相反的
                                                        }

                                                break; //截取数据完成。直接跳出截取数据的while(i<(BCD8_MAX+4))循环
                                }
                                                else
                                                {
                                                        i++;
                                                        ucDataBCD4_cnt_2++;  //统计第2个有效数据的长度
                                }
                            }
                        }
                        //注意ucDataBCD8_cnt_1和ucDataBCD8_cnt_2要带地址符号&传址进去
                        BCD4_to_BCD8(ucDataBCD4_1,ucDataBCD4_cnt_1,ucDataBCD8_1,&ucDataBCD8_cnt_1); //把接收到的组合BCD码转换成非组合BCD码  第1个数
                        BCD4_to_BCD8(ucDataBCD4_2,ucDataBCD4_cnt_2,ucDataBCD8_2,&ucDataBCD8_cnt_2); //把接收到的组合BCD码转换成非组合BCD码  第2个数
                                ClearAllData(BCD8_MAX,ucDataBCD8_3);  //清零第3个参与运算的数据,用来
```

```
接收运行的结果
                                        ucResultFlag=AddData(ucDataBCD8_1,ucDataBCD8_2,ucDataBCD8_3);
//相加运算，结果放在ucDataBCD8_3数组里
                                        if(ucResultFlag==11) //表示运算结果没有超范围
                                        {
                            ucDataBCD8_cnt_3=GetDataLength(ucDataBCD8_3,BCD8_MAX);  //获取和的有效字节数
BCD8_to_BCD4(ucDataBCD8_3,ucDataBCD8_cnt_3,ucDataBCD4_3,&ucDataBCD4_cnt_3); //把非组合BCD码转成组合BCD码
。注意，&ucDataBCD4_cnt_3带地址符号&
                            for(k=0;k<ucDataBCD4_cnt_3;k++) //返回运算结果到上位机上观察。看到的是组合BCD码形
式。返回的时候注意数组下标的顺序要反过来发送，先发高位的下标数组
                                            {
                                                eusart_send(ucDataBCD4_3[ucDataBCD4_cnt_3-1-k]); //往上
位机发送一个字节的函数
                                            }
                                        }
                                        else //运算结果超范围，返回EE EE EE
                                        {
                                             eusart_send(0xee); //往上位机发送一个字节的函数
                                             eusart_send(0xee); //往上位机发送一个字节的函数
                                             eusart_send(0xee); //往上位机发送一个字节的函数
                                        }
                        break;    //退出循环
                    }
                    uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
                }
                ucRcregBuf[0]=0; //把数据头清零，方便下次接收判断新数据
                        ucRcregBuf[1]=0;
                        ucRcregBuf[2]=0;

                uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

        }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目
来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)    //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
```

```
        uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                    //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
```

```
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;     //开总中断
   ES=1;     //允许串口中断
   ET0=1;    //允许定时中断
   TR0=1;    //启动定时中断
}
```

复制代码

总结陈词:

既然这节讲了加法程序，那么下一节接着讲常用的减法程序，这种大数据的减法程序是什么样的？欲知详情，请听下回分解----大数据的减法运算。

（未完待续，下节更精彩，不要走开哦）

第六十三节：大数据的减法运算。

开场白:

直接用C语言的“-”运算符进行加法运算时，“被减数”，“ 减数”，“差”，这三个数据的最大范围是unsigned long 类型，也就是数据最大范围是4个字节，十进制的范围是0至4294967295。一旦超过了这个范围，则运算会出错。因此，当进行大数据减法运算时，我们要额外编程序，实现大数据的算法。其实这种算法并不难，就是我们在小学里学的四则运算算法。

我们先要弄清楚一个新的概念。不考虑小数点的情况下，数据有两种表现形式。一种是常用的变量形式，另外一种是BCD码数组形式。变量的最大范围有限，而BCD码数组的形式是无限的，正因为这个特点，所以我们可以进行大数据运算。

这一节要教大家两个知识点:

第一个：如何编写比较两个非组合BCD码数据的大小。

第二个：如何编写涉及到大数据减法运算的算法程序函数，同时也复习了指针的用途。

具体内容，请看源代码讲解。

第六十三节：大数据的减法运算。

开场白:

直接用C语言的“-”运算符进行加法运算时，“被减数”，“ 减数”，“差”，这三个数据的最大范围是unsigned long 类型，也就是数据最大范围是4个字节，十进制的范围是0至4294967295。一旦超过了这个范围，则运算会出错。因此，当进行大数据减法运算时，我们要额外编程序，实现大数据的算法。其实这种算法并不难，就是我们在小学里学的四则运算算法。

我们先要弄清楚一个新的概念。不考虑小数点的情况下，数据有两种表现形式。一种是常用的变量形式，另外一种是BCD码数组形式。变量的最大范围有限，而BCD码数组的形式是无限的，正因为这个特点，所以我们可以进行大数据运算。

这一节要教大家两个知识点:

第一个: 如何编写比较两个非组合BCD码数据的大小。

第二个: 如何编写涉及到大数据减法运算的算法程序函数，同时也复习了指针的用途。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是: 9600 。

通过电脑串口调试助手模拟上位机，往单片机发送组合BCD码的被减数和减数。单片机把组合BCD码的运算结果返回到上位机。最大范围4位，从0到9999，如果被减数小于减数则返回EE EE EE报错。往单片机发送的数据格式: EB 00 55 XX XX 0d 0a YY YY 0d 0a指令，其中EB 00 55是数据头，XX 是被减数，可以是1个字节，也可以是2个字节。YY是减数，可以是1个字节，也可以是2个字节。0d 0a是固定的结束标志。

例如:

（a）8259 5267 = 2992

上位机发送数据: eb 00 55 82 59 0d 0a 52 67 0d 0a

单片机返回: 29 92

（b）5267 - 8259=小于0 所以报错

上位机发送数据: eb 00 55 52 67 0d 0a 82 59 0d 0a

单片机返回: EE EE EE 表示出错了

（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 本系统中，规定最大运算位数是4位。
* 由于STC89C52单片机的RAM只有256个，也就是说系统的变量数最大
* 不能超过256个，如果超过了这个极限，编译器就会报错。如果这个算法
* 移植到stm32或者PIC等RAM比较大的单片机上，那么就可以把这个运算位数
* 设置得更加大一点。
*/
#define BCD4_MAX 2 //本系统中，规定的组合BCD码最大字节数，一个字节包含2位，因此4位有效运算数
#define BCD8_MAX (BCD4_MAX*2) //本系统中，规定的非组合BCD码最大字节数，一个字节包含1位，因此4位有效运算数
#define const_rc_size 30 //接收串口中断数据的缓冲区数组大小
#define const_receive_time 5 //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define uchar unsigned char //方便移植平台
#define ulong unsigned long //方便移植平台
//如果在VC的平台模拟此算法，则都定义成int类型，如下:
//#define uchar int
//#define ulong int
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
```

```
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt);
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt);
void ClearAllData(uchar ucARRAY_MAX,uchar *destData);
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX);
uchar CmpData(const uchar *destData,const uchar *sourceData); //比较两个数的大小
uchar SubData(const uchar *destData,const uchar *sourceData,uchar *resultData);//两个数相减
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucDataBCD4_1[BCD4_MAX]; //接收到的第1个数组合BCD码数组形式  这里是指被减数
unsigned char ucDataBCD4_cnt_1=0;  //接收到的第1个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_2[BCD4_MAX]; //接收到的第2个数组合BCD码数组形式  这里是指减数
unsigned char ucDataBCD4_cnt_2=0;  //接收到的第2个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_3[BCD4_MAX]; //接收到的第3个数组合BCD码数组形式  这里是指差
unsigned char ucDataBCD4_cnt_3=0;  //接收到的第3个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_1[BCD8_MAX]; //接收到的第1个数非组合BCD码数组形式   这里是指被减数
unsigned char ucDataBCD8_cnt_1=0;  //接收到的第1个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_2[BCD8_MAX]; //接收到的第2个数非组合BCD码数组形式   这里是指减数
unsigned char ucDataBCD8_cnt_2=0;  //接收到的第2个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_3[BCD8_MAX]; //接收到的第3个数非组合BCD码数组形式   这里是指差
unsigned char ucDataBCD8_cnt_3=0;  //接收到的第3个数非组合BCD码数组的有效数据长度
unsigned char ucResultFlag=11; //运算结果标志，10代表计算结果超出范围出错，11代表正常。
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释二:
* 组合BCD码转成非组合BCD码。
* 这里的变量ucBCD4_cnt代表组合BCD码的有效字节数.
* 这里的变量*p_ucBCD8_cnt代表经过转换后，非组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
```

```
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt)
{
   unsigned char ucTmep;
   unsigned char i;
   for(i=0;i<BCD8_MAX;i++)    //先把即将保存转换结果的缓冲区清零
   {
      p_ucBCD_bit8[i]=0;
   }
   *p_ucBCD8_cnt=ucBCD4_cnt*2; //转换成非组合BCD码后的有效数据长度
   for(i=0;i<ucBCD4_cnt;i++)
   {
      ucTmep=p_ucBCD_bit4[ucBCD4_cnt-1-i];
      p_ucBCD_bit8[ucBCD4_cnt*2-i*2-1]=ucTmep>>4;
      p_ucBCD_bit8[ucBCD4_cnt*2-i*2-2]=ucTmep&0x0f;
   }
}
/* 注释三:
* 非组合BCD码转成组合BCD码。
* 这里的变量ucBCD8_cnt代表非组合BCD码的有效字节数.
* 这里的变量*p_ucBCD4_cnt代表经过转换后，组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了非组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt)
{
   unsigned char ucTmep;
   unsigned char i;
   unsigned char ucBCD4_cnt;
   for(i=0;i<BCD4_MAX;i++)    //先把即将保存转换结果的缓冲区清零
   {
      p_ucBCD_bit4[i]=0;
   }
   ucBCD4_cnt=(ucBCD8_cnt+1)/2; //非组合BCD码转化成组合BCD码的有效数,这里+1避免非组合数据长度是奇数位
   *p_ucBCD4_cnt=ucBCD4_cnt; //把转换后的结果付给接口指针的数据，可以对外输出结果
   for(i=0;i<ucBCD4_cnt;i++)
   {
      ucTmep=p_ucBCD_bit8[ucBCD4_cnt*2-1-i*2];     //把非组合BCD码第8位分解出来
      p_ucBCD_bit4[ucBCD4_cnt-1-i]=ucTmep<<4;
      p_ucBCD_bit4[ucBCD4_cnt-1-i]=p_ucBCD_bit4[ucBCD4_cnt-1-i]+p_ucBCD_bit8[ucBCD4_cnt*2-2-i*2];
//把非组合BCD码第7位分解出来
   }

}
/* 注释四:
*函数介绍: 清零数组的全部数组数据
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*输入输出参数: *destData--被清零的数组。
*/
```

```
void ClearAllData(uchar ucARRAY_MAX,uchar *destData)
{
  uchar i;
  for(i=0;i<ucARRAY_MAX;i++)
  {
     destData[i]=0;
  }
}
/* 注释五:
*函数介绍: 获取数组的有效长度
*输入参数: *destData--被获取的数组。
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*返回值  : 返回数组的有效长度。比如58786这个数据的有效长度是5
*电子开发者作者: 吴坚鸿
*/
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX)
{
  uchar i;
  uchar DataLength=ucARRAY_MAX;
  for(i=0;i<ucARRAY_MAX;i++)
  {
      if(0!=destData[ucARRAY_MAX-1-i])
          {
             break;
          }
          else
          {
             DataLength--;
          }
  }
  return DataLength;
}
/* 注释六:
*函数介绍: 比较两个数的大小
*输入参数:
*(1)*destData--被比较数的数组。
*(2)*sourceData--比较数的数组。
*返回值  : 9代表小于，10代表相等，11代表大于。
*/
uchar CmpData(const uchar *destData,const uchar *sourceData)
{
uchar cmpResult=10; //开始默认相等
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
destCnt=GetDataLength(destData,BCD8_MAX);
sourceCnt=GetDataLength(sourceData,BCD8_MAX);
if(destCnt>sourceCnt)  //大于
{
  cmpResult=11;
```

```
}
else if(destCnt<sourceCnt) //小于
{
  cmpResult=9;
}
else if((destCnt==0)&&(sourceCnt==0))  //如果都是等于0则等于
{
  cmpResult=10;
}
else  //否则就要继续判断
{
  for(i=0;i<destCnt;i++)
  {
     if(destData[destCnt-1-i]>sourceData[destCnt-1-i])   //从最高位开始判断,如果最高位大于则大于
         {
           cmpResult=11;
           break;
         }
     else if(destData[destCnt-1-i]<sourceData[destCnt-1-i])  //从最高位开始判断,如果最高位小于则小于
         {
           cmpResult=9;
           break;
         }
     //否则继续判断下一位
  }
}
return cmpResult;
}
/* 注释七:
*函数介绍: 两个数相减
*输入参数:
*(1)*destData--被减数的数组。
*(2)*sourceData--减数的数组。
*(3)*resultData--差的数组。注意，调用本函数前，必须先把这个数组清零
*返回值  : 10代表计算结果是负数或者超出范围出错，11代表正常。
*/
uchar SubData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar subResult=11; //开始默认正常
uchar destCnt=0;
uchar i;
uchar carryData=0;  //进位
uchar maxCnt=0; //最大位数
uchar resultTemp=0; //存放临时运算结果的中间变量
//为什么不在本函数内先把resultData数组清零？因为后面章节中的除法运算中要用到此函数实现连减功能。
//因此如果纯粹实现减法运算时，在调用本函数之前，必须先在外面把差的数组清零，否则会计算出错。
if(CmpData(destData,sourceData)==9)  //被减数小于减数，报错
{
   subResult=10;
   return subResult;  //返回判断结果，并且退出本程序，不往下执行本程序余下代码
```

```
}
destCnt=GetDataLength(destData,BCD8_MAX);  //获取被减数的有效数据长度
maxCnt=destCnt;
for(i=0;i<maxCnt;i++)
{
   resultTemp=sourceData[i]+carryData; //按位相加
   if(resultTemp>destData[i])
   {
      resultData[i]=destData[i]+10-sourceData[i]-carryData;    //借位
          carryData=1;
   }
   else
   {
      resultData[i]=destData[i]-sourceData[i]-carryData;    //不用借位
          carryData=0;
   }
}
return subResult;
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     unsigned char k=0;
         unsigned char ucGetDataStep=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
           ucSendLock=0;    //处理一次就锁起来，不用每次都进来,除非有新接收的数据
           //下面的代码进入数据协议解析和数据处理的阶段
           uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
           while(uiRcMoveIndex<uiRcregTotal)  //说明还没有把缓冲区的数据读取完
           {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
              {

                  i=0;
                  ucGetDataStep=0;
                  ucDataBCD4_cnt_1=0;  //第1个数组合BCD码数组的有效数据长度
                  ucDataBCD4_cnt_2=0;  //第2个数组合BCD码数组的有效数据长度
                  ClearAllData(BCD4_MAX,ucDataBCD4_1);  //清零第1个参与运算的数据
                  ClearAllData(BCD4_MAX,ucDataBCD4_2);  //清零第2个参与运算的数据
                  //以下while循环是通过关键字0x0d 0x0a来截取第1个和第2个参与运算的数据。
                  while(i<(BCD8_MAX+4))//这里+4是因为有2对0x0d 0x0a结尾特殊符号，一个共4个字节
                  {
                     if(ucGetDataStep==0)//步骤0，相当于我平时用的case 0,获取第1个数，在这里是指被加数
                     {
                          if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a)
//结束标志
                          {
                               for(k=0;k<ucDataBCD4_cnt_1;k++) //提取第1个参与运算的数组数据
```

```
                    {
                        ucDataBCD4_1[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的数组数据与实际存储的数组数据的下标方向是相反的
                    }

                    i=i+2; //跳过 0x0d 0x0a 这两个字节，进行下一轮的关键字提取
                    ucGetDataStep=1;  //切换到下一个关键字提取的步骤
                }
                else
                {
                    i++;
                    ucDataBCD4_cnt_1++;  //统计第1个有效数据的长度
                }

            }
            else if(ucGetDataStep==1) //步骤1，相当于我平时用的case 1,获取第2个参与运行的数，在这里是加数
            {
                if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a) //结束标志
                {
                    for(k=0;k<ucDataBCD4_cnt_2;k++) //提取第2个参与运算的数组数据
                    {
                        ucDataBCD4_2[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的数组数据与实际存储的数组数据的下标方向是相反的
                    }

                    break; //截取数据完成。直接跳出截取数据的while(i<(BCD8_MAX+4))循环
                }
                else
                {
                    i++;
                    ucDataBCD4_cnt_2++;  //统计第2个有效数据的长度
                }
            }
        }
        //注意ucDataBCD8_cnt_1和ucDataBCD8_cnt_2要带地址符号&传址进去
        BCD4_to_BCD8(ucDataBCD4_1,ucDataBCD4_cnt_1,ucDataBCD8_1,&ucDataBCD8_cnt_1); //把接收到的组合BCD码转换成非组合BCD码  第1个数
        BCD4_to_BCD8(ucDataBCD4_2,ucDataBCD4_cnt_2,ucDataBCD8_2,&ucDataBCD8_cnt_2); //把接收到的组合BCD码转换成非组合BCD码  第2个数
        ClearAllData(BCD8_MAX,ucDataBCD8_3);  //清零第3个参与运算的数据,用来接收运行的结果
        ucResultFlag=SubData(ucDataBCD8_1,ucDataBCD8_2,ucDataBCD8_3); //相减运算，结果放在ucDataBCD8_3数组里
        if(ucResultFlag==11) //表示运算结果没有超范围
        {
            ucDataBCD8_cnt_3=GetDataLength(ucDataBCD8_3,BCD8_MAX);  //获取运算结果的有效字节数
            BCD8_to_BCD4(ucDataBCD8_3,ucDataBCD8_cnt_3,ucDataBCD4_3,&ucDataBCD4_cnt_3); //把非组合BCD码转成组合BCD码。注意，&ucDataBCD4_cnt_3带地址符号&
```

```
                        for(k=0;k<ucDataBCD4_cnt_3;k++) //返回运算结果到上位机上观察。看到的是组合BCD码形式。返回的时候注意数组下标的顺序要反过来发送，先发高位的下标数组
                        {
                           eusart_send(ucDataBCD4_3[ucDataBCD4_cnt_3-1-k]); //往上位机发送一个字节的函数
                        }
                     }
                     else //运算结果超范围，返回EE EE EE
                     {
                        eusart_send(0xee); //往上位机发送一个字节的函数
                        eusart_send(0xee); //往上位机发送一个字节的函数
                        eusart_send(0xee); //往上位机发送一个字节的函数
                     }
                     break;    //退出循环
                  }
                  uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
               }
               ucRcregBuf[0]=0; //把数据头清零，方便下次接收判断新数据
               ucRcregBuf[1]=0;
               ucRcregBuf[2]=0;

               uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

         }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                //串口接收数据中断
```

```
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
```

```
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

既然这节讲了减法程序，那么下一节接着讲常用的乘法程序，这种大数据的乘法程序是什么样的？欲知详情，请听下回分解----大数据的乘法运算。

（未完待续，下节更精彩，不要走开哦）

第六十四节：大数据的乘法运算。

开场白:

直接用C语言的“*”运算符进行乘法运算时，“被乘数”，“ 乘数”，“积”，这三个数据的最大范围是unsigned long 类型，也就是数据最大范围是4个字节，十进制的范围是0至4294967295。一旦超过了这个范围，则运算会出错。因此，当进行大数据乘法运算时，我们要额外编程序，实现大数据的算法。其实这种算法并不难，就是我们在小学里学的四则运算算法。

我们先要弄清楚一个新的概念。不考虑小数点的情况下，数据有两种表现形式。一种是常用的变量形式，另外一种是BCD码数组形式。变量的最大范围有限，而BCD码数组的形式是无限的，正因为这个特点，所以我们可以进行大数据运算。

这一节要教大家一个知识点:

第一个: 如何编写涉及到大数据乘法运算的算法程序函数，同时也复习了指针的用途。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是: 9600 。

通过电脑串口调试助手模拟上位机，往单片机发送组合BCD码的被乘数和乘数，单片机把组合BCD码的运算结果返回到上位机。被乘数与乘数的最大范围都是从0到99，如果运算的乘积超过允许保存的最大位数范围则返回EE EE EE报错。

往单片机发送的数据格式: EB 00 55 XX 0d 0a YY 0d 0a指令，其中EB 00 55是数据头，XX 是被乘数，是1个字节的组合BCD码。YY是乘数，可以是1个字节的组合BCD码。0d 0a是固定的结束标志。

例如:

（a）83 x 98 = 8134

上位机发送数据: eb 00 55 83 0d 0a 98 0d 0a

单片机返回: 81 34

（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 本系统中的乘法运算，规定两个乘数的最大范围是0至99.
* 由于STC89C52单片机的RAM只有256个，也就是说系统的变量数最大
* 不能超过256个，如果超过了这个极限，编译器就会报错。由于51单片机RAM资源有限，
* 因此规定乘数的最大范围不能超过99，如果这个算法移植到stm32或者PIC等RAM比较大
* 的单片机上，那么就可以把这个运算位数设置得更加大一点。调整下面 BCD4_MAX的大小，
* 可以调整运算的数据范围。
*/
#define  BCD4_MAX     3  //为了让乘法的结果不超过范围，因此把组合BCD码最大字节数从上一节的2改成3，一个字
```

```
节包含2位，因此可以保存6位有效数
#define  BCD8_MAX    (BCD4_MAX*2)  //本系统中，规定的非组合BCD码能保存的最大字节数，一个字节包含1位，因此能保存6位有效运算数
#define const_rc_size  30  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define uchar unsigned char    //方便移植平台
#define ulong unsigned long   //方便移植平台
//如果在VC的平台模拟此算法，则都定义成int类型，如下:
//#define uchar int
//#define ulong int
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt);
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt);
void ClearAllData(uchar ucARRAY_MAX,uchar *destData);
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX);
uchar AddData(const uchar *destData,const uchar *sourceData,uchar *resultData);  //两个数相加
void EnlargeData(uchar *destData,uchar enlarge_cnt); //数组向大索引值移位，移一位相当于放大10倍
uchar MultData(const uchar *destData,const uchar *sourceData,uchar *resultData); //两个数相乘
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucDataBCD4_1[BCD4_MAX]; //接收到的第1个数组合BCD码数组形式  这里是指被乘数
unsigned char ucDataBCD4_cnt_1=0;  //接收到的第1个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_2[BCD4_MAX]; //接收到的第2个数组合BCD码数组形式  这里是指乘数
unsigned char ucDataBCD4_cnt_2=0;  //接收到的第2个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_3[BCD4_MAX]; //接收到的第3个数组合BCD码数组形式  这里是指积
unsigned char ucDataBCD4_cnt_3=0;  //接收到的第3个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_1[BCD8_MAX]; //接收到的第1个数非组合BCD码数组形式   这里是指被乘数
unsigned char ucDataBCD8_cnt_1=0;  //接收到的第1个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_2[BCD8_MAX]; //接收到的第2个数非组合BCD码数组形式   这里是指乘数
unsigned char ucDataBCD8_cnt_2=0;  //接收到的第2个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_3[BCD8_MAX]; //接收到的第3个数非组合BCD码数组形式   这里是指积
unsigned char ucDataBCD8_cnt_3=0;  //接收到的第3个数非组合BCD码数组的有效数据长度
unsigned char ucResultFlag=11; //运算结果标志，10代表计算结果超出范围出错，11代表正常。
void main()
  {
   initial_myself();
```

```
    delay_long(100);
    initial_peripheral();
    while(1)
    {
        usart_service();  //串口服务程序
    }
}
/* 注释二:
* 组合BCD码转成非组合BCD码。
* 这里的变量ucBCD4_cnt代表组合BCD码的有效字节数.
* 这里的变量*p_ucBCD8_cnt代表经过转换后，非组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt)
{
    unsigned char ucTmep;
    unsigned char i;
    for(i=0;i<BCD8_MAX;i++)    //先把即将保存转换结果的缓冲区清零
    {
        p_ucBCD_bit8[i]=0;
    }
    *p_ucBCD8_cnt=ucBCD4_cnt*2; //转换成非组合BCD码后的有效数据长度
    for(i=0;i<ucBCD4_cnt;i++)
    {
        ucTmep=p_ucBCD_bit4[ucBCD4_cnt-1-i];
        p_ucBCD_bit8[ucBCD4_cnt*2-i*2-1]=ucTmep>>4;
        p_ucBCD_bit8[ucBCD4_cnt*2-i*2-2]=ucTmep&0x0f;
    }
}
/* 注释三:
* 非组合BCD码转成组合BCD码。
* 这里的变量ucBCD8_cnt代表非组合BCD码的有效字节数.
* 这里的变量*p_ucBCD4_cnt代表经过转换后，组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了非组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt)
{
    unsigned char ucTmep;
    unsigned char i;
    unsigned char ucBCD4_cnt;
    for(i=0;i<BCD4_MAX;i++)    //先把即将保存转换结果的缓冲区清零
    {
        p_ucBCD_bit4[i]=0;
    }
```

```
   ucBCD4_cnt=(ucBCD8_cnt+1)/2; //非组合BCD码转化成组合BCD码的有效数,这里+1避免非组合数据长度是奇数位
   *p_ucBCD4_cnt=ucBCD4_cnt; //把转换后的结果付给接口指针的数据，可以对外输出结果
   for(i=0;i<ucBCD4_cnt;i++)
   {
      ucTmep=p_ucBCD_bit8[ucBCD4_cnt*2-1-i*2];     //把非组合BCD码第8位分解出来
      p_ucBCD_bit4[ucBCD4_cnt-1-i]=ucTmep<<4;
      p_ucBCD_bit4[ucBCD4_cnt-1-i]=p_ucBCD_bit4[ucBCD4_cnt-1-i]+p_ucBCD_bit8[ucBCD4_cnt*2-2-i*2];
//把非组合BCD码第7位分解出来
   }

}
/* 注释四:
*函数介绍: 清零数组的全部数组数据
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*输入输出参数: *destData--被清零的数组。
*/
void ClearAllData(uchar ucARRAY_MAX,uchar *destData)
{
  uchar i;
  for(i=0;i<ucARRAY_MAX;i++)
  {
     destData[i]=0;
  }
}
/* 注释五:
*函数介绍: 获取数组的有效长度
*输入参数: *destData--被获取的数组。
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*返回值  : 返回数组的有效长度。比如58786这个数据的有效长度是5
*电子开发者作者: 吴坚鸿
*/
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX)
{
  uchar i;
  uchar DataLength=ucARRAY_MAX;
  for(i=0;i<ucARRAY_MAX;i++)
  {
      if(0!=destData[ucARRAY_MAX-1-i])
          {
             break;
          }
          else
          {
             DataLength--;
          }
  }
  return DataLength;
}
/* 注释六:
*函数介绍: 两个数相加
```

```
*输入参数:
*（1）*destData--被加数的数组。
*（2）*sourceData--加数的数组。
*（3）*resultData--和的数组。注意，调用本函数前，必须先把这个数组清零
*返回值  : 10代表计算结果超出范围出错，11代表正常。
*/
uchar AddData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar addResult=11; //开始默认返回的运算结果是正常
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
uchar carryData=0;  //进位
uchar maxCnt=0; //最大位数
uchar resultTemp=0; //存放临时运算结果的中间变量
//为什么不在本函数内先把resultData数组清零？因为后面章节中的乘法运算中要用到此函数实现连加功能。
//因此如果纯粹实现加法运算时，在调用本函数之前，必须先在外面把和的数组清零，否则会计算出错。
destCnt=GetDataLength(destData,BCD8_MAX);   //获取被加数的有效位数
sourceCnt=GetDataLength(sourceData,BCD8_MAX);  //获取加数的有效位数
if(destCnt>=sourceCnt)  //找出两个运算数据中最大的有效位数
{
   maxCnt=destCnt;
}
else
{
   maxCnt=sourceCnt;
}
for(i=0;i<maxCnt;i++)
{
   resultTemp=destData[i]+sourceData[i]+carryData; //按位相加
   resultData[i]=resultTemp%10;   //截取最低位存放进保存结果的数组
   carryData=resultTemp/10;    //存放进位
}
resultData[i]=carryData;
if((maxCnt==BCD8_MAX)&&(carryData==1))  //如果数组的有效位是最大值并且最后的进位是1，则计算溢出报错
{
  ClearAllData(BCD8_MAX,resultData);
  addResult=10;  //报错
}
return addResult;
}
/* 注释七:
*函数介绍: 数组向大索引值移位，移一位相当于放大10倍
*输入参数: *destData--被移位的数组。
*输入参数: enlarge_cnt--被移位的个数。
*/
void EnlargeData(uchar *destData,uchar enlarge_cnt)
{
  uchar i;
  if(enlarge_cnt!=0)
```

```
  {
    for(i=0;i<(BCD8_MAX-enlarge_cnt);i++)
        {
       destData[BCD8_MAX-1-i]=destData[BCD8_MAX-1-enlarge_cnt-i];
        }
    for(i=0;i<enlarge_cnt;i++) //最低位被移空的补上0
        {
      destData[i]=0;
        }
  }
}
/* 注释八:
*函数介绍: 两个数相乘
*输入参数:
*(1)*destData--被乘数的数组。
*(2)*sourceData--乘数的数组。
*(3)*resultData--积的数组。
*返回值 : 10代表计算结果超出范围出错, 11代表正常。
*/
uchar MultData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar multResult=11; //开始默认正常
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
uchar j;
uchar carryData=0;  //进位
uchar resultTemp=0; //存放临时运算结果的中间变量
uchar nc_add_result;  //接收相加的运算是否超出范围, 这里不用判断, 因为不会溢出
uchar multArrayTemp[BCD8_MAX]; //存放临时运算结果的数组中间变量
destCnt=GetDataLength(destData,BCD8_MAX);      //获取被乘数的长度
sourceCnt=GetDataLength(sourceData,BCD8_MAX);   //获取乘数的长度
ClearAllData(BCD8_MAX,resultData);   //清零存储的结果
if((0==destCnt)||(0==sourceCnt)) //被乘数或者乘数为0, 则结果为0
{
   return multResult;
}
if((destCnt+sourceCnt+2)>BCD8_MAX)
{
   multResult=10; //运算结果有可能超范围报错
   return multResult;
}
for(i=0;i<sourceCnt;i++)  //乘数
{
   carryData=0; //清零进位
   ClearAllData(BCD8_MAX,multArrayTemp); //清零一位乘数相乘的结果中间变量数组
   for(j=0;j<destCnt;j++) //被乘数
   {
      resultTemp=destData[j]*sourceData[i]+carryData;  //乘数的一位依次与被乘数各位相乘,并且加进位
      multArrayTemp[j]=resultTemp%10;  //存储一位乘数相乘的结果
```

```
            carryData=resultTemp/10; //保存进位
    }
    multArrayTemp[j]=carryData; //存储最后的进位
    EnlargeData(multArrayTemp,i); //移位。移一次相当于放大10倍。
    nc_add_result=AddData(resultData,multArrayTemp,resultData); //把一位乘数相乘的结果存储进总结果
}
return multResult;
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     unsigned char k=0;
         unsigned char ucGetDataStep=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
            uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
            while(uiRcMoveIndex<uiRcregTotal)  //说明还没有把缓冲区的数据读取完
            {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
               {

                   i=0;
                   ucGetDataStep=0;
                   ucDataBCD4_cnt_1=0;  //第1个数组合BCD码数组的有效数据长度
                   ucDataBCD4_cnt_2=0;  //第2个数组合BCD码数组的有效数据长度
                   ClearAllData(BCD4_MAX,ucDataBCD4_1);  //清零第1个参与运算的数据
                   ClearAllData(BCD4_MAX,ucDataBCD4_2);  //清零第2个参与运算的数据
                   //以下while循环是通过关键字0x0d 0x0a来截取第1个和第2个参与运算的数据。
                   while(i<(BCD8_MAX+4))//这里+4是因为有2对0x0d 0x0a结尾特殊符号，一个共4个字节
                   {
                      if(ucGetDataStep==0)//步骤0，相当于我平时用的case 0,获取第1个数，在这里是指被乘数
                      {
                           if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a)
//结束标志
                           {
                                for(k=0;k<ucDataBCD4_cnt_1;k++) //提取第1个参与运算的数组数据
                                {
                                    ucDataBCD4_1[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的
数组数据与实际存储的数组数据的下标方向是相反的
                                }

                                i=i+2; //跳过 0x0d 0x0a 这两个字节，进行下一轮的关键字提取
                                ucGetDataStep=1;  //切换到下一个关键字提取的步骤
                           }
                           else
                           {
                                i++;
```

```
                    ucDataBCD4_cnt_1++;  //统计第1个有效数据的长度
                }

            }
            else if(ucGetDataStep==1) //步骤1，相当于我平时用的case 1,获取第2个参与运行的数，在这里是乘数
            {
                if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a) //结束标志
                {
                    for(k=0;k<ucDataBCD4_cnt_2;k++) //提取第2个参与运算的数组数据
                    {
                        ucDataBCD4_2[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的数组数据与实际存储的数组数据的下标方向是相反的
                    }

                    break; //截取数据完成。直接跳出截取数据的while(i<(BCD8_MAX+4))循环
                }
                else
                {
                    i++;
                    ucDataBCD4_cnt_2++;  //统计第2个有效数据的长度
                }
            }
        }
        //注意ucDataBCD8_cnt_1和ucDataBCD8_cnt_2要带地址符号&传址进去
        BCD4_to_BCD8(ucDataBCD4_1,ucDataBCD4_cnt_1,ucDataBCD8_1,&ucDataBCD8_cnt_1); //把接收到的组合BCD码转换成非组合BCD码  第1个数
        BCD4_to_BCD8(ucDataBCD4_2,ucDataBCD4_cnt_2,ucDataBCD8_2,&ucDataBCD8_cnt_2); //把接收到的组合BCD码转换成非组合BCD码  第2个数
        ClearAllData(BCD8_MAX,ucDataBCD8_3);  //清零第3个参与运算的数据,用来接收运行的结果
        ucResultFlag=MultData(ucDataBCD8_1,ucDataBCD8_2,ucDataBCD8_3); //相乘运算，结果放在ucDataBCD8_3数组里
        if(ucResultFlag==11) //表示运算结果没有超范围
        {
            ucDataBCD8_cnt_3=GetDataLength(ucDataBCD8_3,BCD8_MAX);  //获取运算结果的有效字节数
            if(ucDataBCD8_cnt_3==0) //如果1个有效位数都没有，表示数组所有的数据都是0，这个时候的有效位数应该人为的默认是1,表示一个0
                                    {
                                        ucDataBCD8_cnt_3=1;
                                    }
            BCD8_to_BCD4(ucDataBCD8_3,ucDataBCD8_cnt_3,ucDataBCD4_3,&ucDataBCD4_cnt_3); //把非组合BCD码转成组合BCD码。注意，&ucDataBCD4_cnt_3带地址符号&
            for(k=0;k<ucDataBCD4_cnt_3;k++) //返回运算结果到上位机上观察。看到的是组合BCD码形式。返回的时候注意数组下标的顺序要反过来发送，先发高位的下标数组
            {
                eusart_send(ucDataBCD4_3[ucDataBCD4_cnt_3-1-k]); //往上位机发送一个字节的函数
            }
        }
```

```
                    else //运算结果超范围，返回EE EE EE
                    {
                       eusart_send(0xee); //往上位机发送一个字节的函数
                       eusart_send(0xee); //往上位机发送一个字节的函数
                       eusart_send(0xee); //往上位机发送一个字节的函数
                    }
                    break;   //退出循环
                }
                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
            }
            ucRcregBuf[0]=0; //把数据头清零，方便下次接收判断新数据
            ucRcregBuf[1]=0;
            ucRcregBuf[2]=0;

            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

    }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                 //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
```

```
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;     //开总中断
```

```
    ES=1;       //允许串口中断
    ET0=1;      //允许定时中断
    TR0=1;      //启动定时中断
}
```
复制代码

总结陈词:

既然这节讲了乘法程序，那么下一节接着讲常用的除法程序，这种大数据的除法程序是什么样的？欲知详情，请听下回分解----大数据的除法运算。

（未完待续，下节更精】彩，不要走开哦）

第六十五节：大数据的除法运算。

开场白:

直接用C语言的“/”运算符进行除法运算时，“被除数”，“ 除数”，“商”，这三个数据的最大范围是unsigned long 类型，也就是数据最大范围是4个字节，十进制的范围是0至4294967295。一旦超过了这个范围，则运算会出错。因此，当进行大数据除法运算时，我们要额外编程序，实现大数据的算法。其实这种算法并不难，就是我们在小学里学的四则运算算法。

我们先要弄清楚一个新的概念。不考虑小数点的情况下，数据有两种表现形式。一种是常用的变量形式，另外一种是BCD码数组形式。变量的最大范围有限，而BCD码数组的形式是无限的，正因为这个特点，所以我们可以进行大数据运算。

这一节要教大家一个知识点:

第一个: 如何编写涉及到大数据除法运算的算法程序函数，同时也复习了指针的用途。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能:

波特率是: 9600 。

通过电脑串口调试助手模拟上位机，往单片机发送组合BCD码的被除数和除数，单片机把组合BCD码的运算结果返回到上位机。被除数与除数的最大范围都是从0到9999，如果运算的商超过允许保存的最大位数范围或者除数为0，则返回EE EE EE报错。

往单片机发送的数据格式: EB 00 55 XX XX 0d 0a YY YY 0d 0a指令，其中EB 00 55是数据头，XX XX是被除数，是1到2个字节的组合BCD码。YY YY是除数，是1到2个字节的组合BCD码。0d 0a是固定的结束标志。

例如:

（a）9816 ÷ 8= 1227

上位机发送数据: eb 00 55 98 16 0d 0a 08 0d 0a

单片机返回: 12 27

（b）9816 ÷ 0= 出错了，除数不能为0。

上位机发送数据: eb 00 55 98 16 0d 0a 00 0d 0a

单片机返回: EE EE EE

（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 本系统中的除法运算，规定被除数和除数的最大范围是0至9999.
* 由于STC89C52单片机的RAM只有256个，也就是说系统的变量数最大
* 不能超过256个，如果超过了这个极限，编译器就会报错。由于51单片机RAM资源有限，
* 因此规定除数的最大范围不能超过9999，如果这个算法移植到stm32或者PIC等RAM比较大
* 的单片机上，那么就可以把这个运算位数设置得更加大一点。调整下面 BCD4_MAX的大小，
* 可以调整运算的数据范围。
*/
#define  BCD4_MAX     3  //调整BCD4_MAX的大小，可以调整运算的数据范围。
#define  BCD8_MAX    (BCD4_MAX*2)  //本系统中，规定的非组合BCD码能保存的最大字节数，一个字节包含1位有效
```

运算数

```
#define const_rc_size  30  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define uchar unsigned char    //方便移植平台
#define ulong unsigned long   //方便移植平台
//如果在VC的平台模拟此算法，则都定义成int类型，如下:
//#define uchar int
//#define ulong int
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char *p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt);
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char *p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt);
void ClearAllData(uchar ucARRAY_MAX,uchar *destData);
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX);
uchar AddData(const uchar *destData,const uchar *sourceData,uchar *resultData);  //两个数相加
uchar CmpData(const uchar *destData,const uchar *sourceData); //比较两个数的大小
uchar SubData(const uchar *destData,const uchar *sourceData,uchar *resultData);//两个数相减
void EnlargeData(uchar *destData,uchar enlarge_cnt); //数组向大索引值移位，移一位相当于放大10倍
uchar MultData(const uchar *destData,const uchar *sourceData,uchar *resultData); //两个数相乘
uchar DivLessTenData(const uchar *destData,const uchar *sourceData,uchar *resultData,uchar *remData);//局部两个数相除，商不超过10。当商为0时，余数等于被除数
uchar Div(const uchar *destData,const uchar *sourceData,uchar *resultData);//两个数相除
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned char ucDataBCD4_1[BCD4_MAX]; //接收到的第1个数组合BCD码数组形式  这里是指被乘数
unsigned char ucDataBCD4_cnt_1=0;  //接收到的第1个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_2[BCD4_MAX]; //接收到的第2个数组合BCD码数组形式  这里是指乘数
unsigned char ucDataBCD4_cnt_2=0;  //接收到的第2个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD4_3[BCD4_MAX]; //接收到的第3个数组合BCD码数组形式  这里是指积
unsigned char ucDataBCD4_cnt_3=0;  //接收到的第3个数组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_1[BCD8_MAX]; //接收到的第1个数非组合BCD码数组形式   这里是指被乘数
unsigned char ucDataBCD8_cnt_1=0;  //接收到的第1个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_2[BCD8_MAX]; //接收到的第2个数非组合BCD码数组形式   这里是指乘数
unsigned char ucDataBCD8_cnt_2=0;  //接收到的第2个数非组合BCD码数组的有效数据长度
unsigned char ucDataBCD8_3[BCD8_MAX]; //接收到的第3个数非组合BCD码数组形式   这里是指积
unsigned char ucDataBCD8_cnt_3=0;  //接收到的第3个数非组合BCD码数组的有效数据长度
unsigned char ucResultFlag=11; //运算结果标志，10代表计算结果超出范围出错，11代表正常。
```

```
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       usart_service();  //串口服务程序
   }
}
/* 注释二:
* 组合BCD码转成非组合BCD码。
* 这里的变量ucBCD4_cnt代表组合BCD码的有效字节数.
* 这里的变量*p_ucBCD8_cnt代表经过转换后，非组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD4_to_BCD8(const unsigned char *p_ucBCD_bit4,unsigned char ucBCD4_cnt,unsigned char
*p_ucBCD_bit8,unsigned char *p_ucBCD8_cnt)
{
   unsigned char ucTmep;
   unsigned char i;
   for(i=0;i<BCD8_MAX;i++)    //先把即将保存转换结果的缓冲区清零
   {
      p_ucBCD_bit8[i]=0;
   }
   *p_ucBCD8_cnt=ucBCD4_cnt*2; //转换成非组合BCD码后的有效数据长度
   for(i=0;i<ucBCD4_cnt;i++)
   {
      ucTmep=p_ucBCD_bit4[ucBCD4_cnt-1-i];
      p_ucBCD_bit8[ucBCD4_cnt*2-i*2-1]=ucTmep>>4;
      p_ucBCD_bit8[ucBCD4_cnt*2-i*2-2]=ucTmep&0x0f;
   }
}
/* 注释三:
* 非组合BCD码转成组合BCD码。
* 这里的变量ucBCD8_cnt代表非组合BCD码的有效字节数.
* 这里的变量*p_ucBCD4_cnt代表经过转换后，组合BCD码的有效字节数，记得加地址符号&传址进去
* 本程序在上一节的基础上，略作修改，用循环for语句压缩了代码，
* 同时引进了非组合BCD码的有效字节数变量。这样就不限定了数据的长度，
* 可以让我们根据数据的实际大小灵活运用。
*/
void BCD8_to_BCD4(const unsigned char *p_ucBCD_bit8,unsigned char ucBCD8_cnt,unsigned char
*p_ucBCD_bit4,unsigned char *p_ucBCD4_cnt)
{
   unsigned char ucTmep;
   unsigned char i;
   unsigned char ucBCD4_cnt;
   for(i=0;i<BCD4_MAX;i++)    //先把即将保存转换结果的缓冲区清零
```

```
    {
       p_ucBCD_bit4[i]=0;
    }
    ucBCD4_cnt=(ucBCD8_cnt+1)/2; //非组合BCD码转化成组合BCD码的有效数,这里+1避免非组合数据长度是奇数位
    *p_ucBCD4_cnt=ucBCD4_cnt; //把转换后的结果付给接口指针的数据，可以对外输出结果
    for(i=0;i<ucBCD4_cnt;i++)
    {
       ucTmep=p_ucBCD_bit8[ucBCD4_cnt*2-1-i*2];     //把非组合BCD码第8位分解出来
       p_ucBCD_bit4[ucBCD4_cnt-1-i]=ucTmep<<4;
       p_ucBCD_bit4[ucBCD4_cnt-1-i]=p_ucBCD_bit4[ucBCD4_cnt-1-i]+p_ucBCD_bit8[ucBCD4_cnt*2-2-i*2];
//把非组合BCD码第7位分解出来
    }

}
/* 注释四:
*函数介绍: 清零数组的全部数组数据
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*输入输出参数: *destData--被清零的数组。
*/
void ClearAllData(uchar ucARRAY_MAX,uchar *destData)
{
  uchar i;
  for(i=0;i<ucARRAY_MAX;i++)
  {
     destData[i]=0;
  }
}
/* 注释五:
*函数介绍: 获取数组的有效长度
*输入参数: *destData--被获取的数组。
*输入参数: ucARRAY_MAX代表数组定义的最大长度
*返回值  : 返回数组的有效长度。比如58786这个数据的有效长度是5
*电子开发者作者: 吴坚鸿
*/
uchar GetDataLength(const uchar *destData,uchar ucARRAY_MAX)
{
  uchar i;
  uchar DataLength=ucARRAY_MAX;
  for(i=0;i<ucARRAY_MAX;i++)
  {
      if(0!=destData[ucARRAY_MAX-1-i])
          {
             break;
          }
          else
          {
             DataLength--;
          }
  }
  return DataLength;
```

```
}
/* 注释六:
*函数介绍: 比较两个数的大小
*输入参数:
*（1）*destData--被比较数的数组。
*（2）*sourceData--比较数的数组。
*返回值 : 9代表小于，10代表相等，11代表大于。
*/
uchar CmpData(const uchar *destData,const uchar *sourceData)
{
uchar cmpResult=10; //开始默认相等
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
destCnt=GetDataLength(destData,BCD8_MAX);
sourceCnt=GetDataLength(sourceData,BCD8_MAX);
if(destCnt>sourceCnt)  //大于
{
  cmpResult=11;
}
else if(destCnt<sourceCnt) //小于
{
  cmpResult=9;
}
else if((destCnt==0)&&(sourceCnt==0))  //如果都是等于0则等于
{
  cmpResult=10;
}
else  //否则就要继续判断
{
  for(i=0;i<destCnt;i++)
  {
     if(destData[destCnt-1-i]>sourceData[destCnt-1-i])   //从最高位开始判断,如果最高位大于则大于
         {
           cmpResult=11;
           break;
         }
     else if(destData[destCnt-1-i]<sourceData[destCnt-1-i])  //从最高位开始判断,如果最高位小于则小于
         {
           cmpResult=9;
           break;
         }
     //否则继续判断下一位
  }
}
return cmpResult;
}
/* 注释七:
*函数介绍: 两个数相减
*输入参数:
```

```
*（1）*destData--被减数的数组。
*（2）*sourceData--减数的数组。
*（3）*resultData--差的数组。注意，调用本函数前，必须先把这个数组清零
*返回值  ：10代表计算结果是负数或者超出范围出错，11代表正常。
*/
uchar SubData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar subResult=11; //开始默认正常
uchar destCnt=0;
uchar i;
uchar carryData=0;  //进位
uchar maxCnt=0; //最大位数
uchar resultTemp=0; //存放临时运算结果的中间变量
//为什么不在本函数内先把resultData数组清零？因为后面章节中的除法运算中要用到此函数实现连减功能。
//因此如果纯粹实现减法运算时，在调用本函数之前，必须先在外面把差的数组清零，否则会计算出错。
if(CmpData(destData,sourceData)==9)  //被减数小于减数，报错
{
   subResult=10;
   return subResult;  //返回判断结果，并且退出本程序，不往下执行本程序余下代码
}
destCnt=GetDataLength(destData,BCD8_MAX);  //获取被减数的有效数据长度
maxCnt=destCnt;
for(i=0;i<maxCnt;i++)
{
   resultTemp=sourceData[i]+carryData; //按位相加
   if(resultTemp>destData[i])
   {
      resultData[i]=destData[i]+10-sourceData[i]-carryData;    //借位
          carryData=1;
   }
   else
   {
      resultData[i]=destData[i]-sourceData[i]-carryData;    //不用借位
          carryData=0;
   }
}
return subResult;
}
/* 注释八:
*函数介绍: 两个数相加
*输入参数:
*（1）*destData--被加数的数组。
*（2）*sourceData--加数的数组。
*（3）*resultData--和的数组。注意，调用本函数前，必须先把这个数组清零
*返回值  ：10代表计算结果超出范围出错，11代表正常。
*/
uchar AddData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar addResult=11; //开始默认返回的运算结果是正常
uchar destCnt=0;
```

```
uchar sourceCnt=0;
uchar i;
uchar carryData=0;  //进位
uchar maxCnt=0; //最大位数
uchar resultTemp=0; //存放临时运算结果的中间变量
//为什么不在本函数内先把resultData数组清零？因为后面章节中的乘法运算中要用到此函数实现连加功能。
//因此如果纯粹实现加法运算时，在调用本函数之前，必须先在外面把和的数组清零，否则会计算出错。
destCnt=GetDataLength(destData,BCD8_MAX);   //获取被加数的有效位数
sourceCnt=GetDataLength(sourceData,BCD8_MAX);  //获取加数的有效位数
if(destCnt>=sourceCnt)  //找出两个运算数据中最大的有效位数
{
   maxCnt=destCnt;
}
else
{
   maxCnt=sourceCnt;
}
for(i=0;i<maxCnt;i++)
{
   resultTemp=destData[i]+sourceData[i]+carryData; //按位相加
   resultData[i]=resultTemp%10;   //截取最低位存放进保存结果的数组
   carryData=resultTemp/10;    //存放进位
}
resultData[i]=carryData;
if((maxCnt==BCD8_MAX)&&(carryData==1))  //如果数组的有效位是最大值并且最后的进位是1，则计算溢出报错
{
  ClearAllData(BCD8_MAX,resultData);
  addResult=10;  //报错
}
return addResult;
}
/* 注释九:
*函数介绍: 数组向大索引值移位，移一位相当于放大10倍
*输入参数: *destData--被移位的数组。
*输入参数: enlarge_cnt--被移位的个数。
*/
void EnlargeData(uchar *destData,uchar enlarge_cnt)
{
  uchar i;
  if(enlarge_cnt!=0)
  {
    for(i=0;i<(BCD8_MAX-enlarge_cnt);i++)
        {
       destData[BCD8_MAX-1-i]=destData[BCD8_MAX-1-enlarge_cnt-i];
        }
    for(i=0;i<enlarge_cnt;i++) //最低位被移空的补上0
        {
      destData[i]=0;
        }
  }
```

```
}
/* 注释十:
*函数介绍: 两个数相乘
*输入参数:
*(1)*destData--被乘数的数组。
*(2)*sourceData--乘数的数组。
*(3)*resultData--积的数组。
*返回值 : 10代表计算结果超出范围出错，11代表正常。
*/
uchar MultData(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar multResult=11; //开始默认正常
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
uchar j;
uchar carryData=0;  //进位
uchar resultTemp=0; //存放临时运算结果的中间变量
uchar nc_add_result;  //接收相加的运算是否超出范围，这里不用判断，因为不会溢出
uchar multArrayTemp[BCD8_MAX]; //存放临时运算结果的数组中间变量
destCnt=GetDataLength(destData,BCD8_MAX);      //获取被乘数的长度
sourceCnt=GetDataLength(sourceData,BCD8_MAX);   //获取乘数的长度
ClearAllData(BCD8_MAX,resultData);   //清零存储的结果
if((0==destCnt)||(0==sourceCnt)) //被乘数或者乘数为0，则结果为0
{
   return multResult;
}
if((destCnt+sourceCnt+2)>BCD8_MAX)
{
   multResult=10; //运算结果有可能超范围报错
   return multResult;
}
for(i=0;i<sourceCnt;i++)  //乘数
{
   carryData=0; //清零进位
   ClearAllData(BCD8_MAX,multArrayTemp); //清零一位乘数相乘的结果中间变量数组
   for(j=0;j<destCnt;j++) //被乘数
   {
      resultTemp=destData[j]*sourceData[i]+carryData;  //乘数的一位依次与被乘数各位相乘,并且加进位
      multArrayTemp[j]=resultTemp%10;  //存储一位乘数相乘的结果
          carryData=resultTemp/10; //保存进位
   }
   multArrayTemp[j]=carryData; //存储最后的进位
   EnlargeData(multArrayTemp,i); //移位。移一次相当于放大10倍。
   nc_add_result=AddData(resultData,multArrayTemp,resultData); //把一位乘数相乘的结果存储进总结果
}
return multResult;
}
/* 注释十一:
*函数介绍: 局部两个数相除，商不超过10。当商为0时，余数是被除数
```

```
*原理精髓: 根据手工除法的原理，我们都是从高位开始借位相除，此时是局部相除，因此商都不超过10，剩下的余数继续借位
*依次除下去。这个程序的除法原理是挨个猜值，反正商是从0，1,2.。。。9这10个数中的其中一个，为了快速找到我们想要的那个商，我是
*利用中间法则进行寻找，先猜是5，然后判断一下是大了还是小了，如果是大了，就猜是3，如果小了就猜是7，最后肯定会找到商。
*输入参数:
*（1）*destData--被除数的数组。
*（2）*sourceData--除数的数组。
*（3）*resultData--商的数,不是数组，传址进去，是0,1,2到9中的某个数
*（4）*remData--余数的数组。
*返回值 : 10代表计算结果超出范围出错，11代表正常。
*/
uchar DivLessTenData(const uchar *destData,const uchar *sourceData,uchar *resultData,uchar *remData)
{
uchar DivLessTenResult=11; //开始默认正常
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
uchar resultRunStep=5;
uchar cmpError=10;
uchar DivLessTenArrayTemp[BCD8_MAX]; //存放临时运算结果的数组中间变量
uchar DivLessTenArrayResult[BCD8_MAX]; //存放临时运算结果的数组中间变量的结果
uchar DivLessTenArrayBackup[BCD8_MAX]; //存放临时运算结果的数组中间变量的备份
uchar while_flag=0;  //结束猜算的中间变量
uchar multError=11;
uchar subError=11;
destCnt=GetDataLength(destData,BCD8_MAX);      //获取被除数的数据有效长度
sourceCnt=GetDataLength(sourceData,BCD8_MAX); //获取除数的数据有效长度
cmpError=CmpData(destData,sourceData); //比较被除数和除数的大小
ClearAllData(BCD8_MAX,remData); //清空余数,余数为0
if(cmpError==9) //被除数比除数小
{
   *resultData=0;    //商肯定为0
   for(i=0;i<destCnt;i++)
   {
     remData[i]=destData[i]; //余数等于被除数
   }
   return DivLessTenResult;
}
else if(cmpError==10) //被除数与除数相等
{
   *resultData=1; //商等于1余数为0
   return DivLessTenResult;
}
else  //开始猜值
{
   resultRunStep=5;  //先猜是5  ,从这里开始直接看以下 case 5 的详细讲解，其他case原理相同
   while_flag=0;
   while(1)
```

```
{
   switch(resultRunStep)
       {
            case 1:
                    ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
                    ClearAllData(BCD8_MAX,DivLessTenArrayResult);
          DivLessTenArrayTemp[0]=resultRunStep;
                    multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
          subError=SubData(destData,DivLessTenArrayResult,remData);//求余数
          *resultData=1; //商等于1
                 while_flag=1; //退出循环
                    break;
               case 2:
                    ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
                    ClearAllData(BCD8_MAX,DivLessTenArrayResult);
          DivLessTenArrayTemp[0]=resultRunStep;
                    multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
          cmpError=CmpData(DivLessTenArrayResult,destData);
                    if(cmpError==10) //等于
                    {
               *resultData=2; //商等于2余数为0
                             while_flag=1; //退出循环
                    }
                    else if(cmpError==11) //大于
                    {
               resultRunStep=1;
                    }
                    else             //小于
                    {
               subError=SubData(destData,DivLessTenArrayResult,remData);//求余数
               *resultData=2; //商等于2
                             while_flag=1; //退出循环
                    }
                    break;
               case 3:
                    ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
                    ClearAllData(BCD8_MAX,DivLessTenArrayResult);
          DivLessTenArrayTemp[0]=resultRunStep;
                    multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
          cmpError=CmpData(DivLessTenArrayResult,destData);
                    if(cmpError==10) //等于
                    {
               *resultData=3; //商等于3余数为0
                             while_flag=1; //退出循环
                    }
                    else if(cmpError==11) //大于
                    {
             resultRunStep=2;
                    }
                    else             //小于
```

```
                {
            resultRunStep=4;
                   ClearAllData(BCD8_MAX,DivLessTenArrayBackup);
                   for(i=0;i<BCD8_MAX;i++) //备份
                       {
                     DivLessTenArrayBackup[i]=DivLessTenArrayResult[i];
                       }
                }
                break;
             case 4:
                ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
                ClearAllData(BCD8_MAX,DivLessTenArrayResult);
        DivLessTenArrayTemp[0]=resultRunStep;
                multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
        cmpError=CmpData(DivLessTenArrayResult,destData);
                if(cmpError==10) //等于
                {
             *resultData=4; //商等于4余数为0
                        while_flag=1; //退出循环
                }
                else if(cmpError==11) //大于
                {
             subError=SubData(destData,DivLessTenArrayBackup,remData);//求余数
             *resultData=3; //商等于3
                        while_flag=1; //退出循环
                }
                else             //小于
                {
             subError=SubData(destData,DivLessTenArrayResult,remData);//求余数
             *resultData=4; //商等于4
                        while_flag=1; //退出循环
                }
                break;
             case 5:    //重点讲解一下case 5，其它case 原理相同，不多讲
                ClearAllData(BCD8_MAX,DivLessTenArrayTemp); //清空运算中需要用到的中间数组变量
                ClearAllData(BCD8_MAX,DivLessTenArrayResult); //清空运算中需要用到的中间数组变
量
        DivLessTenArrayTemp[0]=resultRunStep;  //把猜的变量形式的商传递给数组形式的变量
                multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);  //猜
的商跟除数像乘，看看结果跟被除数谁大。
        cmpError=CmpData(DivLessTenArrayResult,destData); //猜的商跟除数像乘，看看结果跟被除数谁大
。
                if(cmpError==10) //等于 恭喜猜中是5
                {
             *resultData=5; //商等于5余数为0
                        while_flag=1; //退出循环
                }
                else if(cmpError==11) //大于   猜不中，大了，就继续往小的猜，看看有没有可能是3
                {
            resultRunStep=3;
```

```
            }
            else             //小于        猜不中，小了，就继续往大的猜，看看有没有可能是7
            {
   resultRunStep=7;
               ClearAllData(BCD8_MAX,DivLessTenArrayBackup);
               for(i=0;i<BCD8_MAX;i++) //备份
                    {
                   DivLessTenArrayBackup[i]=DivLessTenArrayResult[i];
                    }
            }
            break;
    case 6:
            ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
            ClearAllData(BCD8_MAX,DivLessTenArrayResult);
DivLessTenArrayTemp[0]=resultRunStep;
            multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
cmpError=CmpData(DivLessTenArrayResult,destData);
            if(cmpError==10) //等于
            {
    *resultData=6; //商等于6余数为0
                    while_flag=1; //退出循环
            }
            else if(cmpError==11) //大于
            {
    subError=SubData(destData,DivLessTenArrayBackup,remData);//求余数
    *resultData=5; //商等于5
                    while_flag=1; //退出循环
            }
            else             //小于
            {
    subError=SubData(destData,DivLessTenArrayResult,remData);//求余数
    *resultData=6; //商等于6
                    while_flag=1; //退出循环
            }
            break;
    case 7:
            ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
            ClearAllData(BCD8_MAX,DivLessTenArrayResult);
DivLessTenArrayTemp[0]=resultRunStep;
            multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
cmpError=CmpData(DivLessTenArrayResult,destData);
            if(cmpError==10) //等于
            {
    *resultData=7; //商等于7余数为0
                    while_flag=1; //退出循环
            }
            else if(cmpError==11) //大于
            {
   resultRunStep=6;
            }
```

```
            else            //小于
            {
   resultRunStep=8;
                ClearAllData(BCD8_MAX,DivLessTenArrayBackup);
               for(i=0;i<BCD8_MAX;i++) //备份
                   {
                  DivLessTenArrayBackup[i]=DivLessTenArrayResult[i];
                   }
            }
            break;
    case 8:
            ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
            ClearAllData(BCD8_MAX,DivLessTenArrayResult);
DivLessTenArrayTemp[0]=resultRunStep;
            multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
cmpError=CmpData(DivLessTenArrayResult,destData);
            if(cmpError==10) //等于
            {
    *resultData=8; //商等于8余数为0
                    while_flag=1; //退出循环
            }
            else if(cmpError==11) //大于
            {
    subError=SubData(destData,DivLessTenArrayBackup,remData);//求余数
    *resultData=7; //商等于7
                    while_flag=1; //退出循环
            }
            else            //小于
            {
   resultRunStep=9;
                ClearAllData(BCD8_MAX,DivLessTenArrayBackup);
               for(i=0;i<BCD8_MAX;i++) //备份
                   {
                  DivLessTenArrayBackup[i]=DivLessTenArrayResult[i];
                   }
            }
            break;
    case 9:
            ClearAllData(BCD8_MAX,DivLessTenArrayTemp);
            ClearAllData(BCD8_MAX,DivLessTenArrayResult);
DivLessTenArrayTemp[0]=resultRunStep;
            multError=MultData(sourceData,DivLessTenArrayTemp,DivLessTenArrayResult);
cmpError=CmpData(DivLessTenArrayResult,destData);
            if(cmpError==10) //等于
            {
    *resultData=9; //商等于9余数为0
                    while_flag=1; //退出循环
            }
            else if(cmpError==11) //大于
            {
```

```
                subError=SubData(destData,DivLessTenArrayBackup,remData);//求余数
                *resultData=8; //商等于8
                            while_flag=1; //退出循环
                    }
                    else            //小于
                    {
                subError=SubData(destData,DivLessTenArrayResult,remData);//求余数
                *resultData=9; //商等于9
                            while_flag=1; //退出循环
                    }
                    break;
          }

      if(while_flag==1)  //猜中了就退出循环
          {
             break;
          }
   }
}
return DivLessTenResult;
}
/* 注释十二:
*函数介绍: 两个数相除
*输入参数:
*(1)*destData--被除数的数组。
*(2)*sourceData--除数的数组。
*(3)*resultData--商的数组
*返回值  : 10代表计算结果超出范围出错，11代表正常。
*/
uchar Div(const uchar *destData,const uchar *sourceData,uchar *resultData)
{
uchar DivResult=11; //开始默认正常
uchar destCnt=0;
uchar sourceCnt=0;
uchar i;
uchar j;
uchar resultTemp=0; //存放临时运算结果的中间变量
uchar DivArrayTemp[BCD8_MAX]; //存放临时运算结果的数组中间变量
uchar DivArrayResult[BCD8_MAX]; //存放临时运算结果的数组中间变量的结果
uchar divError=11;
destCnt=GetDataLength(destData,BCD8_MAX);    //获取被除数的数据有效长度
sourceCnt=GetDataLength(sourceData,BCD8_MAX);    //获取除数的数据有效长度
ClearAllData(BCD8_MAX,resultData);    //把结果清零
if(sourceCnt==0)  //除数为0，报错
{
   DivResult=10; //报错
}
else
{
   ClearAllData(BCD8_MAX,DivArrayTemp); //清零局部被除数的数组
```

```
   for(i=0;i<destCnt;i++)
   {
      DivArrayTemp[0]=destData[destCnt-1-i];  //从被除数的高位开始借位,放到局部被除数的个位
      divError=DivLessTenData(DivArrayTemp,sourceData,&resultTemp,DivArrayResult);  //局部相除，求商
resultTemp，此商resultTemp不超过10
          if(divError==10)  //报错
          {
                DivResult=10;
            break;
          }
          else
          {
        resultData[destCnt-1-i]=resultTemp;  //保存商
            for(j=0;j<(destCnt-1);j++)  //把余数移一次位,相当于放大十倍,重新放进DivArrayTemp数组对应的位
                {
              DivArrayTemp[j+1]=DivArrayResult[j];
                }
          }
   }
}
return DivResult;
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     unsigned char k=0;
         unsigned char ucGetDataStep=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
           ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
           //下面的代码进入数据协议解析和数据处理的阶段
           uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动
           while(uiRcMoveIndex<uiRcregTotal)  //说明还没有把缓冲区的数据读取完
           {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
              {

                  i=0;
                  ucGetDataStep=0;
                  ucDataBCD4_cnt_1=0;  //第1个数组合BCD码数组的有效数据长度
                  ucDataBCD4_cnt_2=0;  //第2个数组合BCD码数组的有效数据长度
                  ClearAllData(BCD4_MAX,ucDataBCD4_1);  //清零第1个参与运算的数据
                  ClearAllData(BCD4_MAX,ucDataBCD4_2);  //清零第2个参与运算的数据
                  //以下while循环是通过关键字0x0d 0x0a来截取第1个和第2个参与运算的数据。
                  while(i<(BCD8_MAX+4))//这里+4是因为有2对0x0d 0x0a结尾特殊符号，一个共4个字节
                  {
                     if(ucGetDataStep==0)//步骤0，相当于我平时用的case 0,获取第1个数，在这里是指被除数
                     {
                          if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a)
```

```
//结束标志
                        {
                            for(k=0;k<ucDataBCD4_cnt_1;k++) //提取第1个参与运算的数组数据
                            {
                                ucDataBCD4_1[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的
数组数据与实际存储的数组数据的下标方向是相反的
                            }

                            i=i+2; //跳过 0x0d 0x0a 这两个字节，进行下一轮的关键字提取
                            ucGetDataStep=1;  //切换到下一个关键字提取的步骤
                        }
                        else
                        {
                            i++;
                            ucDataBCD4_cnt_1++;  //统计第1个有效数据的长度
                        }

                    }
                    else if(ucGetDataStep==1) //步骤1，相当于我平时用的case 1,获取第2个参与运行的数
，在这里是除数
                    {
                        if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a)
//结束标志
                        {
                            for(k=0;k<ucDataBCD4_cnt_2;k++) //提取第2个参与运算的数组数据
                            {
                                ucDataBCD4_2[k]=ucRcregBuf[uiRcMoveIndex+3+i-1-k]; //注意，接收到的
数组数据与实际存储的数组数据的下标方向是相反的
                            }

                            break; //截取数据完成。直接跳出截取数据的while(i<(BCD8_MAX+4))循环
                        }
                        else
                        {
                            i++;
                            ucDataBCD4_cnt_2++;  //统计第2个有效数据的长度
                        }
                    }
                }
                //注意ucDataBCD8_cnt_1和ucDataBCD8_cnt_2要带地址符号&传址进去
                BCD4_to_BCD8(ucDataBCD4_1,ucDataBCD4_cnt_1,ucDataBCD8_1,&ucDataBCD8_cnt_1); //把接收
到的组合BCD码转换成非组合BCD码  第1个数
                BCD4_to_BCD8(ucDataBCD4_2,ucDataBCD4_cnt_2,ucDataBCD8_2,&ucDataBCD8_cnt_2); //把接收
到的组合BCD码转换成非组合BCD码  第2个数
                ClearAllData(BCD8_MAX,ucDataBCD8_3);  //清零第3个参与运算的数据,用来接收运行的结果
                ucResultFlag=Div(ucDataBCD8_1,ucDataBCD8_2,ucDataBCD8_3); //相除运算，结果放在
ucDataBCD8_3数组里
                if(ucResultFlag==11) //表示运算结果没有超范围
                {
                    ucDataBCD8_cnt_3=GetDataLength(ucDataBCD8_3,BCD8_MAX);  //获取运算结果的有效字节
```

```
数
                        if(ucDataBCD8_cnt_3==0) //如果1个有效位数都没有，表示数组所有的数据都是0，这个时候的有效位数应该人为的默认是1位数据,表示一个0
                        {
                           ucDataBCD8_cnt_3=1;
                        }
                        BCD8_to_BCD4(ucDataBCD8_3,ucDataBCD8_cnt_3,ucDataBCD4_3,&ucDataBCD4_cnt_3); //把非组合BCD码转成组合BCD码。注意，&ucDataBCD4_cnt_3带地址符号&
                        for(k=0;k<ucDataBCD4_cnt_3;k++) //返回运算结果到上位机上观察。看到的是组合BCD码形式。返回的时候注意数组下标的顺序要反过来发送，先发高位的下标数组
                        {
                           eusart_send(ucDataBCD4_3[ucDataBCD4_cnt_3-1-k]); //往上位机发送一个字节的函数
                        }
                     }
                     else //运算结果超范围，返回EE EE EE
                     {
                        eusart_send(0xee); //往上位机发送一个字节的函数
                        eusart_send(0xee); //往上位机发送一个字节的函数
                        eusart_send(0xee); //往上位机发送一个字节的函数
                     }
                     break;    //退出循环
                  }
                  uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
               }
               ucRcregBuf[0]=0; //把数据头清零，方便下次接收判断新数据
               ucRcregBuf[1]=0;
               ucRcregBuf[2]=0;

               uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

         }

}
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
  ES = 0; //关串口中断
  TI = 0; //清零串口发送完成中断请求标志
  SBUF =ucSendData; //发送一个字节
  delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项目来调整
  TI = 0; //清零串口发送完成中断请求标志
  ES = 1; //允许串口中断
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)    //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这
```

```
个中间没有串口数据过来
      ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                    //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

   }
   else  //发送中断，及时把发送中断标志位清零
   {
        TI = 0;
   }

}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
```

```
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

前面四个章节讲完了四则运算的大数据算法，下一节讲单片机的外部中断功能。外部中断是单片机非常重要的内部资源，应用很广，它是单片机的高速开关感应器输入接口，它可以检测脉冲输入，可以接收红外遥控器的输入信号，可以检测高速运转的车轮或者电机圆周运动的反馈信号，可以检测输液器里瞬间即逝的水滴信号，可以接收模拟串口的数据信息，等等。单片机外部中断的有什么特点？欲知详情，请听下回分解----单片机外部中断的基础。

（未完待续，下节更精彩，不要走开哦）

第六十六节：单片机外部中断的基础。

开场白:

外部中断是单片机非常重要的内部资源，应用很广，它是单片机的高速开关感应器输入接口，它可以检测脉冲输入，可以接收红外遥控器的输入信号，可以检测高速运转的车轮或者电机圆周运动的反馈信号，可以检测输液器里瞬间即逝的水滴信号，可以接收模拟串口的数据信息，等等。

这一节要教大家两个知识点:

第一个：外部中断的初始化代码和中断函数的基本程序模板。

第二个：当系统存在两种中断以上时，如何设置外部中断0为最高优先级，实现中断嵌套功能。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。用S1按键作为模拟外部中断0的下降沿脉冲输入。原来S1按键是直接连接到P0^0口的，因此必须通过跳线把P0^0口连接到单片机外部中断0专用IO口P3^2上，只需把P0^0和P3^2的两根黄颜色跳冒去掉，通过一根线把P0^0和P3^2相互连接起来即可。这时每按下一次S1按键，就会给P3^2口产生一个下降沿的脉冲，然后程序会自动跳到中断函数中执行一次。

（2）实现功能:

用数码管低4位显示记录当前的下降沿脉冲数。用S1按键经过跳线后模拟外部中断0的下降沿输入，每按一次数码管就会显示往上累加的脉冲数。由于按键按下去的时候有抖动，也就按一次可能产生几个脉冲，所以按一次往往看到数据一次加了三四个，这种实验现象都是正常的。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
```

```
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void INT0_int();//外部0中断函数
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Update=1; //窗口1更新显示标志
unsigned char ucWd=1;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。本程序只有一个显示窗口
unsigned int  uiPluseCnt=0;  //本程序中累加中断脉冲数的变量
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp3=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
```

```
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
       display_service(); //显示的窗口菜单服务程序
   }
}
void display_service() //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示第一个窗口的数据  本系统中只有一个显示窗口
            if(ucWd1Update==1)  //窗口1要全部更新显示
            {
               ucWd1Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow8=10;  //第8位数码管显示无
               ucDigShow7=10;  //第7位数码管显示无
               ucDigShow6=10;  //第6位数码管显示无
               ucDigShow5=10;  //第5位数码管显示无
              //先分解数据
               ucTemp4=uiPluseCnt/1000;
               ucTemp3=uiPluseCnt%1000/100;
               ucTemp2=uiPluseCnt%100/10;
               ucTemp1=uiPluseCnt%10;

             //再过渡需要显示的数据到缓冲变量里，让过渡的时间越短越好
              //以下增加的if判断就是略作修改，把整个4位数据中高位为0的去掉不显示。
               if(uiPluseCnt<1000)
               {
                   ucDigShow4=10;  //如果小于1000，千位显示无
               }
               else
               {
                  ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
               }
               if(uiPluseCnt<100)
               {
                  ucDigShow3=10;  //如果小于100，百位显示无
               }
               else
```

```
                {
                    ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                }
                if(uiPluseCnt<10)
                {
                    ucDigShow2=10;  //如果小于10，十位显示无
                }
                else
                {
                    ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                }
                ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
            }
            break;

    }

}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                if(ucDigDot1==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                if(ucDigDot2==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                if(ucDigDot3==1)
                {
                    ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
```

```
                    {
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xf7);
                break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                    if(ucDigDot5==1)
                    {
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                    if(ucDigDot6==1)
                    {
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                    if(ucDigDot7==1)
                    {
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                    if(ucDigDot8==1)
                    {
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
```

```
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
void T0_time() interrupt 1   //定时器中断函数
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
/* 注释一:
* 用朱兆祺51学习板中的S1按键作为模拟外部中断0的下降沿脉冲输入。
* 原来S1按键是直接连接到P0^0口的，因此必须通过跳线把P0^0口连接到
* 单片机外部中断0专用IO口P3^2上，只需把P0^0和P3^2的两个黄颜色跳冒去掉，通过一根
* 线把P0^0和P3^2相互连接起来即可。这时每按下一次S1按键，就会给P3^2口
* 产生一个下降沿的脉冲，然后程序会自动跳到以下中断函数中执行一次。
* 由于按键按下去的时候有抖动，也就按一次可能产生几个脉冲，所以按一次往往看到数据一次加了三四个，
* 这种实验现象都是正常的。
*/
void  INT0_int(void) interrupt 0  //INT0外部中断函数
```

```
{
   EX0=0;    //禁止外部0中断 这个只是我个人的编程习惯，也可以不关闭
   uiPluseCnt++;  //累计外部中断下降沿的脉冲数
   ucWd1Update=1;  //窗口1更新显示
   EX0=1;  //打开外部0中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //初始化单片机
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。S1经过跳线后
* 连接到单片机的外部中断专用接口P3^2上，用来模拟外部下降沿脉冲输入。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
```

```
   EX0=1; //允许外部中断0
   IT0=1;  //下降沿触发外部中断0    如果是0代表低电平中断
/* 注释三:
* 注意，由于本系统中用了2个中断，一个是定时中断，一个是外部中断，
* 因此必须设置IP寄存器，让外部中断0为最高优先级，让外部中断0可以打断
* 定时中断。
*/
   IP=0x01; //设置外部中断0为最高优先级，可以打断低优先级中断服务。实现中断嵌套功能
   EA=1;      //开总中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

复制代码

总结陈词:

这节讲了外部中断的基本程序模板，下一节我会讲一个外部中断的实际应用项目例子。欲知详情，请听下回分解----利用外部中断实现模拟串口数据的收发。

（未完待续，下节更精彩，不要走开哦）

第六十七节：利用外部中断实现模拟串口数据的收发。

开场白:

鸿哥曾经亲自用外部中断做过红外遥控器的数据接收，步进电机圆周运动的光电反馈信号检测，输液器里瞬间即逝的水滴信号，以及本节的模拟串口数据的接收，其实这些项目的原理都大同小异，会一样即可触类旁通其它的。

这一节要教大家四个知识点:

第一个：如何利用外部中断实现模拟串口数据的收发。

第二个：在退出外部中断函数时，必须通过软件把外部中断标志位IE0清零，否则在接收到的数据包最后面会多收到一个无效的字节0xFF。

第三个：实际做项目的时候，尽量利用单片机内部自带的集成串口，不到万不得已尽量不要用自制的模拟串口，如果非要用本节讲的模拟串口，那么一次接收的数据包不要太长，尽可能越短越好，因为自己做的模拟串口在稳定性上肯定比不上单片机自带的串口。这种模拟串口在批量生产时容易因为晶振的误差，以及外界各地温度的温差而影响产品的一致性，是有隐患的。

第四个：用模拟串口时，尽量不要选用动态数码管的显示方案，因为单片机在收发串口数据时，只能专心干一件事，此时不能中途被动态数码管扫描程序占用。而动态数码管得不到均匀扫描，就会产生略微闪烁的现象瑕疵。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。当把程序下载到单片机之后，要做以下跳线处理:

单片机原来的P3.1引脚是TI串口输出引脚，P3.0是RI串口输入引脚，分别把P3.1和P3.0的黄颜色跳冒去掉，同时也把外部中断0的引脚P3.2和一根IO口P1.0引脚的换颜色跳冒去掉，把P3.2跳冒的右针连接到P3.0跳冒的左针，作为模拟串口的接收数据线。把P1.0跳冒的右针连接到P3.1跳冒的左针，作为模拟串口的发送数据线。

（2）实现功能:

波特率是：9600 。

通过电脑串口调试助手模拟上位机，往单片机任意发送一串不超过10个的数据包，单片机如实地返回接收到的整包数据给上位机。

例如:

(a)上位机发送数据：01 02 03 04 05 06 07 08 09 0A

单片机返回:　　01 02 03 04 05 06 07 08 09 0A

(b)上位机发送数据:　05 07 EE A8 F9

单片机返回:　　05 07 EE A8 F9

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_rc_size  20  //接收串口中断数据的缓冲区数组大小
```

```
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
/* 注释一:
* 以下时序脉冲延时参数我是在keil uVision2 平台下,Memory Model在small模式，Code Rom Size在Large模式下编译的，
* 如果在不同keil版本，不同的模式下，编译出来的程序有可能此参数会不一样。
* 以下的时序脉冲延时参数是需要一步一步慢慢调的。我一开始的时候先编写一个简单的发送数据测试程序，
* 先确调试出合适的发送时序延时数据。然后再编写串口接收数据的程序，从而调试出接收时序的延时参数。
* 比如: 我第一步发送数据的测试程序是这样的:
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
    {
  //      usart_service();  //串口服务程序
        eusart_send(0x08);    //测试程序，让它不断发送数据给上位机观察，确保发送延时时序的参数准确性
            delay_long(300);
        eusart_send(0xE5);    //测试程序，让它不断发送数据给上位机观察，确保发送延时时序的参数准确性
            delay_long(300);
   }
}
*/
#define const_t_1  10  //发送时序延时1  第一步先调出此数据
#define const_t_2  9  //发送时序延时2   第一步先调出此数据
#define const_r_1  7  //接收时序延时1   第二步再调出此数据
#define const_r_2  9 //接收时序延时2  第二步再调出此数据
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void delay_short(unsigned int uiDelayShort);
void delay_minimum(unsigned char ucDelayMinimum);  //细分度最小的延时,用char类型一个字节
void T0_time(void);  //定时中断函数
void INT0_int(void);  //外部0中断函数，在本系统中是模拟串口的接收中断函数。
void usart_service(void);  //串口服务程序,在main函数里
void eusart_send(unsigned char ucSendData);
unsigned char read_eusart_byte();//从串口读一个字节
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit ti_dr=P1^0;  //模拟串口发送数据的IO口
sbit ri_sr=P3^2;  //模拟串口接收数据的IO口 也是外部中断0的复用IO口
unsigned int  uiSendCnt=0;     //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;    //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned char ucTest=0;
void main()
  {
   initial_myself();
   delay_long(100);
```

```
    initial_peripheral();
    while(1)
    {
        usart_service();  //串口服务程序
    }
}
void usart_service(void)  //串口服务程序,在main函数里
{
     unsigned char i=0;
     if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
     {
            ucSendLock=0;     //处理一次就锁起来，不用每次都进来,除非有新接收的数据
            //下面的代码进入数据协议解析和数据处理的阶段
                        for(i=0;i<uiRcregTotal;i++)  //返回全部接收到的数据包
                        {
                           eusart_send(ucRcregBuf[i]);
                        }


            uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

     }

}
//往串口发送一个字节
void eusart_send(unsigned char ucSendData) //往上位机发送一个字节的函数
{
     unsigned char i=8;
     EA=0;      //关总中断
     ti_dr=0; //发送启始位
         delay_minimum(const_t_1); //发送时序延时1   delay_minimum是本程序细分度最小的延时
     while(i--)
     {
         ti_dr=ucSendData&0x01;       //先传低位
             delay_minimum(const_t_2); //发送时序延时2   delay_minimum是本程序细分度最小的延时
         ucSendData=ucSendData>>1;
     }
     ti_dr=1;  //发送结束位
     delay_short(400);  //每个字节之间的延时，这里非常关键，也是最容易出错的地方。延时的大小请根据实际项
目来调整
     EA=1;      //开总中断
}
//从串口读取一个字节
unsigned char read_eusart_byte()
{
    unsigned char  ucReadData=0;
    unsigned char  i=8;
         delay_minimum(const_r_1);  //接收时序延时1 。作用是等过起始位  delay_minimum是本程序细分度最小
的延时
     while(i--)
```

```
    {
        ucReadData >>=1;
        if(ri_sr==1)
                {
                    ucReadData|=0x80;       //先收低位
                }
        if(ri_sr==0) //此处空指令，是为了让驱动时序的时间保持一致性
                {
                    ;
                }
            delay_minimum(const_r_2);     //接收时序延时2    delay_minimum是本程序细分度最小的延时
    }
    return ucReadData;
}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)   //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
     uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
     ucSendLock=1;      //开自锁标志
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}

void  INT0_int(void) interrupt 0  //INT0外部中断函数
{
   EX0=0;    //禁止外部0中断 这个只是我个人的编程习惯，也可以不关闭
   ++uiRcregTotal;
   if(uiRcregTotal>const_rc_size)  //超过缓冲区
   {
          uiRcregTotal=const_rc_size;
   }
   ucRcregBuf[uiRcregTotal-1]=read_eusart_byte();    //将串口接收到的数据缓存到接收缓冲区里
   uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远也长不大,因为每个中断都被清零。
/* 注释二:
* 注意，此处必须把IE0中断标志清零，否则在接收到的数据包最后面会多收到一个无效的字节0xFF。
*/
   IE0=0;  //外部中断0标志位清零，必须的!
   EX0=1;  //打开外部0中断
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
```

```
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
/* 注释三:
* 由于IO口模拟的串口时序要求很高，所以用的延时函数尽可能细分度越高越好，以下用一个字节的延时计时器
*/
void delay_minimum(unsigned char ucDelayMinimum)  //细分度最小的延时,用char类型一个字节
{
   unsigned char i;
   for(i=0;i<ucDelayMinimum;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EX0=1; //允许外部中断0
   IT0=1;  //下降沿触发外部中断0   如果是0代表低电平中断
   IP=0x01; //设置外部中断0为最高优先级，可以打断低优先级中断服务。实现中断嵌套功能
   EA=1;     //开总中断
   ET0=1;    //允许定时中断
   TR0=1;    //启动定时中断
}
```

复制代码

总结陈词:

这节讲完了外部中断的应用例子，下一节我会开始讲单片机C语言的多文件编程技巧。很多人也把多文件编程称作模块化编程，其实我觉得叫多文件编程会更加符合实际一些。多文件编程有两个最大的好处，一个是给我们的程序增加了目录，方便我们查找。另外一个好处是方便移植别人已经做好的功能程序模块，利用这个特点，特别适合团队一起做大型项目。很多初学者刚开始学多文件编程时，会经常遇到重复定义等问题，想知道怎么解决这些问题吗？欲知详情，请听

下回分解----单片机C语言的多文件编程技巧。
（未完待续，下节更精彩，不要走开哦）

第六十八节：单片机C语言的多文件编程技巧。

开场白：
很多人也把多文件编程称作模块化编程，其实我觉得叫多文件编程会更加符合实际一些。多文件编程有两个最大的好处，一个是给我们的程序增加了目录，方便我们查找。另外一个好处是方便移植别人已经做好的功能程序模块，利用这个特点，特别适合团队一起做大型项目。很多初学者刚开始学多文件编程时，会经常遇到重复定义等问题，想知道怎么解决这些问题吗？只要按照以下鸿哥教的规则来做，这些问题就不存在了。
第一个：每个文件保持成双成对出现。每个.c源文件必须有一个.h头文件跟它对应，每个.h头文件必须有一个.c源文件跟它对应。比如：main.c与main.h，delay.c与 delay.h。
第二个：.c源文件只负责函数的定义和变量的定义，但是不负责函数的声明和变量的声明。比如：

```
unsigned char ucLedStep=0; //这个是全局变量的定义
void led_flicker()   //这个是函数的定义
{
   //…里面是具体代码内容
  }
```

第三个：.h头文件只负责函数的声明和变量的声明，以及常量和IO口的宏定义，但是不负责函数的定义和变量的定义。
比如：

```
#define const_time_level 200  //这个是常量的宏定义
sbit led_dr=P3^5;     //这个是IO口的宏定义
void led_flicker();      //这个是函数的声明
extern unsigned char ucLedStep;    //这个是全局变量的声明，不能赋初始值
```

第四个：每个.h头文件都必须固定以#ifndef，#define，#endif语句为模板，此模板是用来避免编译时由于重复包含头文件里面的内容而导致出错。其中标志变量_XXX_鸿哥建议用它本身的文件名称加前后下划线_。
比如：

```
#ifndef _LED_   //标志变量_LED_是用它本身的文件名称命名
#define _LED_   //标志变量_LED_是用它本身的文件名称命名
#define const_time_level 200  //这个是常量的宏定义
sbit led_dr=P3^5;     //这个是IO口的宏定义
void led_flicker();      //这个是函数的声明
extern unsigned char ucLedStep;    //这个是全局变量的声明，不能赋初始值
#endif
```

第五个：每个.h头文件里都必须声明它对应的.c源文件里的所有定义函数和全局变量，注意：.c源文件里所有的全局变量都要在它所对应的.h头文件里声明一次，不仅仅是函数，这个地方很容易被人忽略。
比如：在led.h头文件中：

```
void led_flicker();      //这个是函数的声明,因为在这个函数在led.c文件里定义了。
   extern unsigned char ucLedStep;    //这个是全局变量的声明，不许赋初值
```

第六个：每个.c源文件里都必须包含两个文件，一个是单片机的系统头文件REG52.H，另外一个是它自己本身的头文件比如initial.h.剩下其它的头文件看实际情况来决定是否调用，我们用到了哪些文件的函数，全局变量或者宏定义，就需要调用对应的头文件。
比如：在initial.c源文件中：

```
#include"REG52.H"  //必须包含的单片机系统头文件
#include"initial.h"  //必须包含它本身的头文件
/* 注释：
   由于本源文件中用到了led_dr的语句，而led_dr是在led.h文件里宏定义的，所以必须把led.h也包含进来
*/
#include"led.h"  //由于本源文件中用到了led_dr的语句，所以必须把led.h也包含进来
void initial_myself()  //这个是函数定义
{
  led_dr=0;  //led_dr是在led文件里定义和声明的
```

}
第七个：声明一个全局变量必须加extern关键字，同时千万不能在声明全局变量的时候赋初始值，比如：
extern unsigned char ucLedStep=0; //这样是绝对错误的。
extern unsigned char ucLedStep; //这个是全局变量的声明，这个才是正确的
第八个：对于函数与全局变量的声明，编译器都不分配内存空间。对于函数与全局变量的定义，编译器都分配内存空间。函数与全局变量的定义只能在一个.c源文件中出现一次，而函数与全局变量的声明可以在多个.h文件中出现。
具体内容，请看源代码讲解，本程序例程是直接把前面第四节一个源文件更改成多文件编程方式。
（1）硬件平台：
基于朱兆祺51单片机学习板。把前面第四节一个源文件更改成多文件编程方式。
（2）实现功能：跟前面第四节的功能一模一样，让一个LED闪烁。

（3）keil多文件编程的截图预览：

第六十九节：使用static关键字可以减少全局变量的使用

开场白：
本来这一节打算开始讲液晶屏的，但是昨天经过网友“任军”的点拨，我发现了一个惊天秘密，原来static关键字是这么好的东西我却错过了那么多年。以前就有一些网友抱怨，鸿哥的程序好是好，就是全局变量满天飞，当时我觉得我也没招呀，C语言就全局变量和局部变量，单单靠局部变量肯定不行，局部变量每次进入函数内部数值都会被初始化改变，所以我在很多场合也只能靠全局变量了。但是自从昨天知道了static关键字的秘密后，我恍然大悟，很多场合只要在局部变量前面加上static关键字，就可以大大减少全局变量的使用了。
这一节要教会大家一个知识点：
大家都知道，普通的局部变量在每次程序执行到函数内部的时候，数值都会被重新初始化，数值会发生变化，不能保持之前的数值。但是在局部变量加上static关键字后，系统在刚上电的时候就已经把带static的局部变量赋初始值了，从此程序每次进入函数内部，都不会初始化带static关键字的局部变量，它会保持最近一次被程序执行更改的数值不变，像全局变量一样。跟全局变量唯一的差别是，带static关键字的局部变量的作用域仅仅在函数内部，而普通全局变量的作用域是整个工程。
本程序例程是直接在第八节程序上修改，大大减少了全局变量的使用。具体内容，请看源代码讲解。
（1）硬件平台：
基于朱兆祺51单片机学习板。用矩阵键盘中的S1和S5号键作为独立按键，记得把输出线P0.4一直输出低电平，模拟独立按键的触发地GND。
（2）实现功能：跟前面第八节的功能一模一样，有两个独立按键，每按一个独立按键，蜂鸣器发出“滴”的一声后就停。

（3）源代码讲解如下：

```
#include "REG52.H"
  #define const_voice_short  40
   #define const_key_time1  20
  #define const_key_time2  20
 void initial_myself();
 void initial_peripheral();
 void delay_long(unsigned int uiDelaylong);
void T0_time();
 void key_service();
  void key_scan();
 sbit key_sr1=P0^0;
 sbit key_sr2=P0^1;
 sbit key_gnd_dr=P0^4;
  sbit beep_dr=P2^7;
  unsigned char ucKeySec=0;    //一些需要在不同函数之间使用的核心变量，只能用全局变量
unsigned int  uiVoiceCnt=0;  //一些需要在不同函数之间使用的核心变量，只能用全局变量
 void main()
```

```
 {
 initial_myself();
   delay_long(100);
  initial_peripheral();
 while(1)
 {
   key_service();
 }
}
 void key_scan()
 {
 /* 注释一:      (1)大家都知道，普通的局部变量在每次程序执行到函数内部的时候，数值都会被重新初始化，
数值会发生变化，不能保持之前的数值。
 (2)但是在局部变量加上static关键字后，系统在刚上电的时候就已经把带static的局部变量     赋初始值了，从此程
序每次进入函数内部，都不会初始化带static关键字的局部变量，它会保持     最近一次被程序执行更改的数值不变
，像全局变量一样。跟全局变量唯一的差别是，带static关键字     的局部变量的作用域仅仅在函数内部，而普通全局
变量的作用域是整个工程。
 (3)以下这些变量我原来在第八节是用普通全局变量的，现在改成用static的局部变量了，减少了全局变量     的使用
，让程序阅读起来更加简洁。大家也可以试试把以下变量的static去掉试试，结果会发现去掉了static后，     按键就
不会被触发了。  */
  static unsigned int  uiKeyTimeCnt1=0; //带static的局部变量
static unsigned char ucKeyLock1=0;
 static unsigned int  uiKeyTimeCnt2=0;
 static unsigned char ucKeyLock2=0;
 if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
 {      ucKeyLock1=0; //按键自锁标志清零           uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧
妙，是我实战中摸索出来的。
   }
else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
 {      uiKeyTimeCnt1++; //累加定时中断次数
 if(uiKeyTimeCnt1>const_key_time1)
 {         uiKeyTimeCnt1=0;
   ucKeyLock1=1;  //自锁按键置位,避免一直触发
 ucKeySec=1;     //触发1号键
  }
 }
if(key_sr2==1)
  {      ucKeyLock2=0;
 uiKeyTimeCnt2=0;   }
else if(ucKeyLock2==0)
 {
  uiKeyTimeCnt2++;
 if(uiKeyTimeCnt2>const_key_time2)
 {
 uiKeyTimeCnt2=0;
 ucKeyLock2=1;
 ucKeySec=2;
 }
}
 }
```

```
void key_service()
{
switch(ucKeySec)
 {
 case 1:   uiVoiceCnt=const_voice_short;
       ucKeySec=0;
       break;
case 2:    uiVoiceCnt=const_voice_short;
       ucKeySec=0;
       break;
 }
 }
void T0_time() interrupt 1
{
 TF0=0;
TR0=0;
key_scan();
if(uiVoiceCnt!=0)
 {
uiVoiceCnt--;
 beep_dr=0;
 }
else
{     ;
beep_dr=1;
}
TH0=0xf8;
TL0=0x2f;
TR0=1;
}
void delay_long(unsigned int uiDelayLong)
 {
unsigned int i;
 unsigned int j;
 for(i=0;i<uiDelayLong;i++)
  {
  for(j=0;j<500;j++)
  {          ;
  }
 }
}
void initial_myself()
 {
key_gnd_dr=0;
 beep_dr=1;
TMOD=0x01;
 TH0=0xf8;
TL0=0x2f;
}
void initial_peripheral()
```

```
{
  EA=1;
ET0=1;
TR0=1;
}
```

第七十节：深入讲解液晶屏的构字过程。

开场白：

液晶屏模块本身带控制芯片，驱动液晶屏的本质就是单片机通过串行或者并行方式，根据芯片资料指定的协议跟液晶芯片进行通讯的过程。这个详细的通讯协议驱动程序厂家都会免费提供的，也可以在网上找到大量的示范程序。那么我们最应该关注的核心是什么？我认为最核心的是要理清楚程序坐标与实际显示坐标之间的关系规律。本程序不使用模块自带的字库，而是使用自己构造的字库，目的就是为了让读者理解更底层的字模显示。

这一节要教会大家三个知识点：

第一个：对于驱动芯片是st7920的12864液晶屏，它的真实坐标体系的本质是256x32的点阵液晶屏。

第二个：鸿哥刻意在驱动显示函数里增加了大延时函数，目的是通过慢镜头回放，让大家观察到横向取模的字是如何一个字节一个字节构建而成的。

第三个：数组带const关键字，表示数据常量存放在ROM程序区，不占用RAM的变量。

具体内容，请看源代码讲解。

（1）硬件平台：

基于朱兆祺51单片机学习板。

（2）实现功能：开机上电后，可以观察到0x01,0x02,0x03,0x04这4个显示数字在不同的排列方式下，出现在不同的液晶屏显示位置。也可以观察到“馒头”这两个字是如何一个字节一个字节构建而成的，加深理解字模数组跟显示现象的关系。

（3）源代码讲解如下：

```
#include "REG52.H"
  sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
 void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
 void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount); //显示任意点阵函数
 void display_clear(void); // 清屏
  void delay_short(unsigned int uiDelayshort); //延时
  /* 注释一:   * 数组带const关键字，表示数据常量存放在ROM程序区，不占用RAM的变量  */
const unsigned char Hz1616_man[]= /*馒   横向取模  16X16点阵 网上有很多免费的字模软件生成字模数组 */
{ 0x21,0xF8,0x21,0x08,0x21,0xF8,0x3D,0x08,0x45,0xF8,0x48,0x00,0x83,
0xFC,0x22,0x94,  0x23,0xFC,0x20,0x00,0x21,0xF8,0x20,0x90,0x28,0x60,0x30,0x90,0x23,0x0E,0x00,0x00, };
 const unsigned char Hz1616_tou[]=
/*头   横向取模  16X16点阵 网上有很多免费的字模软件生成字模数组 */
{ 0x00,0x80,0x10,0x80,0x0C,0x80,0x04,0x80,0x10,0x80,0x0C,0x80,0x08,0x80,0x00,0x80,  0xFF,0xFE,0x00,
0x80,0x01,0x40,0x02,0x20,0x04,0x30,0x08,0x18,0x10,0x0C,0x20,0x08, };
 /* 注释二:   * 为了方便观察字模的数字与显示的关系，以下3个数组的本质是完全一样的，只是排列不一样而已。
*/
const unsigned char Byte_1[]=  //4横，1列
```

```
 { 0x01,0x02,0x03,0x04, };
 const unsigned char Byte_2[]=
//2横，2列
{ 0x01,0x02, 0x03,0x04, };
 const unsigned char Byte_3[]=
//1横，4列
{ 0x01, 0x02, 0x03, 0x04, };
 void main()
 {
    LCDInit(); //初始化12864 内部包含液晶模块的复位
display_clear(); // 清屏
display_lattice(0,0,Byte_1,0,4,1);     //显示<4横，1列>的数组数字
display_lattice(0,16,Byte_1,1,4,1);    //显示<4横，1列>的数组数字 反显
display_lattice(7,0,Byte_2,0,2,2);    //显示<2横，2列>的数组数字
display_lattice(7,16,Byte_2,1,2,2);   //显示<2横，2列>的数组数字 反显
display_lattice(8,0,Byte_3,0,1,4);   //显示<1横，4列>的数组数字
display_lattice(8,16,Byte_3,1,1,4); //显示<1横，4列>的数组数字 反显
display_lattice(14,0,Hz1616_man,0,2,16);   //显示<馒>字
display_lattice(15,0,Hz1616_tou,0,2,16);   //显示<头>字
display_lattice(14,16,Hz1616_man,1,2,16); //显示<馒>字 反显
display_lattice(15,16,Hz1616_tou,1,2,16); //显示<头>字 反显
while(1)
{
    ;
 }
}    /* 注释三: 真实坐标体系的本质。   * 从坐标体系的角度来看，本液晶屏表面上是128x64的液晶屏，实际上可以看做是256x32的液晶屏。   * 把256x32的液晶屏分左右两半，把左半屏128x32放在上面，把右半屏128x32放下面，就合并成了   * 一个128x64的液晶屏。由于液晶模块内部控制器的原因，虽然横向有256个点阵，但是我们的x轴   * 坐标没办法精确到每个点，只能以16个点(2个字节)为一个单位，因此256个点的x轴坐标范围是0至15。   * 而y轴的坐标可以精确到每个点为一行，所以32个点的y轴坐标范围是0至31.   */
 void display_clear(void) // 清屏
 {
unsigned char x,y;    //
WriteCommand(0x34);   //关显示缓冲指令
WriteCommand(0x36); //这次为了观察每个数字在显示屏上的关系，所以把这个显示缓冲的命令提前打开，下一节放到本函数最后
y=0;      while(y<32)   //y轴的范围0至31
{
WriteCommand(y+0x80);          //垂直地址
WriteCommand(0x80);            //水平地址
for(x=0;x<32;x++)   //256个横向点，有32个字节
{
LCDWriteData(0x00);
 }
 y++;
 }
}       /* 注释四: 本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。   * 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.   * 第3个参数*ucArray是字模的数组。   * 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。   * 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。   * 本函数后面故意增加一个长延时delay_short(30000)，是为了方便读者观察横向取模的   * 字是
```

```
如何一个字节一个字节构建而成的。  */
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
unsigned int j=0;
unsigned int i=0;
unsigned char ucTemp;  //
WriteCommand(0x34);   //关显示缓冲指令
WriteCommand(0x36);  //这次为了观察每个数字在显示屏上的关系，所以把这个显示缓冲的命令提前打开，下一节放
到本函数最后
for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
{        WriteCommand(y+j+0x80);         //垂直地址
WriteCommand(x+0x80);           //水平地址
for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
{          ucTemp=ucArray[j*x_amount+i];
if(ucFbFlag==1)  //反白显示
{
ucTemp=~ucTemp;
}
LCDWriteData(ucTemp);
delay_short(30000);  //本函数故意增加这个长延时，是为了方便读者观察横向取模的字是如何一个字节一个字节构
建而成的。
}
}
}   /* 注释五:   * 以下是液晶屏模块的驱动程序，我觉得没有什么好讲的，因为我是直接在网上寻找现成的驱动时
序修改而成。
       * 它的本质就是单片机跟这个液晶模块芯片进行串行通信。  */
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
 {
unsigned char i;
for ( i = 0; i < 8; i++ )
{
if ( (ucData << i) & 0x80 )
{
 LCDSID_dr = 1;
 }
else
 {
 LCDSID_dr = 0;
}
LCDCLK_dr = 0;
 LCDCLK_dr = 1;
}
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
{
SendByteToLcd( 0xf8 + (ucWRS << 1) );
SendByteToLcd( ucWData & 0xf0 );
 SendByteToLcd( (ucWData << 4) & 0xf0);
```

```
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
LCDCS_dr = 0;
LCDCS_dr = 1;
SPIWrite(ucCommand, 0);
delay_short(90);  }
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
LCDCS_dr = 0;
LCDCS_dr = 1;
SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
LCDRST_dr = 1;  //复位
 LCDRST_dr = 0;
LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
 {
unsigned int i;
 for(i=0;i<uiDelayShort;i++)
 {
;
}
}
```

总结陈词:

这节重点讲了液晶屏的构字过程，下节将会在本节的基础上，略作修改，显示常用的不同点阵字模。欲知详情，请听下回分解-----液晶屏的字符，16点阵，24点阵和32点阵的显示程序。

（未完待续，下节更精彩，不要走开哦）

第七十一节：液晶屏的字符，16点阵，24点阵和32点阵的显示程序。

开场白:

这一节要教会大家二个知识点:

第一个：如何利用任意点阵字体显示函数display_lattice来显示8x16的字符，16点阵汉字，24点阵汉字和32点阵汉字。

第二个：纠正上一节的一个小错误。C51编译器跟其它单片机的编译器有点不一样。想把常量数据保存在ROM程序存储区里并不是用const关键字，而是是用code关键字。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能：开机上电后，可以看到液晶屏分别显示32点阵，24点阵和16点阵的“馒头”两个字，还有“V5”这两个8x16点阵的字符。

（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
```

```
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount); //显示任意点阵函数
void display_clear(void); // 清屏
void delay_short(unsigned int uiDelayshort); //延时
/* 注释一:
* 纠正上一节的一个小错误。C51编译器跟其它的编译器有点不一样。
* 存在ROM程序存储区里的常量数据并不是用const关键字，而是是用code关键字。
*/
code unsigned char Hz3232_man[]= /*馒   横向取模  32x32点阵 */
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x07,0x03,0x00,0x0F,0x87,0xFF,0x80,
0x0F,0x07,0x03,0x80,0x0E,0x07,0x03,0x80,0x0E,0x37,0xFF,0x80,0x1C,0x7F,0x03,0x80,
0x1F,0xFF,0x03,0x80,0x18,0x77,0xFF,0x00,0x38,0xE0,0x00,0xC0,0x36,0xDF,0xFF,0xF0,
0x77,0x9C,0xCE,0xE0,0x67,0x1C,0xCE,0xE0,0xC7,0x1C,0xCE,0xE0,0x07,0x1C,0xCE,0xE0,
0x07,0x1F,0xFF,0xE0,0x07,0x18,0x00,0x00,0x07,0x00,0x03,0x80,0x07,0x0F,0xFF,0xC0,
0x07,0x71,0x8F,0x00,0x07,0xE0,0xDE,0x00,0x07,0xC0,0xFC,0x00,0x07,0x80,0x78,0x00,
0x0F,0x01,0xFE,0x00,0x07,0x03,0x8F,0xE0,0x00,0x1E,0x03,0xF0,0x00,0xF8,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz3232_tou[]= /*头   横向取模  32x32点阵 */
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0xC0,0x00,0x00,0x03,0xE0,0x00,
0x03,0xC3,0xC0,0x00,0x00,0xF3,0x80,0x00,0x00,0x7B,0x80,0x00,0x00,0x7B,0x80,0x00,
0x00,0x3B,0x80,0x00,0x0E,0x03,0x80,0x00,0x07,0x83,0x80,0x00,0x03,0xC3,0x80,0x00,
0x01,0xE3,0x80,0x00,0x01,0xE3,0x80,0x00,0x00,0xC3,0x80,0x00,0x00,0x03,0x81,0xE0,
0x7F,0xFF,0xFF,0xF0,0x00,0x07,0x80,0x30,0x00,0x07,0x00,0x00,0x00,0x07,0x80,0x00,
0x00,0x0E,0xE0,0x00,0x00,0x1E,0x7C,0x00,0x00,0x3C,0x1F,0x00,0x00,0x78,0x0F,0xC0,
0x00,0xF0,0x03,0xC0,0x03,0xC0,0x01,0xE0,0x0F,0x00,0x00,0xE0,0x78,0x00,0x00,0x00,
0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz2424_man[]= /*馒   横向取模  24x24点阵 */
{
0x00,0x00,0x00,0x00,0x00,0x00,0x0C,0x18,0x30,0x1E,0x1F,0xF8,0x1C,0x1C,0x38,0x1C,
0x1F,0xF8,0x19,0xFC,0x38,0x3F,0xFF,0xF8,0x31,0x98,0x30,0x7B,0xE0,0x0E,0x6F,0x7F,
0xFE,0x6E,0x76,0xEE,0xCC,0x76,0xEE,0x0C,0x7F,0xFE,0x0C,0x70,0x0C,0x0C,0x00,0x38,
0x0C,0x3F,0xF8,0x0D,0xCE,0x70,0x0F,0x87,0xE0,0x0F,0x03,0x80,0x1E,0x07,0xE0,0x0C,
0x1C,0x7E,0x01,0xF0,0x1F,0x00,0x00,0x00,
};
code unsigned char Hz2424_tou[]= /*头   横向取模  24x24点阵 */
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x00,0x06,0x0F,0x00,0x07,0x8E,0x00,0x01,
0xEE,0x00,0x00,0xEE,0x00,0x00,0xEC,0x00,0x1C,0x0C,0x00,0x0F,0x0C,0x00,0x07,0x9C,
0x00,0x03,0x9C,0x00,0x00,0x1C,0x0C,0x00,0x1C,0x1E,0x7F,0xFF,0xF6,0x00,0x1C,0x00,
```

```
0x00,0x3C,0x00,0x00,0x3F,0x80,0x00,0x71,0xE0,0x00,0xE0,0xF8,0x01,0xC0,0x3C,0x07,
0x00,0x1C,0x3C,0x00,0x0C,0x70,0x00,0x00,
};
code unsigned char Hz1616_man[]= /*馒   横向取模  16X16点阵 */
{
0x21,0xF8,0x21,0x08,0x21,0xF8,0x3D,0x08,0x45,0xF8,0x48,0x00,0x83,0xFC,0x22,0x94,
0x23,0xFC,0x20,0x00,0x21,0xF8,0x20,0x90,0x28,0x60,0x30,0x90,0x23,0x0E,0x00,0x00,
};
code unsigned char Hz1616_tou[]= /*头   横向取模  16X16点阵 */
{
0x00,0x80,0x10,0x80,0x0C,0x80,0x04,0x80,0x10,0x80,0x0C,0x80,0x08,0x80,0x00,0x80,
0xFF,0xFE,0x00,0x80,0x01,0x40,0x02,0x20,0x04,0x30,0x08,0x18,0x10,0x0C,0x20,0x08,
};
code unsigned char Zf816_V[]= /*V   横向取模  8x16点阵 */
{
0x00,0x00,0x00,0xE7,0x42,0x42,0x44,0x24,0x24,0x28,0x28,0x18,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_5[]= /*5   横向取模  8x16点阵 */
{
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
void main()
  {
       LCDInit(); //初始化12864 内部包含液晶模块的复位
    display_clear(); // 清屏
       display_lattice(0,0,Hz3232_man,0,4,32);  //显示32点阵的<馒>字
       display_lattice(2,0,Hz3232_tou,0,4,32);  //显示32点阵的<头>字
       display_lattice(4,0,Hz2424_man,0,3,24);  //显示24点阵的<馒>字
       display_lattice(6,0,Hz2424_tou,0,3,24);  //显示24点阵的<头>字
       display_lattice(8,0,Hz1616_man,0,2,16);  //显示16点阵的<馒>字
       display_lattice(9,0,Hz1616_tou,0,2,16);  //显示16点阵的<头>字
       display_lattice(11,0,Zf816_V,0,1,16);  //显示8x16点阵的<V>字符
       display_lattice(12,0,Zf816_5,0,1,16);  //显示8x16点阵的<5>字符
    while(1)
    {
      ;
    }
}
void display_clear(void) // 清屏
{
       unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
        y=0;
       while(y<32)  //y轴的范围0至31
    {
                WriteCommand(y+0x80);         //垂直地址
        WriteCommand(0x80);           //水平地址
        for(x=0;x<32;x++)  //256个横向点，有32个字节
```

```
         {
            LCDWriteData(0x00);
             }
                 y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释二: 本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   WriteCommand(0x34);  //关显示缓冲指令
   WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
         ucTemp=ucArray[j*x_amount+i];
            if(ucFbFlag==1)  //反白显示
                {
            ucTemp=~ucTemp;
         }
            LCDWriteData(ucTemp);
          //         delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
```

```
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
```

总结陈词:

我们现在讲的字体显示都是横向的，如果某个项目要把整个液晶屏顺时针旋转90度，要求像对联一样纵向显示一串字体的时候，该怎么办？我前两个月就遇到了这样的项目，当时我的做法就是把字体的字库数组通过算法旋转90度就达到了目的。这种算法程序是怎样编写的？欲知详情，请听下回分解-----把字体顺时针旋转90度显示的算法程序。

第七十二节：在液晶屏中把字体顺时针旋转90度显示的算法程序。

开场白:

我曾经遇到过这样的项目，客户由于外壳结果的原因，故意把液晶屏物理位置逆时针旋转了90度，在这种情况下，如果按之前的显示驱动就会发现字体也跟着倒了过来，影响了阅读。当时我的解决办法就是把字体的字库数组通过算法顺时针旋转90度就达到了目的。这一节把这个算法教给大家。

这个算法的本质是：请看以下附图1，附图2，附图3.

第一步：旋转90度的本质，就是把原来横向取模改成纵向去模。先把代表每一行16个点阵数的2个char型数据合并成1个int型数据。
第二步：再把每一列的16个点阵按2个字节分别取到一个数组里，就是纵向取模的过程了。
具体内容，请看源代码讲解。

（1）硬件平台:
　　基于朱兆祺51单片机学习板。
（2）实现功能：把液晶屏物理位置逆时针旋转了90度，开机上电后，可以看到液晶屏像对联的显示顺序一样，从上往下分别显示“馒头V5”四个字。

（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount); //显示任意点阵函数
void display_clear(void); // 清屏
void hz1616_s90(const unsigned char  *p_ucHz,unsigned char *p_ucResult);  //把16x16汉字字模顺时针旋转90度的转换函数
void hz816_s90(const unsigned char  *p_ucHz,unsigned char *p_ucResult);  //把8x16字符字模顺时针旋转90度的转换函数
void delay_short(unsigned int uiDelayshort); //延时
code unsigned char Hz1616_man[]= /*馒   横向取模  16X16点阵 */
{
0x21,0xF8,0x21,0x08,0x21,0xF8,0x3D,0x08,0x45,0xF8,0x48,0x00,0x83,0xFC,0x22,0x94,
0x23,0xFC,0x20,0x00,0x21,0xF8,0x20,0x90,0x28,0x60,0x30,0x90,0x23,0x0E,0x00,0x00,
};
code unsigned char Hz1616_tou[]= /*头   横向取模  16X16点阵 */
{
0x00,0x80,0x10,0x80,0x0C,0x80,0x04,0x80,0x10,0x80,0x0C,0x80,0x08,0x80,0x00,0x80,
0xFF,0xFE,0x00,0x80,0x01,0x40,0x02,0x20,0x04,0x30,0x08,0x18,0x10,0x0C,0x20,0x08,
};
code unsigned char Zf816_V[]= /*V   横向取模  8x16点阵 */
{
0x00,0x00,0x00,0xE7,0x42,0x42,0x44,0x24,0x24,0x28,0x28,0x18,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_5[]= /*5   横向取模  8x16点阵 */
{
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
```

```
unsigned char ucBufferResult[32]; //用于临时存放转换结束后的字模数组
void main()
  {
        LCDInit(); //初始化12864 内部包含液晶模块的复位
        display_clear(); // 清屏
/* 注释一:
* (1)把原来的液晶屏物理位置逆时针旋转90度后，从上往下阅读，类似对联的阅读习惯。所以请注意坐标体系参数的变化。
* (2)为了让字符居中显示，请注意在显示V和5两个字符时坐标体系的变化。
* (3)字符8x16经过旋转处理后，变成了16x8，在调用display_lattice函数时，要注意修改响应的参数。
*/
        hz1616_s90(Hz1616_man,ucBufferResult);  //把<馒>字顺时针旋转90度放到ucBufferResult临时变量里。
        display_lattice(7,0,ucBufferResult,0,2,16);  //显示旋转90度后的<馒>字
        hz1616_s90(Hz1616_tou,ucBufferResult);  //把<头>字顺时针旋转90度放到ucBufferResult临时变量里。
        display_lattice(6,0,ucBufferResult,0,2,16);  //显示旋转90度后的<头>字
        hz816_s90(Zf816_V,ucBufferResult);  //把<V>字符顺时针旋转90度放到ucBufferResult临时变量里。
        display_lattice(5,4,ucBufferResult,0,2,8);  //显示旋转90度后的<V>字符。注意在最后两个个参数，2表示每一行有2个字节，8表示8列。第二个坐标参数4是为了偏移居中显示。
        hz816_s90(Zf816_5,ucBufferResult);  //把<5>字符顺时针旋转90度放到ucBufferResult临时变量里。
        display_lattice(4,4,ucBufferResult,0,2,8);  //显示旋转90度后的<5>字符。注意在最后两个个参数，2表示每一行有2个字节，8表示8列。第二个坐标参数4是为了偏移居中显示。
    while(1)
    {
       ;
    }
}
void display_clear(void) // 清屏
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);          //垂直地址
         WriteCommand(0x80);            //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(0x00);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释二:
* 把16x16汉字字模顺时针旋转90度的步骤: 请看附图1，附图2，附图3.
* 第一步: 旋转90度的本质，就是把原来横向取模改成纵向去模。先把代表每一行16个点阵数的2个char型数据合并成1个int型数据。
* 第二步: 再把每一列的16个点阵按2个字节分别取到一个数组里，就是纵向取模的过程了。以下程序int型数据每取
```

8个数据的最高位，
* 就左移一次，本质就是纵向取模的过程。
*/
void hz1616_s90(const unsigned char *p_ucHz,unsigned char *p_ucResult) //把16x16汉字字模顺时针旋转90度的转换函数
{
        unsigned char a;
        unsigned char b;
        unsigned char c;
    unsigned int uiBuffer[16]; //注意，是int类型数据，一个数据包含2个字节。

        for(a=0;a<16;a++) //把原来以字节为单位的字库每一行的2个字节合并成1个int型数据。放到一个包含16个int类型的数组里，为旋转90度算法处理做准备
        {
        uiBuffer[a]=p_ucHz[a*2];
                uiBuffer[a]=uiBuffer[a]<<8;
                uiBuffer[a]=uiBuffer[a]+p_ucHz[a*2+1];
    }

        c=0;
        for(a=0;a<16;a++) //这里的16代表16列
        {
                for(b=0;b<8;b++)   //每一列中有16个点，有2个字节，这里的8代表第一个字节的8个位或点。
                {
                        p_ucResult[c]=p_ucResult[c]<<1;
            p_ucResult[c]=p_ucResult[c]&0xfe;
            if(uiBuffer[15-b]>=0x8000)   //注意，int类型数据的判断是0x8000,char型的是0x80
                    {
               p_ucResult[c]=p_ucResult[c]+1;
            }
                    uiBuffer[15-b]=uiBuffer[15-b]<<1;
        }
                c++;

                for(b=0;b<8;b++) //每一列中有16个点，有2个字节，这里的8代表第二个字节的8个位或点。
                {
                        p_ucResult[c]=p_ucResult[c]<<1;
            p_ucResult[c]=p_ucResult[c]&0xfe;
            if(uiBuffer[7-b]>=0x8000)
                    {
              p_ucResult[c]=p_ucResult[c]+1;
            }
                        uiBuffer[7-b]=uiBuffer[7-b]<<1;
        }
                c++;
    }

}
/* 注释三:
* 把8x16字符字模顺时针旋转90度的步骤:

```
* 第一步: 旋转90度的本质，就是把原来横向取模改成纵向去模。由于原来的字库存放在带code关键字的ROM区，只能读不能写，所以
* 先把原来的字模数组读取出来，放到一个变量缓冲区里。
* 第二步: 再把每一列的16个点阵按2个字节分别取到一个数组里，就是纵向取模的过程了。以下程序int型数据每取8个数据的最高位，
   就左移一次，本质就是纵向取模的过程。
*/
void hz816_s90(const unsigned char  *p_ucHz,unsigned char *p_ucResult)  //把8x16字符字模顺时针旋转90度的转换函数
{
        unsigned char a;
        unsigned char b;
        unsigned char c;
    unsigned char uiBuffer[16]; //注意，跟16x16点阵不一样，这里是char数据。因为横向的只有8个点

        for(a=0;a<16;a++) //把存放在ROM的字库放到一个16个char类型的数组里
        {
        uiBuffer[a]=p_ucHz[a];
    }

        c=0;
        for(a=0;a<8;a++)  //这里的8代表8列
        {
                for(b=0;b<8;b++)  //每一列中有16个点，有2个字节，这里的8代表第一个字节的8个位或点。
                {
                        p_ucResult[c]=p_ucResult[c]<<1;
            p_ucResult[c]=p_ucResult[c]&0xfe;
            if(uiBuffer[15-b]>=0x80)   //注意，int类型数据的判断是0x8000,char型的是0x80
                      {
                 p_ucResult[c]=p_ucResult[c]+1;
            }
                    uiBuffer[15-b]=uiBuffer[15-b]<<1;
        }
              c++;

                for(b=0;b<8;b++)  //每一列中有16个点，有2个字节，这里的8代表第二个字节的8个位或点。
                {
                        p_ucResult[c]=p_ucResult[c]<<1;
            p_ucResult[c]=p_ucResult[c]&0xfe;
            if(uiBuffer[7-b]>=0x80)   //注意，int类型数据的判断是0x8000,char型的是0x80
                        {
                p_ucResult[c]=p_ucResult[c]+1;
            }
                        uiBuffer[7-b]=uiBuffer[7-b]<<1;
        }
                c++;
    }

}
/* 注释四: 本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
```

```
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   WriteCommand(0x34);  //关显示缓冲指令
   WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
         ucTemp=ucArray[j*x_amount+i];
             if(ucFbFlag==1)  //反白显示
                 {
            ucTemp=~ucTemp;
        }
             LCDWriteData(ucTemp);
          //         delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
```

```
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
```

复制代码

总结陈词:

有的项目会要求把字体或者图像进行镜像显示处理，这种算法程序是怎样编写的？欲知详情，请听下回分解-----在液晶屏中把字体镜像显示的算法程序。

第七十三节：在液晶屏中把字体镜像显示的算法程序。

开场白:

有的项目会要求把字体或者图像进行镜像显示处理，这一节把这个算法教给大家。

这个算法的本质是:

16x16点阵的图像或者字体有16行，每行有2个字节，如果把这2个字节看成是一个16位int型数据，那么就是要这个数据从原来左边是高位，右边是低位的顺序颠倒过来。本程序没有把2个字节合并成一个int型数据，而是直接在一个字节数据内把高低位顺序颠倒过来，然后把第1字节数据跟第2字节数据交换。

8x16点阵的图像或者字体有16行，每行有1个字节，把这个数据从原来左边是高位，右边是低位的顺序颠倒过来。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能：开机上电后，从上往下分别显示“馒头V5”四个字以及右边镜像后的“馒头V5”四个字。

（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount); //显示任意点阵函数
void display_clear(void); // 清屏
void hz1616_mirror(const unsigned char  *p_ucHz,unsigned char *p_ucResult);  //把16x16点阵字库镜像
void hz816_mirror(const unsigned char  *p_ucHz,unsigned char *p_ucResult);  //把8x16点阵字库镜像
void delay_short(unsigned int uiDelayshort); //延时
code unsigned char Hz1616_man[]= /*馒   横向取模  16X16点阵 */
{
0x21,0xF8,0x21,0x08,0x21,0xF8,0x3D,0x08,0x45,0xF8,0x48,0x00,0x83,0xFC,0x22,0x94,
0x23,0xFC,0x20,0x00,0x21,0xF8,0x20,0x90,0x28,0x60,0x30,0x90,0x23,0x0E,0x00,0x00,
};
code unsigned char Hz1616_tou[]= /*头   横向取模  16X16点阵 */
{
0x00,0x80,0x10,0x80,0x0C,0x80,0x04,0x80,0x10,0x80,0x0C,0x80,0x08,0x80,0x00,0x80,
0xFF,0xFE,0x00,0x80,0x01,0x40,0x02,0x20,0x04,0x30,0x08,0x18,0x10,0x0C,0x20,0x08,
};
code unsigned char Zf816_V[]= /*V   横向取模  8x16点阵 */
{
0x00,0x00,0x00,0xE7,0x42,0x42,0x44,0x24,0x24,0x28,0x28,0x18,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_5[]= /*5   横向取模  8x16点阵 */
{
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
unsigned char ucBufferResult[32]; //用于临时存放转换结束后的字模数组
void main()
  {
        LCDInit(); //初始化12864 内部包含液晶模块的复位
        display_clear(); // 清屏
        display_lattice(0,0,Hz1616_man,0,2,16);  //显示镜像前的<馒>字
        hz1616_mirror(Hz1616_man,ucBufferResult);  //把<馒>字镜像后放到ucBufferResult临时变量里。
        display_lattice(1,0,ucBufferResult,0,2,16);  //显示镜像后的<馒>字
        display_lattice(0,16,Hz1616_tou,0,2,16);  //显示镜像前的<头>字
        hz1616_mirror(Hz1616_tou,ucBufferResult);  //把<头>字镜像后放到ucBufferResult临时变量里。
        display_lattice(1,16,ucBufferResult,0,2,16);  //显示镜像后的<头>字
        display_lattice(8,0,Zf816_V,0,1,16);  //显示镜像前的<V>字符
        hz816_mirror(Zf816_V,ucBufferResult);  //把<V>字符镜像后放到ucBufferResult临时变量里。
        display_lattice(9,0,ucBufferResult,0,1,16);  //显示镜像后的<V>字符
        display_lattice(8,16,Zf816_5,0,1,16);  //显示镜像前的<5>字符
```

```
        hz816_mirror(Zf816_5,ucBufferResult);  //把<5>字符镜像后放到ucBufferResult临时变量里。
        display_lattice(9,16,ucBufferResult,0,1,16);  //显示镜像后的<5>字符
        while(1)
        {
             ;
        }
}
void display_clear(void) // 清屏
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);          //垂直地址
         WriteCommand(0x80);            //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(0x00);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释一:
* 16x16点阵镜像的本质:
* 16x16点阵有16行，每行有2个字节，如果把这2个字节看成是一个16位int型数据，
* 那么就是要这个数据从原来左边是高位，右边是低位的顺序颠倒过来。本程序没有把2个字节
* 合并成一个int型数据，而是直接在一个字节数据内把高低位顺序颠倒过来，然后把第1字节数据跟第2字节数据交换。
*/
void hz1616_mirror(const unsigned char  *p_ucHz,unsigned char *p_ucResult)  //把16x16点阵字库镜像的函数
{
         unsigned char a;
         unsigned char b;
         unsigned char c;
         unsigned char d;

         for(a=0;a<16;a++) //这里16代表有16行。每一行有2个字节。把每一个字节看做一列，这里先把第1列字节的数据从原来左边是高位，右边是低位的顺序颠倒过来，相当于镜像。
         {
            b=p_ucHz[a*2+0];  //这里的2代表16x16点阵每行有2列字节，0代表从第1列开始。
            c=0;
            for(d=0;d<8;d++)  //把一个字节调换顺序
                {
               c=c>>1;
           if((b&0x80)==0x80)
                    {
```

```
            c=c|0x80;
          }
                b=b<<1;
        }
        p_ucResult[a*2+1]=c;    //注意，因为是镜像，所以要把颠倒顺序后的字节从原来是第1列的调换到第2列

    }

        for(a=0;a<16;a++)  //这里16代表有16行。每一行有2个字节。把每一个字节看做一列，这里先把第2列字节的数据从原来左边是高位，右边是低位的顺序颠倒过来，相当于镜像。
        {
           b=p_ucHz[a*2+1];    //这里的2代表16x16点阵每行有2列字节，1代表从第2列开始。

           c=0;
           for(d=0;d<8;d++)  //把一个字节调换顺序
               {
                       c=c>>1;
           if((b&0x80)==0x80)
                       {
             c=c|0x80;
           }
                       b=b<<1;
        }
        p_ucResult[a*2+0]=c;          //注意，因为是镜像，所以要把颠倒顺序后的字节从原来是第2列的调换到第1列

    }

}
/* 注释二:
* 8x16点阵镜像的本质:
* 8x16点阵有16行，每行有1个字节，把这个数据从原来左边是高位，右边是低位的顺序颠倒过来。
*/
void hz816_mirror(const unsigned char  *p_ucHz,unsigned char *p_ucResult)  //把8x16点阵字库镜像的函数
{
        unsigned char a;
        unsigned char b;
        unsigned char c;
        unsigned char d;

        for(a=0;a<16;a++) //这里16代表有16行。每一行有1个字节。这里先把每一行字节的数据从原来左边是高位，右边是低位的顺序颠倒过来，相当于镜像。
        {
           b=p_ucHz[a*1+0];  //这里的1代表8x16点阵每行有1列字节，0代表从第1列开始。
           c=0;
           for(d=0;d<8;d++)  //把一个字节调换顺序
               {
              c=c>>1;
          if((b&0x80)==0x80)
                   {
```

```
            c=c|0x80;
          }
                b=b<<1;
        }
        p_ucResult[a*1+0]=c;    //注意，因为每一行只有一列，所以不用像16x16点阵那样把第1列跟第2列对调交换
。
     }

}
/* 注释三：本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   WriteCommand(0x34);  //关显示缓冲指令
   WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
         ucTemp=ucArray[j*x_amount+i];
             if(ucFbFlag==1)  //反白显示
                 {
            ucTemp=~ucTemp;
        }
             LCDWriteData(ucTemp);
          //         delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
```

```
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
```

复制代码

总结陈词:

细心的网友一定会发现，这种12864液晶屏普遍有个毛病，在坐标轴x,y方向上不能完全做到以一个点阵为单位进行随心所欲的显示，比如横向的至少是一个字节8个点阵为单位，而第1,2行跟第3,4行又做不到无缝对接显示，假如我要把汉字一半显示在第2行一半显示在第3行，行不行？当然可以。但是需要我们编写额外的算法程序。这种算法程序是怎样编写的？欲知详情，请听下回分解-----在液晶屏中让字体可以跨区域无缝对接显示的算法程序。

（未完待续，下节更精彩，不要走开哦）

第七十四节：在液晶屏中让字体可以跨区域无缝对接显示的算法程序。

开场白:
细心的网友会发现，这种12864液晶屏在显示自造字库时普遍有个毛病，在坐标轴x方向上是以每16个点阵为一个单位的，如果显示两个8x16字符”V”和”5”，虽然它们的x坐标轴是相邻的，但是实际显示的效果是中间隔了8个点阵。另外，这种12864液晶屏是由上半屏和下半屏组成的，软件上的坐标体系并没有做到跟物理的坐标体系一致，需要转换的。
如果我们想把一个整体字符的一半显示在上半屏，另一半显示在下半屏，那怎么办?
这一节就要教给大家这个算法程序:
为了实现跨区域无缝显示，就先在某个区域显示一块画布，我们只要在这块画布数组中插入字模数组，就可以达到跨区域无缝显示的目的。
具体内容，请看源代码讲解。
（1）硬件平台:
　　基于朱兆祺51单片机学习板。
（2）实现功能: 开机上电后，看到液晶屏所有的点阵都显示。正中间露出一小方块空白的32x16点阵画布，从左到右分别显示“V5”两个字符。这两个字符是紧紧挨在一起的，中间并没有8个点阵的空格，同时这两个字符的上半部分显示在上半屏，下半部分显示在下半屏。实现了真正的跨区域无缝对接显示。
（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
void delay_short(unsigned int uiDelayshort); //延时
code unsigned char Zf816_V[]= /*V   横向取模  8x16点阵 每一行只要1个字节，共16行 */
{
0x00,
0x00,
0x00,
0xE7,
0x42,
0x42,
0x44,
0x24,
0x24,
0x28,
0x28,
0x18,
0x10,
0x10,
0x00,
0x00,
```

```
};
code unsigned char Zf816_5[]= /*5   横向取模  8x16点阵 每一行只要1个字节，共16行 */
{
0x00,
0x00,
0x00,
0x7E,
0x40,
0x40,
0x40,
0x58,
0x64,
0x02,
0x02,
0x42,
0x44,
0x38,
0x00,
0x00,
};
/* 注释一:
* 为了实现跨区域无缝显示，就先在某个区域显示一块画布，我们只要在这块画布数组中插入字模数组，
* 就可以达到跨区域无缝显示的目的。根据上几节的介绍，12864液晶屏由上下两半屏组成，以下这块画布
* 显示在上半屏和下半屏之间。横向4个字节，纵向16行。其中上半屏显示8行，下半屏显示8行。注意，这个数组
* 不带code关键字，是全局变量，这样可读可写。画布的横向x坐标范围是0至3，因为画布的横向只要4个字节。
* 画布的纵向y坐标范围是0至15，因为画布的纵向只有16行。
*/
unsigned char ucCanvasBuffer[]= //画布显示数组。注意，这里没有code关键字，是全局变量。初始化全部填充0x00
{
0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
};
void main()
  {
        LCDInit(); //初始化12864 内部包含液晶模块的复位
```

```
        display_clear(0xff); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
        insert_buffer_to_canvas(0,0,Zf816_V,0,1,16);//把<V>的字模插入画布
        insert_buffer_to_canvas(1,0,Zf816_5,0,1,16);//把<5>的字模插入画布
        display_lattice(3,24,ucCanvasBuffer,0,4,8,0);    //显示上半屏的画布,最后的参数0是偏移量
        display_lattice(11,0,ucCanvasBuffer,0,4,8,32);   //显示下半屏的画布,最后的参数32是偏移量
        while(1)
        {
             ;
        }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释二:
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至3，因为画布的横向只要4个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
                  ucTemp=ucArray[j*x_amount+i];
                  if(ucFbFlag==0)
```

```
                {
            ucCanvasBuffer[(y+j)*4+x+i]=ucTemp; //这里的4代表画布每一行只有4个字节
                }
                else
                {
            ucCanvasBuffer[(y+j)*4+x+i]=~ucTemp; //这里的4代表画布每一行只有4个字节
                }
      }
   }
}
/* 注释三:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   WriteCommand(0x34);  //关显示缓冲指令
   WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
```

```
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
```

复制代码

总结陈词:

经过这一节的算法处理后，字符终于可以在x轴上紧紧挨着显示了。也就是把原来x坐标是16个点阵为一个单位，改

成了以8个点阵为一个单位。如果要求以1个点阵为单位显示，那该怎么办？这个还真有点难度，因为横向的最小显示单位就是一个字节8个点，不过鸿哥在下一节中照样有办法实现这个功能。欲知详情，请听下回分解-----在12864液晶屏中让字体以1个点阵为单位进行移动显示的算法程序。

（未完待续，下节更精彩，不要走开哦）

第七十五节：在12864液晶屏中让字体以1个点阵为单位进行移动显示的算法程序。

开场白：

假设有一个固定的四方形透明窗口，在窗口里面放了一张画布，只要想办法让这个画布往右边拖动，那么画布里面的内容就会跟着画布整体往右边移动，这个就是能以1个点阵为单位进行移动显示的本质。同理，这个画布有16行，每行有4个字节，我们只要把每行4个字节看作是一个首尾连接的二进制数据，把每一行的二进制数据每次整体往右边移动一位，就相当于移动一个点阵了。这一节就要把这个算法教给大家。

具体内容，请看源代码讲解。

（1）硬件平台：

基于朱兆祺51单片机学习板。

（2）实现功能：开机上电后，能看到正中间显示的两个字符“V5”整体以1个点阵为单位向右边慢慢移动。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_MoveTime 400  //每移动一位后的延时时间
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
void delay_short(unsigned int uiDelayshort); //延时
void move_service(void); //整体画布移动的应用程序
void lcd_display_service(void); //应用层面的液晶屏显示程序
void move_canvas_to_one_bit(void);  //把画布整体往右边移动一个点阵
void clear_all_canvas(void);  //把画布全部清零
void T0_time(void);  //定时中断函数
code unsigned char Zf816_V[]= /*V   横向取模  8x16点阵 每一行只要1个字节，共16行 */
{
0x00,
0x00,
0x00,
0xE7,
0x42,
0x42,
0x44,
0x24,
0x24,
0x28,
```

```
0x28,
0x18,
0x10,
0x10,
0x00,
0x00,
};
code unsigned char Zf816_5[]= /*5   横向取模  8x16点阵 每一行只要1个字节，共16行 */
{
0x00,
0x00,
0x00,
0x7E,
0x40,
0x40,
0x40,
0x58,
0x64,
0x02,
0x02,
0x42,
0x44,
0x38,
0x00,
0x00,
};
/* 注释一:
* 为了实现跨区域无缝显示，就先在某个区域显示一块画布，我们只要在这块画布数组中插入字模数组，
* 就可以达到跨区域无缝显示的目的。根据上几节的介绍，12864液晶屏由上下两半屏组成，以下这块画布
* 显示在上半屏和下半屏之间。横向4个字节，纵向16行。其中上半屏显示8行，下半屏显示8行。注意，这个数组
* 不带code关键字，是全局变量，这样可读可写。画布的横向x坐标范围是0至3，因为画布的横向只要4个字节。
* 画布的纵向y坐标范围是0至15，因为画布的纵向只有16行。
*/
unsigned char ucCanvasBuffer[]= //画布显示数组。注意，这里没有code关键字，是全局变量。初始化全部填充0x00
{
0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
```

```
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
};
unsigned char ucDisplayUpdate=1;  //更新显示变量
unsigned char ucMoveStepReset=0;  //这个变量是为了方便外部程序初始化应用程序内部后缀为step的步骤变量
unsigned char ucMoveTimeStart=0; //定时器的开关标志  也相当于原子锁或互斥量的功能
unsigned int uiMoveTime=0;  //定时器累计时间
void main()
  {
        LCDInit(); //初始化12864 内部包含液晶模块的复位
        display_clear(0xff); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
        TMOD=0x01;  //设置定时器0为工作方式1
        TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
        TL0=0x2f;
        EA=1;      //开总中断
        ET0=1;     //允许定时中断
        TR0=1;     //启动定时中断
        while(1)
        {
           move_service(); //整体画布移动的应用程序
           lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void move_service(void) //整体画布移动的应用程序
{
   static unsigned char ucMoveStep=0; //运行步骤。前面加关键字static表示上电后这个变量只初始化一次，以后每次进出函数此变量不会重新初始化，保存之前的更改数值不变。
   static unsigned char ucMoveCnt=0; //统计当前已经往左边移动了多少位。关键字static表示此变量上电后只初始化一次，不会每次进入函数都初始化。
   if(ucMoveStepReset==1)  //运行步骤的复位标志，此段代码结构方便外部程序初始化函数内部的步骤变量ucMoveStep
   {
      ucMoveStepReset=0; //及时把复位标志清零。避免一直处于复位的状态、
          ucMoveStep=0; //运行步骤变量被外部程序通过复位标志初始化。
   }
   switch(ucMoveStep)
   {
      case 0:
               clear_all_canvas();  //把画布全部清零
           insert_buffer_to_canvas(0,0,Zf816_V,0,1,16);//把<V>的字模插入画布
           insert_buffer_to_canvas(1,0,Zf816_5,0,1,16);//把<5>的字模插入画布
           ucDisplayUpdate=1; //更新液晶屏显示

                   uiMoveTime=0;  //定时器清零
                   ucMoveTimeStart=1; //开定时器      也相当于原子锁或互斥量的功能
                   ucMoveCnt=0; //统计当前已经往左边移动了多少位
                   ucMoveStep=1; //切换到下一个运行步骤
               break;
      case 1:
               if(uiMoveTime>const_MoveTime)  //延时一定的时间后
```

```
                    {
                            ucMoveTimeStart=0; //关定时器     也相当于原子锁或互斥量的功能
                        uiMoveTime=0;  //定时器清零
                     if(ucMoveCnt<16)
                        {
                           ucMoveCnt++;
                    move_canvas_to_one_bit(); //把画布整体往左边移动一个点阵
                    ucDisplayUpdate=1; //更新液晶屏显示
                                ucMoveTimeStart=1; //开定时器    也相当于原子锁或互斥量的功能
                        }
                        else
                        {
                                ucMoveStep=0; //移动了16个点阵后，返回上一个运行步骤，把字模重新插入画
布
                        }
            }
                break;
    }
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    if(ucDisplayUpdate==1)  //需要更新显示
    {
       ucDisplayUpdate=0;  //及时把标志清零，避免一直处于不断更新的状态。
       display_lattice(3,24,ucCanvasBuffer,0,4,8,0);    //显示上半屏的画布,最后的参数0是偏移量
       display_lattice(11,0,ucCanvasBuffer,0,4,8,32);  //显示下半屏的画布,最后的参数32是偏移量
    }
}
/* 注释二:
* 假设有一个固定的四方形透明窗口，在窗口里面放了一张画布，只要想办法让这个画布
* 往右边拖动，那么画布里面的内容就会跟着画布整体往右边移动，这个就是能以1个点阵为单位进行移动显示的本质
。
* 同理，这个画布有16行，每行有4个字节，我们只要把每行4个字节看作是一个首尾连接的二进制数据，
* 把每一行的二进制数据每次整体往右边移动一位，就相当于移动一个点阵了。
*/
void move_canvas_to_one_bit(void)  //把画布整体往右边移动一个点阵
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucBitH;  //临时保存一个字节中的最高位
   unsigned char ucBitL;  //临时保存一个字节中的最低位
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      ucBitH=0;
          ucBitL=0;
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                  if((ucCanvasBuffer[j*4+i]&0x01)==0x01)  //临时保存一个字节中的最低位
                  {
                     ucBitL=1;
```

```
                }
                else
                {
                   ucBitL=0;
                }
                ucCanvasBuffer[j*4+i]=ucCanvasBuffer[j*4+i]>>1;  //一行中的一个字节右移一位
                if(ucBitH==1)   //原来左边相邻的字节最低位移动到了当前字节的最高位
                {
            ucCanvasBuffer[j*4+i]=ucCanvasBuffer[j*4+i]|0x80; //把最高位补上
                }
          ucBitH=ucBitL;  //把当前的最低位赋值给最高位，为下一个相邻字节做准备。
      }
   }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void T0_time(void) interrupt 1  //定时中断函数
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(ucMoveTimeStart==1) //已经开了定时器  也相当于原子锁或互斥量的功能
  {
      uiMoveTime++; //定时器累加计时开始
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
```

```
        {
           LCDWriteData(ucFillDate);
        }
        y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释三:
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至3，因为画布的横向只要4个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
                  ucTemp=ucArray[j*x_amount+i];
                  if(ucFbFlag==0)
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=ucTemp; //这里的4代表画布每一行只有4个字节
                  }
                  else
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=~ucTemp; //这里的4代表画布每一行只有4个字节
                  }
      }
   }
}
/* 注释四:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
```

```
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //         delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
```

```
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
```

总结陈词:

从下一节开始讲大家关注已久的液晶屏菜单程序。欲知详情，请听下回分解-----在1个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。

（未完待续，下节更精彩，不要走开哦）

乐于分享，勇于质疑!

第七十六节: 如何把一个任意数值的变量显示在液晶屏上。

开场白:

本来这一节打算开始讲液晶屏的菜单程序，但是我担心跳跃太大，恐怕很多初学者跟不上，所以多插入这一节讲讲后面菜单程序中经常用到的基本功能，如何把一个任意数值的变量显示在液晶屏上。我们需要做一个变量转换成字模的函数，以后只要调用这个转换函数就可以了。这一节就要把这个转换函数教给大家。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能: 我们定义一个char型的全局变量，把它默认初始化为218，开机上电后，能看到正中间恰好显示这个全局变量的数值218。大家也可以试着更改它的默认初始值，只要不超过char型最大数值255范围，我们就会看到它上电后显示的就是这个初始值。

（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
```

```
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
```

```
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
/* 注释一:
* 为了实现跨区域无缝显示，就先在某个区域显示一块画布，我们只要在这块画布数组中插入字模数组，
* 就可以达到跨区域无缝显示的目的。根据上几节的介绍，12864液晶屏由上下两半屏组成，以下这块画布
* 显示在上半屏和下半屏之间。横向4个字节，纵向16行。其中上半屏显示8行，下半屏显示8行。注意，这个数组
* 不带code关键字，是全局变量，这样可读可写。画布的横向x坐标范围是0至3，因为画布的横向只要4个字节。
* 画布的纵向y坐标范围是0至15，因为画布的纵向只有16行。
*/
unsigned char ucCanvasBuffer[]= //画布显示数组。注意，这里没有code关键字，是全局变量。初始化全部填充0x00
{
0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,  //下半屏
```

```
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
};
unsigned char ucDisplayUpdate=1;  //更新显示变量
/* 注释二:
* 以下变量就是本程序的任意变量，网友可以自己更改它的大小来测试本程序，不要超过255.
*/
unsigned char ucAnyNumber=218;  //任意变量默认初始化为218。
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
    ;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    display_clear(0xff); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
}
/* 注释三:
* 本程序的核心转换函数。
* 是可以把一位任意数字变量的函数转换成对应的字模，由于字模是数组，所以返回的是指针，代表字模数组的首地址
。
*/
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                    break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                    break;
            case 2:
             p_ucAnyNumber=Zf816_2;
```

```
                break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                break;
            case 10:
             p_ucAnyNumber=Zf816_nc;
                break;
               default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    static unsigned char ucAnyNumber_1; //分解变量的个位
    static unsigned char ucAnyNumber_10; //分解变量的十位
    static unsigned char ucAnyNumber_100; //分解变量的百位
    static unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    static unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
    static unsigned char *p_ucAnyNumber_100; //经过数字转换成字模后，分解变量的百位字模首地址
    if(ucDisplayUpdate==1)  //需要更新显示
    {
       ucDisplayUpdate=0;  //及时把标志清零，避免一直处于不断更新的状态。
           if(ucAnyNumber>=100) //有3位数以上
           {
           ucAnyNumber_100=ucAnyNumber/100; //百位
       }
           else //否则显示空
           {
               ucAnyNumber_100=10;  //在下面的转换函数中，代码10表示空字模
           }
```

```
            if(ucAnyNumber>=10) //有2位数以上
            {
            ucAnyNumber_10=ucAnyNumber%100/10;  //十位
        }
            else //否则显示空
            {
                ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
            }
        ucAnyNumber_1=ucAnyNumber%10/1;  //个位
            p_ucAnyNumber_100=number_to_matrix(ucAnyNumber_100); //把数字转换成字模首地址
            p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
            p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
        clear_all_canvas();  //把画布全部清零
        insert_buffer_to_canvas(0,0,p_ucAnyNumber_100,0,1,16);//把百位的字模插入画布
        insert_buffer_to_canvas(1,0,p_ucAnyNumber_10,0,1,16);//把十的字模插入画布
        insert_buffer_to_canvas(2,0,p_ucAnyNumber_1,0,1,16);//把个的字模插入画布
        display_lattice(3,24,ucCanvasBuffer,0,4,8,0);    //显示上半屏的画布,最后的参数0是偏移量
        display_lattice(11,0,ucCanvasBuffer,0,4,8,32);  //显示下半屏的画布,最后的参数32是偏移量
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                  ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);          //垂直地址
         WriteCommand(0x80);            //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
```

```
}
/* 注释四:
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至3，因为画布的横向只要4个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
                  ucTemp=ucArray[j*x_amount+i];
                  if(ucFbFlag==0)
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=ucTemp; //这里的4代表画布每一行只有4个字节
                  }
                  else
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=~ucTemp; //这里的4代表画布每一行只有4个字节
                  }
      }
   }
}
/* 注释五:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
```

```
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
```

```
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

总结陈词:

有了这一节的基础，我们继续循序渐进，下一节将会讲到液晶屏的菜单程序。欲知详情，请听下回分解-----在1个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。

（未完待续，下节更精彩，不要走开哦）

第七十七节: 在1个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。

开场白:

这一节要教会大家两个知识点:

第一个知识点: 我在前面讲数码管显示的时候就提出了一个 “一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。把每一个窗口的内容分为两种类型，一种类型是那些不用经常刷新显示的内容，只有在切换窗口的时候才需要更新的，这种内容放在整屏更新显示的括号里，比如清屏操作等内容。另外一种是那些经常需要刷新显示的内容，这种内容放在局部更新显示的括号里。

第二个知识点: 按键如何跟液晶屏显示有机的结合起来? 只要遵循鸿哥总结出来的一个规律“在不同的窗口下，根据不同的局部变量来操作不同的参数”，这样再复杂的人机交互程序都会显得很简单清晰。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换“光标”移动按键对应S9键，设置参数按键对应S13键。

（2）实现功能：

通过按键设置4个不同的参数。

有1个窗口。每个窗口显示4个参数。每个参数的范围是从0到99。

有4个按键：

（a）　　一个是设置参数S13按键，按下此按键，液晶屏的第一行会出现反显的光标，表示进入设置参数模式，再次按下此按键，反显光标会消失，表示退出设置参数模式。

（b）　　一个是移动光标S9按键，在进入设置参数的模式下，依次按下此按键，液晶屏上的光标会从上往下移动，表示选中不同的参数。

（c）　　一个是减数S5按键，在设置参数模式下，依次按下此按键，被选中的参数会逐渐减小。

（d）　　一个是加数S1按键，在设置参数模式下，依次按下此按键，被选中的参数会逐渐加大。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void T0_time(); //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
```

```
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
```

```
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_yi[]=
{
/*--  文字:  一  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_er[]=
{
/*--  文字:  二  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x10,0x3F,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_san[]=
{
/*--  文字:  三  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x7F,0xFC,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3F,0xF8,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0xFE,0x00,0x00,0x00,
};
code unsigned char Hz1616_si[]=
{
/*--  文字:  四  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x7F,0xFC,0x44,0x84,0x44,0x84,0x44,0x84,0x44,0x84,0x44,0x84,0x44,0x84,
0x48,0x84,0x48,0x7C,0x50,0x04,0x60,0x04,0x40,0x04,0x7F,0xFC,0x40,0x04,0x00,0x00,
};
code unsigned char Hz1616_chuang[]=
{
```

```
/*--  文字:  窗  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x01,0x00,0x00,0x80,0x7F,0xFE,0x40,0x22,0x09,0x18,0x12,0x06,0x7F,0xF8,0x11,0x08,
0x13,0xE8,0x14,0x48,0x1A,0x88,0x11,0x08,0x12,0x88,0x14,0x08,0x1F,0xF8,0x10,0x08,
};
code unsigned char Hz1616_kou[]=
{
/*--  文字:  口  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x3F,0xF8,0x20,0x08,0x20,0x08,0x20,0x08,0x20,0x08,0x20,0x08,
0x20,0x08,0x20,0x08,0x20,0x08,0x3F,0xF8,0x20,0x08,0x20,0x08,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_hang[]=
{
/*--  文字:  行  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x00,0x1C,0x00,0x31,0xFC,0x40,0x00,0x88,0x00,0x0C,0x00,0x1B,0xFE,0x30,0x20,
0x50,0x20,0x90,0x20,0x10,0x20,0x10,0x20,0x10,0x20,0x10,0x20,0x10,0xA0,0x10,0x40,
};
unsigned char ucCanvasBuffer[]= //画布显示数组。注意，这里没有code关键字，是全局变量。初始化全部填充0x00
{
0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
};
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucPart=0; //局部变量 0代表没有选中任何一行，其它数值1到4代表选中某一行
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量       1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part2Update=0; //窗口1的第2行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part3Update=0; //窗口1的第3行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part4Update=0; //窗口1的第4行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucData_1_1=8;  //第1个窗口第1行的被设置数据
unsigned char ucData_1_2=9;  //第1个窗口第2行的被设置数据
```

```
unsigned char ucData_1_3=10;  //第1个窗口第3行的被设置数据
unsigned char ucData_1_4=11;  //第1个窗口第4行的被设置数据
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                    key_service(); //按键服务的应用程序
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
/* 注释一:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
   key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;     //开总中断
    ET0=1;    //允许定时中断
    TR0=1;    //启动定时中断
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
```

```
  TR0=1;  //开中断
}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  static unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
```

```
    if(uiKeyTimeCnt3>const_key_time3)
    {
       uiKeyTimeCnt3=0;
       ucKeyLock3=1;  //自锁按键置位,避免一直触发
       ucKeySec=3;    //触发3号键
    }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
     uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt4++; //累加定时中断次数
     if(uiKeyTimeCnt4>const_key_time4)
     {
        uiKeyTimeCnt4=0;
        ucKeyLock4=1;  //自锁按键置位,避免一直触发
        ucKeySec=4;    //触发4号键
     }
  }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                   switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                   {
                          case 0:   //无光标显示的状态 此处的case 0可以省略
                                break;
                          case 1:   //设置第1行参数
                                ucData_1_1++;
                                                          if(ucData_1_1>99)
                                                          {
                                                             ucData_1_1=99;
                                                          }
                                ucWd1Part1Update=1; //1代表更新显示，响应函数内部会清零
                                break;
                          case 2:   //设置第2行参数
                                ucData_1_2++;
                                                          if(ucData_1_2>99)
                                                          {
                                                             ucData_1_2=99;
                                                          }
                                ucWd1Part2Update=1; //1代表更新显示，响应函数内部会清零
```

```
                        break;
                    case 3:    //设置第3行参数
                        ucData_1_3++;
                                                    if(ucData_1_3>99)
                                                    {
                                                       ucData_1_3=99;
                                                    }
                        ucWd1Part3Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 4:    //设置第4行参数
                        ucData_1_4++;
                                                    if(ucData_1_4>99)
                                                    {
                                                       ucData_1_4=99;
                                                    }
                        ucWd1Part4Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                }
                break;

        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                    case 0:    //无光标显示的状态 此处的case 0可以省略
                        break;
                    case 1:    //设置第1行参数
                        ucData_1_1--;
                                                    if(ucData_1_1>99) //一直减到最后，单片机
C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_1=0;
                                                    }
                        ucWd1Part1Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 2:    //设置第2行参数
                        ucData_1_2--;
                                                    if(ucData_1_2>99) //一直减到最后，单片机
C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_2=0;
                                                    }
                        ucWd1Part2Update=1; //1代表更新显示，响应函数内部会清零
```

```
                        break;
                    case 3:    //设置第3行参数
                        ucData_1_3--;
                                                    if(ucData_1_3>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_3=0;
                                                    }
                        ucWd1Part3Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 4:    //设置第4行参数
                        ucData_1_4--;
                                                    if(ucData_1_4>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_4=0;
                                                    }
                        ucWd1Part4Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                }
                break;

        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3:// 切换"光标"移动按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                    case 0:    //无光标显示的状态 此处的case 0可以省略
                        break;
                    case 1:    //设置第1行参数
                        ucPart=2; //光标切换到下一行
                        ucWd1Part1Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part2Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 2:    //设置第2行参数
                        ucPart=3; //光标切换到下一行
                        ucWd1Part2Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part3Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 3:    //设置第3行参数
                        ucPart=4; //光标切换到下一行
                        ucWd1Part3Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part4Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 4:    //设置第4行参数
```

```
                        ucPart=1; //光标返回到最上面第一行
                        ucWd1Part4Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part1Update=1; //更新显示最上面第一行，     目的是更新反显光标的状态
                        break;
                }
                break;

        }

        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 4: // 设置按键  对应朱兆祺学习板的S13键，按一次进入设置状态，出现反显光标。再按一次推出设置状态，消除反显光标
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                        case 0:   //无光标显示的状态
                                ucPart=1; //光标显示第一行，进入设置模式
                                ucWd1Part1Update=1; //更新显示
                                break;
                        case 1:   //设置第1行参数
                                ucPart=0; //无光标显示，退出设置模式
                                ucWd1Part1Update=1; //更新显示
                                break;
                        case 2:   //设置第2行参数
                                ucPart=0; //无光标显示，退出设置模式
                                ucWd1Part2Update=1; //更新显示
                                break;
                        case 3:   //设置第3行参数
                                ucPart=0; //无光标显示，退出设置模式
                                ucWd1Part3Update=1; //更新显示
                                break;
                        case 4:   //设置第4行参数
                                ucPart=0; //无光标显示，退出设置模式
                                ucWd1Part4Update=1; //更新显示
                                break;
                }
                break;

        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
  }
}
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
```

```
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                     break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                     break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                     break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                     break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                     break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                     break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                     break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                     break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                     break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                     break;
            case 10:
             p_ucAnyNumber=Zf816_nc;
                     break;
               default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                     break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    unsigned char ucAnyNumber_1; //分解变量的个位
    unsigned char ucAnyNumber_10; //分解变量的十位
    unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
        unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
```

```
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:   //显示窗口1的数据
/* 注释二:
* 把每一个窗口的内容分为两种类型，一种类型是那些不用经常刷新显示的内容，只有在切换窗口的时候
* 才需要更新，这种内容放在整屏更新显示的括号里，比如清屏操作等内容。另外一种是那些经常需要
* 刷新显示的内容，这种内容放在局部更新显示的括号里。
*/
                    if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
                        {
                            ucWd1Update=0;  //及时清零，避免一直更新
                ucWd1Part1Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上有所改进
                ucWd1Part2Update=1; //激活窗口1的第2行局部更新显示变量，这里在前面数码管显示框架上有所改进
                ucWd1Part3Update=1; //激活窗口1的第3行局部更新显示变量，这里在前面数码管显示框架上有所改进
                ucWd1Part4Update=1; //激活窗口1的第4行局部更新显示变量，这里在前面数码管显示框架上有所改进
                display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
                clear_all_canvas();  //把画布全部清零
                insert_buffer_to_canvas(0,0,Zf816_mao_hao,0,1,16);//把冒号的字模插入画布
                display_lattice(0,0,Hz1616_yi,0,2,16,0);    //一窗口一行，这些内容不用经常更新，只有在切换窗口的时候才更新显示
                display_lattice(1,0,Hz1616_chuang,0,2,16,0);
                display_lattice(2,0,Hz1616_kou,0,2,16,0);
                display_lattice(3,0,Hz1616_yi,0,2,16,0);
                display_lattice(4,0,Hz1616_hang,0,2,16,0);
                display_lattice(0,16,Hz1616_yi,0,2,16,0);    //一窗口二行
                display_lattice(1,16,Hz1616_chuang,0,2,16,0);
                display_lattice(2,16,Hz1616_kou,0,2,16,0);
                display_lattice(3,16,Hz1616_er,0,2,16,0);
                display_lattice(4,16,Hz1616_hang,0,2,16,0);
                display_lattice(8,0,Hz1616_yi,0,2,16,0);    //一窗口三行
                display_lattice(9,0,Hz1616_chuang,0,2,16,0);
                display_lattice(10,0,Hz1616_kou,0,2,16,0);
                display_lattice(11,0,Hz1616_san,0,2,16,0);
                display_lattice(12,0,Hz1616_hang,0,2,16,0);
                display_lattice(8,16,Hz1616_yi,0,2,16,0);    //一窗口四行
                display_lattice(9,16,Hz1616_chuang,0,2,16,0);
                display_lattice(10,16,Hz1616_kou,0,2,16,0);
                display_lattice(11,16,Hz1616_si,0,2,16,0);
                display_lattice(12,16,Hz1616_hang,0,2,16,0);
                        }
/* 注释三:
* 注意！我前面讲数码管显示的时候有一句话讲错了，我那时说<局部更新应该写在整屏更新之前>,这是不对的。
* 按照现在的显示程序框架<即整屏显示更新括号里包含了所有局部变量的激活>，应该是<整屏更新应该写在局部更新之前>
* 这样才对。
*/
```

```
            if(ucWd1Part1Update==1) //窗口1的第1行局部更新显示变量,里面放一些经常需要刷新显示的内容
            {
               ucWd1Part1Update=0; //及时清零，避免一直更新
        if(ucPart==1) //被选中
                  {
                     ucCursorFlag=1; //反显 显示
                  }
        else //没被选中
                  {
                     ucCursorFlag=0; //正常 显示
                  }
      if(ucData_1_1>=10) //有2位数以上
      {
         ucAnyNumber_10=ucData_1_1/10;  //十位
      }
      else //否则显示空
      {
         ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
      }
      ucAnyNumber_1=ucData_1_1%10/1;  //个位

      p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
      p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
      insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
      insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
      display_lattice(5,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

            }
            if(ucWd1Part2Update==1) //窗口1的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
            {
               ucWd1Part2Update=0; //及时清零，避免一直更新
        if(ucPart==2) //被选中
                  {
                     ucCursorFlag=1; //反显 显示
                  }
        else //没被选中
                  {
                     ucCursorFlag=0; //正常 显示
                  }
      if(ucData_1_2>=10) //有2位数以上
      {
         ucAnyNumber_10=ucData_1_2/10;  //十位
      }
      else //否则显示空
      {
         ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
      }
      ucAnyNumber_1=ucData_1_2%10/1;  //个位
```

```
p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
display_lattice(5,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

        }
        if(ucWd1Part3Update==1) //窗口1的第3行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
           ucWd1Part3Update=0; //及时清零，避免一直更新
    if(ucPart==3) //被选中
                {
                   ucCursorFlag=1; //反显 显示
                }
    else //没被选中
                {
                   ucCursorFlag=0; //正常 显示
                }
if(ucData_1_3>=10) //有2位数以上
{
   ucAnyNumber_10=ucData_1_3/10;  //十位
}
else //否则显示空
{
   ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
}
ucAnyNumber_1=ucData_1_3%10/1;  //个位

p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
display_lattice(13,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

        }
        if(ucWd1Part4Update==1) //窗口1的第4行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
           ucWd1Part4Update=0; //及时清零，避免一直更新
    if(ucPart==4) //被选中
                {
                   ucCursorFlag=1; //反显 显示
                }
    else //没被选中
                {
                   ucCursorFlag=0; //正常 显示
                }
if(ucData_1_4>=10) //有2位数以上
```

```
                {
                   ucAnyNumber_10=ucData_1_4/10;  //十位
                }
                else //否则显示空
                {
                   ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
                }
                ucAnyNumber_1=ucData_1_4%10/1;  //个位

                p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
                p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
                insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
                insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
                display_lattice(13,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

                        }
            break;
        //本程序只有1个窗口，所以只有一个case 1，如果要增加窗口，就直接增加 case 2，case 3...
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);          //垂直地址
         WriteCommand(0x80);            //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
```

```
}
/* 注释四:
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至3，因为画布的横向只要4个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
                  ucTemp=ucArray[j*x_amount+i];
                  if(ucFbFlag==0)
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=ucTemp; //这里的4代表画布每一行只有4个字节
                  }
                  else
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=~ucTemp; //这里的4代表画布每一行只有4个字节
                  }
      }
   }
}
/* 注释五:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
```

```
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
```

```
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

复制代码

总结陈词:

这一节讲了在一个窗口里设置不同的参数，如果有几个窗口的情况下，该如何编程？欲知详情，请听下回分解-----在多个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。

（未完待续，下节更精彩，不要走开哦）

第七十八节：在多个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。

开场白:

上一节讲了1个窗口下如何设置参数的菜单程序，这一节多增加1个窗口变成2个窗口，看看它们两个窗口之间是如何通过按键程序进行切换的。继续巩固上一节教给大家的两个知识点:

第一个知识点：我在前面讲数码管显示的时候就提出了一个 “一二级菜单显示理论”：凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。把每一个窗口的内容分为两种类型，一种类型是那些不用经常刷新显示的内容，只有在切换窗口的时候才需要更新的，这种内容放在整屏更新显示的括号里，比如清屏操作等内容。另外一种是那些经常需要刷新显示的内容，这种内容放在局部更新显示的括号里。

第二个知识点：按键如何跟液晶屏显示有机的结合起来？只要遵循鸿哥总结出来的一个规律“在不同的窗口下，根据不同的局部变量来操作不同的参数”，这样再复杂的人机交互程序都会显得很简单清晰。

具体内容，请看源代码讲解。
（1）硬件平台：基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换“光标”移动按键对应S9键，设置参数按键对应S13键。
（2）实现功能：
通过按键设置8个不同的参数。
有2个窗口。每个窗口显示4个参数。每个参数的范围是从0到99。
有4个按键：
（a） 一个是设置参数S13按键，按下此按键，液晶屏的第1个窗口第一行会出现反显的光标，表示进入设置参数模式，再次按下此按键，反显光标会消失，并且强行切换到第1个窗口，表示退出设置参数模式。
（b） 一个是移动光标S9按键，在进入设置参数的模式下，依次按下此按键，液晶屏上的光标会从上往下移动，表示选中不同的参数。当移动到每个窗口最下边那一行时，再按下此按键会进行切换窗口的操作。
（c） 一个是减数S5按键，在设置参数模式下，依次按下此按键，被选中的参数会逐渐减小。
（d） 一个是加数S1按键，在设置参数模式下，依次按下此按键，被选中的参数会逐渐加大。
（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void T0_time(); //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
```

```
void clear_all_canvas(void);  //把画布全部清零
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
```

```
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_yi[]=
{
/*--  文字:  一  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_er[]=
{
/*--  文字:  二  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x10,0x3F,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_san[]=
{
/*--  文字:  三  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x7F,0xFC,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3F,0xF8,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7F,0xFE,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_si[]=
{
/*--  文字:  四  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x7F,0xFC,0x44,0x84,0x44,0x84,0x44,0x84,0x44,0x84,0x44,0x84,0x44,0x84,
0x48,0x84,0x48,0x7C,0x50,0x04,0x60,0x04,0x40,0x04,0x7F,0xFC,0x40,0x04,0x00,0x00,
};
code unsigned char Hz1616_chuang[]=
```

```
{
/*--  文字:  窗  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x01,0x00,0x00,0x80,0x7F,0xFE,0x40,0x22,0x09,0x18,0x12,0x06,0x7F,0xF8,0x11,0x08,
0x13,0xE8,0x14,0x48,0x1A,0x88,0x11,0x08,0x12,0x88,0x14,0x08,0x1F,0xF8,0x10,0x08,
};
code unsigned char Hz1616_kou[]=
{
/*--  文字:  口  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x3F,0xF8,0x20,0x08,0x20,0x08,0x20,0x08,0x20,0x08,0x20,0x08,
0x20,0x08,0x20,0x08,0x20,0x08,0x3F,0xF8,0x20,0x08,0x20,0x08,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_hang[]=
{
/*--  文字:  行  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x00,0x1C,0x00,0x31,0xFC,0x40,0x00,0x88,0x00,0x0C,0x00,0x1B,0xFE,0x30,0x20,
0x50,0x20,0x90,0x20,0x10,0x20,0x10,0x20,0x10,0x20,0x10,0x20,0x10,0xA0,0x10,0x40,
};
unsigned char ucCanvasBuffer[]= //画布显示数组。注意，这里没有code关键字，是全局变量。初始化全部填充0x00
{
0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
};
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucPart=0; //局部变量 0代表没有选中任何一行，其它数值1到4代表选中某一行
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量       1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part2Update=0; //窗口1的第2行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part3Update=0; //窗口1的第3行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part4Update=0; //窗口1的第4行局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucData_1_1=8;  //第1个窗口第1行的被设置数据
```

```
unsigned char ucData_1_2=9;  //第1个窗口第2行的被设置数据
unsigned char ucData_1_3=10;  //第1个窗口第3行的被设置数据
unsigned char ucData_1_4=11;  //第1个窗口第4行的被设置数据
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                    key_service(); //按键服务的应用程序
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
/* 注释一:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
   key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
```

```
  TL0=0x2f;
  TR0=1;  //开中断
}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  static unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
```

```
    uiKeyTimeCnt3++; //累加定时中断次数
    if(uiKeyTimeCnt3>const_key_time3)
    {
      uiKeyTimeCnt3=0;
      ucKeyLock3=1;  //自锁按键置位,避免一直触发
      ucKeySec=3;    //触发3号键
    }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
    ucKeyLock4=0; //按键自锁标志清零
    uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
    uiKeyTimeCnt4++; //累加定时中断次数
    if(uiKeyTimeCnt4>const_key_time4)
    {
      uiKeyTimeCnt4=0;
      ucKeyLock4=1;  //自锁按键置位,避免一直触发
      ucKeySec=4;    //触发4号键
    }
  }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
              case 1:
                   switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                   {
                          case 0:   //无光标显示的状态 此处的case 0可以省略
                                break;
                          case 1:   //设置第1行参数
                                ucData_1_1++;
                                                          if(ucData_1_1>99)
                                                          {
                                                            ucData_1_1=99;
                                                          }
                                ucWd1Part1Update=1; //1代表更新显示，响应函数内部会清零
                                break;
                          case 2:   //设置第2行参数
                                ucData_1_2++;
                                                          if(ucData_1_2>99)
                                                          {
                                                            ucData_1_2=99;
                                                          }
```

```
                        ucWd1Part2Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 3:    //设置第3行参数
                        ucData_1_3++;
                                                    if(ucData_1_3>99)
                                                    {
                                                       ucData_1_3=99;
                                                    }
                        ucWd1Part3Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 4:    //设置第4行参数
                        ucData_1_4++;
                                                    if(ucData_1_4>99)
                                                    {
                                                       ucData_1_4=99;
                                                    }
                        ucWd1Part4Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                }
                break;

        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

    case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                    case 0:    //无光标显示的状态 此处的case 0可以省略
                        break;
                    case 1:    //设置第1行参数
                        ucData_1_1--;
                                                    if(ucData_1_1>99) //一直减到最后，单片机
C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_1=0;
                                                    }
                        ucWd1Part1Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 2:    //设置第2行参数
                        ucData_1_2--;
                                                    if(ucData_1_2>99) //一直减到最后，单片机
C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_2=0;
                                                    }
```

```
                        ucWd1Part2Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 3:    //设置第3行参数
                        ucData_1_3--;
                                                    if(ucData_1_3>99) //一直减到最后，单片机
C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_3=0;
                                                    }
                        ucWd1Part3Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 4:    //设置第4行参数
                        ucData_1_4--;
                                                    if(ucData_1_4>99) //一直减到最后，单片机
C语言编译器有一个特征，0减去1会溢出变成255(0xff)
                                                    {
                                                       ucData_1_4=0;
                                                    }
                        ucWd1Part4Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                }
                break;

        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3:// 切换"光标"移动按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                    case 0:    //无光标显示的状态 此处的case 0可以省略
                        break;
                    case 1:    //设置第1行参数
                        ucPart=2; //光标切换到下一行
                        ucWd1Part1Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part2Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 2:    //设置第2行参数
                        ucPart=3; //光标切换到下一行
                        ucWd1Part2Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part3Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 3:    //设置第3行参数
                        ucPart=4; //光标切换到下一行
                        ucWd1Part3Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part4Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
```

```
                        case 4:    //设置第4行参数
                              ucPart=1; //光标返回到最上面第一行
                              ucWd1Part4Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                              ucWd1Part1Update=1; //更新显示最上面第一行，      目的是更新反显光标的状态
                              break;
                  }
                  break;

            }

            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;

      case 4: // 设置按键  对应朱兆祺学习板的S13键，按一次进入设置状态，出现反显光标。再按一次推出设置状态，消除反显光标
            switch(ucWd)  //在不同的窗口下，设置不同的参数
            {
                case 1:
                  switch(ucPart)  //在窗口1下，根据不同的局部变量来设置不同的参数
                  {
                        case 0:    //无光标显示的状态
                              ucPart=1; //光标显示第一行，进入设置模式
                              ucWd1Part1Update=1; //更新显示
                              break;
                        case 1:    //设置第1行参数
                              ucPart=0; //无光标显示，退出设置模式
                              ucWd1Part1Update=1; //更新显示
                              break;
                        case 2:    //设置第2行参数
                              ucPart=0; //无光标显示，退出设置模式
                              ucWd1Part2Update=1; //更新显示
                              break;
                        case 3:    //设置第3行参数
                              ucPart=0; //无光标显示，退出设置模式
                              ucWd1Part3Update=1; //更新显示
                              break;
                        case 4:    //设置第4行参数
                              ucPart=0; //无光标显示，退出设置模式
                              ucWd1Part4Update=1; //更新显示
                              break;
                  }
                  break;

            }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    }
}
```

```
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                    break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                    break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                    break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                    break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                    break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                    break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                    break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                    break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                    break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                    break;
            case 10:
             p_ucAnyNumber=Zf816_nc;
                    break;
               default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                    break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    unsigned char ucAnyNumber_1; //分解变量的个位
    unsigned char ucAnyNumber_10; //分解变量的十位
    unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
```

```
        unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:   //显示窗口1的数据
/* 注释二:
* 把每一个窗口的内容分为两种类型，一种类型是那些不用经常刷新显示的内容，只有在切换窗口的时候
* 才需要更新，这种内容放在整屏更新显示的括号里，比如清屏操作等内容。另外一种是那些经常需要
* 刷新显示的内容，这种内容放在局部更新显示的括号里。
*/
                    if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
                        {
                           ucWd1Update=0;  //及时清零，避免一直更新
                ucWd1Part1Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上有所改进
                ucWd1Part2Update=1; //激活窗口1的第2行局部更新显示变量，这里在前面数码管显示框架上有所改进
                ucWd1Part3Update=1; //激活窗口1的第3行局部更新显示变量，这里在前面数码管显示框架上有所改进
                ucWd1Part4Update=1; //激活窗口1的第4行局部更新显示变量，这里在前面数码管显示框架上有所改进
                display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
                clear_all_canvas();  //把画布全部清零
                insert_buffer_to_canvas(0,0,Zf816_mao_hao,0,1,16);//把冒号的字模插入画布
                display_lattice(0,0,Hz1616_yi,0,2,16,0);     //一窗口一行，这些内容不用经常更新，只有在切换窗口的时候才更新显示
                display_lattice(1,0,Hz1616_chuang,0,2,16,0);
                display_lattice(2,0,Hz1616_kou,0,2,16,0);
                display_lattice(3,0,Hz1616_yi,0,2,16,0);
                display_lattice(4,0,Hz1616_hang,0,2,16,0);
                display_lattice(0,16,Hz1616_yi,0,2,16,0);     //一窗口二行
                display_lattice(1,16,Hz1616_chuang,0,2,16,0);
                display_lattice(2,16,Hz1616_kou,0,2,16,0);
                display_lattice(3,16,Hz1616_er,0,2,16,0);
                display_lattice(4,16,Hz1616_hang,0,2,16,0);
                display_lattice(8,0,Hz1616_yi,0,2,16,0);     //一窗口三行
                display_lattice(9,0,Hz1616_chuang,0,2,16,0);
                display_lattice(10,0,Hz1616_kou,0,2,16,0);
                display_lattice(11,0,Hz1616_san,0,2,16,0);
                display_lattice(12,0,Hz1616_hang,0,2,16,0);
                display_lattice(8,16,Hz1616_yi,0,2,16,0);     //一窗口四行
                display_lattice(9,16,Hz1616_chuang,0,2,16,0);
                display_lattice(10,16,Hz1616_kou,0,2,16,0);
                display_lattice(11,16,Hz1616_si,0,2,16,0);
                display_lattice(12,16,Hz1616_hang,0,2,16,0);
                        }
/* 注释三:
* 注意! 我前面讲数码管显示的时候有一句话讲错了，我那时说<局部更新应该写在整屏更新之前>,这是不对的。
* 按照现在的显示程序框架<即整屏显示更新括号里包含了所有局部变量的激活>，应该是<整屏更新应该写在局部更新之前>
* 这样才对。
```

```
*/
                if(ucWd1Part1Update==1) //窗口1的第1行局部更新显示变量,里面放一些经常需要刷新显示的内容
                {
                   ucWd1Part1Update=0; //及时清零，避免一直更新
    if(ucPart==1) //被选中
                   {
                      ucCursorFlag=1; //反显 显示
                   }
    else //没被选中
                   {
                      ucCursorFlag=0; //正常 显示
                   }
               if(ucData_1_1>=10) //有2位数以上
               {
                  ucAnyNumber_10=ucData_1_1/10;  //十位
               }
               else //否则显示空
               {
                  ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
               }
               ucAnyNumber_1=ucData_1_1%10/1;  //个位

               p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
               p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
               insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
               insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
               display_lattice(5,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

            }
                if(ucWd1Part2Update==1) //窗口1的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
                {
                   ucWd1Part2Update=0; //及时清零，避免一直更新
    if(ucPart==2) //被选中
                   {
                      ucCursorFlag=1; //反显 显示
                   }
    else //没被选中
                   {
                      ucCursorFlag=0; //正常 显示
                   }
               if(ucData_1_2>=10) //有2位数以上
               {
                  ucAnyNumber_10=ucData_1_2/10;  //十位
               }
               else //否则显示空
               {
                  ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
               }
```

```
ucAnyNumber_1=ucData_1_2%10/1;  //个位

p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
display_lattice(5,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

        }
        if(ucWd1Part3Update==1) //窗口1的第3行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
           ucWd1Part3Update=0; //及时清零，避免一直更新
    if(ucPart==3) //被选中
                {
                   ucCursorFlag=1; //反显 显示
                }
    else //没被选中
                {
                   ucCursorFlag=0; //正常 显示
                }
if(ucData_1_3>=10) //有2位数以上
{
   ucAnyNumber_10=ucData_1_3/10;  //十位
}
else //否则显示空
{
   ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
}
ucAnyNumber_1=ucData_1_3%10/1;  //个位

p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
display_lattice(13,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

        }
        if(ucWd1Part4Update==1) //窗口1的第4行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
           ucWd1Part4Update=0; //及时清零，避免一直更新
    if(ucPart==4) //被选中
                {
                   ucCursorFlag=1; //反显 显示
                }
    else //没被选中
                {
                   ucCursorFlag=0; //正常 显示
                }
```

```
                if(ucData_1_4>=10) //有2位数以上
                {
                   ucAnyNumber_10=ucData_1_4/10;  //十位
                }
                else //否则显示空
                {
                   ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
                }
                ucAnyNumber_1=ucData_1_4%10/1;  //个位

                p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
                p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
                insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
                insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
                display_lattice(13,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

                        }
            break;
        //本程序只有1个窗口，所以只有一个case 1，如果要增加窗口，就直接增加 case 2，case 3...
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
```

```
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释四:
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至3，因为画布的横向只要4个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
                  ucTemp=ucArray[j*x_amount+i];
                  if(ucFbFlag==0)
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=ucTemp; //这里的4代表画布每一行只有4个字节
                  }
                  else
                  {
            ucCanvasBuffer[(y+j)*4+x+i]=~ucTemp; //这里的4代表画布每一行只有4个字节
                  }
      }
   }
}
/* 注释五:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
```

```
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
```

```
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

复制代码

总结陈词:

这一节讲了在一个窗口里设置不同的参数，如果有几个窗口的情况下，该如何编程？欲知详情，请听下回分解-----在多个窗口里通过移动光标来设置不同参数的液晶屏菜单程序。

（未完待续，下节更精彩，不要走开哦）

第七十九节: 通过主菜单移动光标来进入子菜单窗口的液晶屏程序。

开场白:

其实主菜单窗口与子菜单窗口本质都是多窗口菜单程序，只不过我在按键服务程序里面建立起来了一条主窗口与子窗口的关系链。这个关系链还是用switch语句搭建起来的，在某个窗口某个局部显示上，操作某个按键就会切换到不同的窗口显示。

继续巩固上一节教给大家的两个知识点:

第一个知识点: 我在前面讲数码管显示的时候就提出了一个 “一二级菜单显示理论”: 凡是人机界面显示，不管是数码管还是液晶屏，都可以把显示的内容分成不同的窗口来显示，每个显示的窗口中又可以分成不同的局部显示。其中窗口就是一级菜单，用ucWd变量表示。局部就是二级菜单，用ucWdxPart来表示。不同的窗口，会有不同的更新显示变量ucWdXUpdate来对应，表示整屏全部更新显示。不同的局部，也会有不同的更新显示变量ucWdXPartYUpdate来对应，表示局部更新显示。把每一个窗口的内容分为两种类型，一种类型是那些不用经常刷新显示的内容，只有在切换窗口的时候才需要更新的，这种内容放在整屏更新显示的括号里，比如清屏操作等内容。另外一种是那些经常需要刷新显示

的内容，这种内容放在局部更新显示的括号里。

第二个知识点：按键如何跟液晶屏显示有机的结合起来？只要遵循鸿哥总结出来的一个规律“在不同的窗口下，根据不同的局部变量来操作不同的参数”，这样再复杂的人机交互程序都会显得很简单清晰。

具体内容，请看源代码讲解。

（1）硬件平台：基于朱兆祺51单片机学习板。加按键对应S1键，减按键对应S5键，切换“光标”移动按键对应S9键，设置参数按键对应S13键。

（2）实现功能：

通过按键设置6个不同的参数。

有4个窗口。第1个窗口是主菜单界面，通过光标切换可以进去设置不同参数的子菜单界面。第2个窗口是设置时间范围界面。第3个窗口是设置速度范围界面。第4个窗口是设置频率范围界面。每个设置界面显示2个参数。每个参数的范围是从0到99。

有4个按键：

（a）　　一个是进入和退出S13按键，按一次进入选中的子菜单。再按一次退出子菜单。

（b）　　一个是移动光标S9按键，依次按下此按键，液晶屏上的光标会从上往下移动，表示选中不同的参数。当移动到每个窗口最下边那一行时，再按下此按键会把光标移动到第一个参数。

（c）　　一个是减数S5按键，在设置参数模式下，依次按下此按键，被选中的参数会逐渐减小。

（d）　　一个是加数S1按键，在设置参数模式下，依次按下此按键，被选中的参数会逐渐加大。

（3）源代码讲解如下：

```
#include "REG52.H"
/* 注释一:
* 本程序用到的变量比较多，所以在keil编译模式里要设置一下编译模式memory model,
* 否则编译会出错.右键单击Target选择“Options for Target′Target1′”就会出来一个框
* 在memory model中选择compact:variables in pdata 就可以了。
*/
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
```

```
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void T0_time(); //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
void wd1(void);//窗口1  主菜单
void wd2(void);//窗口2  设置时间
void wd3(void);//窗口3  设置速度
void wd4(void);//窗口4  设置频率
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
```

```
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_zhu[]=
{
/*--  文字:  主  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x02,0x00,0x01,0x80,0x01,0x00,0x00,0x08,0x3F,0xFC,0x01,0x00,0x01,0x00,0x01,0x08,
0x3F,0xFC,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x04,0x7F,0xFE,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_cai[]=
{
/*--  文字:  菜  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x04,0x40,0xFF,0xFE,0x04,0x40,0x04,0x40,0x3F,0xF8,0x22,0x08,0x11,0x10,0x08,0x20,
0x01,0x00,0x7F,0xFE,0x03,0x80,0x05,0x40,0x09,0x30,0x11,0x1C,0x61,0x08,0x01,0x00,
};
code unsigned char Hz1616_dan[]=
{
```

```
/*--  文字:  单  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x20,0x06,0x30,0x04,0x40,0x3F,0xF8,0x21,0x08,0x3F,0xF8,0x21,0x08,0x21,0x08,
0x3F,0xF8,0x21,0x08,0x01,0x00,0xFF,0xFE,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,
};
code unsigned char Hz1616_she[]=
{
/*--  文字:  设  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x40,0x00,0x21,0xF0,0x31,0x10,0x21,0x10,0x01,0x10,0x01,0x10,0xE2,0x0E,0x25,0xF8,
0x21,0x08,0x21,0x08,0x20,0x90,0x20,0x90,0x28,0x60,0x30,0x90,0x23,0x0E,0x0C,0x04,
};
code unsigned char Hz1616_zhi[]=
{
/*--  文字:  置  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x3F,0xF8,0x24,0x48,0x24,0x48,0x3F,0xF8,0x01,0x00,0x7F,0xFC,0x02,0x00,0x1F,0xF0,
0x10,0x10,0x1F,0xF0,0x10,0x10,0x1F,0xF0,0x10,0x10,0x1F,0xF0,0x10,0x10,0xFF,0xFE,
};
code unsigned char Hz1616_su[]=
{
/*--  文字:  速  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x80,0x40,0x80,0x2F,0xFC,0x20,0x80,0x00,0x80,0x07,0xF8,0xE4,0x88,0x24,0x88,
0x27,0xF8,0x21,0xA0,0x22,0x98,0x2C,0x88,0x20,0x80,0x50,0x80,0x8F,0xFE,0x00,0x00,
};
code unsigned char Hz1616_du[]=
{
/*--  文字:  度  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x01,0x00,0x00,0x80,0x3F,0xFE,0x22,0x20,0x22,0x20,0x2F,0xFC,0x22,0x20,0x23,0xE0,
0x20,0x00,0x27,0xF8,0x22,0x10,0x21,0x20,0x20,0xC0,0x41,0x30,0x46,0x0E,0x98,0x04,
};
code unsigned char Hz1616_shi[]=
{
/*--  文字:  时  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x10,0x00,0x10,0x7C,0x10,0x44,0x10,0x47,0xFE,0x44,0x10,0x7C,0x10,0x45,0x10,
0x44,0x90,0x44,0x90,0x7C,0x10,0x00,0x10,0x00,0x10,0x00,0x10,0x00,0x50,0x00,0x20,
};
code unsigned char Hz1616_jian[]=
{
/*--  文字:  间  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x20,0x00,0x13,0xFC,0x10,0x04,0x40,0x04,0x47,0xE4,0x44,0x24,0x44,0x24,0x47,0xE4,
0x44,0x24,0x44,0x24,0x47,0xE4,0x40,0x04,0x40,0x04,0x40,0x04,0x40,0x14,0x40,0x08,
};
code unsigned char Hz1616_pin[]=
{
/*--  文字:  频  --*/
```

```
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x00,0x08,0xFE,0x4E,0x20,0x48,0x40,0x48,0xFC,0xFE,0x84,0x00,0xA4,0x08,0xA4,
0x4A,0xA4,0x4A,0xA4,0x84,0xA4,0x08,0x50,0x10,0x48,0x20,0x86,0xC3,0x02,0x00,0x00,
};
code unsigned char Hz1616_lv[]=
{
/*--  文字:  率  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x02,0x00,0x01,0x00,0x7F,0xFE,0x41,0x00,0x22,0x28,0x17,0xD0,0x04,0x80,0x11,0x10,
0x22,0x48,0x47,0xC4,0x01,0x20,0xFF,0xFE,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,
};
code unsigned char Hz1616_fan[]=
{
/*--  文字:  范  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x04,0x20,0x04,0x20,0xFF,0xFE,0x04,0x60,0x40,0x00,0x31,0xF8,0x91,0x08,0x61,0x08,
0x49,0x08,0x09,0x38,0x11,0x10,0xE1,0x00,0x21,0x04,0x21,0x04,0x20,0xFC,0x20,0x00,
};
code unsigned char Hz1616_wei[]=
{
/*--  文字:  围  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x7F,0xFC,0x42,0x04,0x42,0x04,0x5F,0xF4,0x42,0x04,0x4F,0xE4,0x42,0x04,0x5F,0xE4,
0x42,0x24,0x42,0x24,0x42,0x24,0x42,0xA4,0x42,0x44,0x40,0x04,0x7F,0xFC,0x40,0x04,
};
code unsigned char Hz1616_shang[]=
{
/*--  文字:  上  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0xF8,0x01,0x00,
0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x04,0x7F,0xFE,0x00,0x00,
};
code unsigned char Hz1616_xia[]=
{
/*--  文字:  下  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x04,0x7F,0xFE,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0xC0,0x01,0x60,0x01,0x30,
0x01,0x20,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x00,0x00,
};
code unsigned char Hz1616_xian[]=
{
/*--  文字:  限  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0xFB,0xF8,0x92,0x08,0x93,0xF8,0xA2,0x08,0xA2,0x08,0x93,0xF8,0x8A,0x80,
0x8A,0x48,0xAA,0x50,0x92,0x20,0x82,0x20,0x82,0x10,0x82,0x8E,0x83,0x04,0x82,0x00,
};
unsigned char ucCanvasBuffer[]= //画布显示数组。注意，这里没有code关键字，是全局变量。初始化全部填充0x00
{
0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,
```

```
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,
};
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucWd1Part=1;  //窗口1的局部变量，代表选中某一行。
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量      1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part2Update=0; //窗口1的第2个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part3Update=0; //窗口1的第3个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd2Part=1;  //窗口2的局部变量，代表选中某一行。
unsigned char ucWd2Update=0; //窗口2的整屏更新显示变量      1代表更新显示，响应函数内部会清零
unsigned char ucWd2Part1Update=0; //窗口2的第1个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd2Part2Update=0; //窗口2的第2个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd3Part=1;  //窗口3的局部变量，代表选中某一行。
unsigned char ucWd3Update=0; //窗口3的整屏更新显示变量      1代表更新显示，响应函数内部会清零
unsigned char ucWd3Part1Update=0; //窗口3的第1个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd3Part2Update=0; //窗口3的第2个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd4Part=1;  //窗口4的局部变量，代表选中某一行。
unsigned char ucWd4Update=0; //窗口4的整屏更新显示变量      1代表更新显示，响应函数内部会清零
unsigned char ucWd4Part1Update=0; //窗口4的第1个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucWd4Part2Update=0; //窗口4的第2个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucTimeH=2;  //设置时间的上限数据
unsigned char ucTimeL=1;  //设置时间的下限数据
unsigned char ucSpeedH=4;  //设置速度的上限数据
unsigned char ucSpeedL=3;  //设置速度的下限数据
unsigned char ucFreqH=6;  //设置频率的上限数据
unsigned char ucFreqL=5;  //设置频率的下限数据
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                    key_service(); //按键服务的应用程序
```

```
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
/* 注释二:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1和S5两个按键就是本程序中用到的两个独立按键。
*/
   key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  static unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
  static unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
  static unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
```

```
static unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
static unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock1=0; //按键自锁标志清零
   uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
{
   uiKeyTimeCnt1++; //累加定时中断次数
   if(uiKeyTimeCnt1>const_key_time1)
   {
      uiKeyTimeCnt1=0;
      ucKeyLock1=1;  //自锁按键置位,避免一直触发
      ucKeySec=1;    //触发1号键
   }
}
if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock2=0; //按键自锁标志清零
   uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
{
   uiKeyTimeCnt2++; //累加定时中断次数
   if(uiKeyTimeCnt2>const_key_time2)
   {
      uiKeyTimeCnt2=0;
      ucKeyLock2=1;  //自锁按键置位,避免一直触发
      ucKeySec=2;    //触发2号键
   }
}
if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock3=0; //按键自锁标志清零
   uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
}
else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
{
   uiKeyTimeCnt3++; //累加定时中断次数
   if(uiKeyTimeCnt3>const_key_time3)
   {
      uiKeyTimeCnt3=0;
      ucKeyLock3=1;  //自锁按键置位,避免一直触发
      ucKeySec=3;    //触发3号键
   }
}
if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
{
   ucKeyLock4=0; //按键自锁标志清零
```

```
      uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
   }
   else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
   {
      uiKeyTimeCnt4++; //累加定时中断次数
      if(uiKeyTimeCnt4>const_key_time4)
      {
         uiKeyTimeCnt4=0;
         ucKeyLock4=1;  //自锁按键置位,避免一直触发
         ucKeySec=4;    //触发4号键
      }
   }
}
void key_service(void) //按键服务的应用程序
{
   switch(ucKeySec) //按键服务状态切换
   {
     case 1:// 加按键 对应朱兆祺学习板的S1键
           switch(ucWd)  //在不同的窗口下，设置不同的参数
           {
               case 2:  //窗口2  设置时间
                    switch(ucWd2Part)  //在窗口2下，根据不同的局部变量来设置不同的参数
                    {
                           case 1:   //设置时间上限
                                 ucTimeH++;
                                 if(ucTimeH>99)
                                 {
                                    ucTimeH=99;
                                 }
                                 ucWd2Part1Update=1; //1代表更新显示，响应函数内部会清零
                                 break;
                           case 2:   //设置时间下限
                                 ucTimeL++;
                                 if(ucTimeL>99)
                                 {
                                    ucTimeL=99;
                                 }
                                 ucWd2Part2Update=1; //1代表更新显示，响应函数内部会清零
                                 break;

                    }
                    break;
               case 3:  //窗口3  设置速度
                    switch(ucWd3Part)  //在窗口3下，根据不同的局部变量来设置不同的参数
                    {
                           case 1:   //设置速度上限
                                 ucSpeedH++;
                                 if(ucSpeedH>99)
                                 {
                                    ucSpeedH=99;
```

```
                        }
                        ucWd3Part1Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 2:   //设置速度下限
                        ucSpeedL++;
                        if(ucSpeedL>99)
                        {
                           ucSpeedL=99;
                        }
                        ucWd3Part2Update=1; //1代表更新显示，响应函数内部会清零
                        break;

                }
                break;
            case 4:  //窗口4  设置速度
                switch(ucWd4Part)  //在窗口4下，根据不同的局部变量来设置不同的参数
                {
                    case 1:   //设置频率上限
                        ucFreqH++;
                        if(ucFreqH>99)
                        {
                           ucFreqH=99;
                        }
                        ucWd4Part1Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 2:   //设置频率下限
                        ucFreqL++;
                        if(ucFreqL>99)
                        {
                           ucFreqL=99;
                        }
                        ucWd4Part2Update=1; //1代表更新显示，响应函数内部会清零
                        break;

                }
                break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 2:  //窗口2 设置时间
                switch(ucWd2Part)  //在窗口2下，根据不同的局部变量来设置不同的参数
                {
                    case 1:   //设置时间上限
                        ucTimeH--;
                        if(ucTimeH>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢出
```

```
变成255(0xff)
                                {
                                   ucTimeH=0;
                                }
                                ucWd2Part1Update=1; //1代表更新显示，响应函数内部会清零
                                break;
                        case 2:    //设置时间下限
                                ucTimeL--;
                                if(ucTimeL>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢出
变成255(0xff)
                                {
                                   ucTimeL=0;
                                }
                                ucWd2Part2Update=1; //1代表更新显示，响应函数内部会清零
                                break;
                    }
                    break;
             case 3:  //窗口3  设置速度
                    switch(ucWd3Part)  //在窗口3下，根据不同的局部变量来设置不同的参数
                    {
                        case 1:    //设置速度上限
                                ucSpeedH--;
                                if(ucSpeedH>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢
出变成255(0xff)
                                {
                                   ucSpeedH=0;
                                }
                                ucWd3Part1Update=1; //1代表更新显示，响应函数内部会清零
                                break;
                        case 2:    //设置速度下限
                                ucSpeedL--;
                                if(ucSpeedL>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢
出变成255(0xff)
                                {
                                   ucSpeedL=0;
                                }
                                ucWd3Part2Update=1; //1代表更新显示，响应函数内部会清零
                                break;
                    }
                    break;
             case 4:  //窗口4  设置频率
                    switch(ucWd4Part)  //在窗口4下，根据不同的局部变量来设置不同的参数
                    {
                        case 1:    //设置频率上限
                                ucFreqH--;
                                if(ucFreqH>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢出
变成255(0xff)
                                {
                                   ucFreqH=0;
                                }
```

```
                        ucWd4Part1Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                    case 2:    //设置频率下限
                        ucFreqL--;
                        if(ucFreqL>99) //一直减到最后，单片机C语言编译器有一个特征，0减去1会溢出
变成255(0xff)
                        {
                           ucFreqL=0;
                        }
                        ucWd4Part2Update=1; //1代表更新显示，响应函数内部会清零
                        break;
                }
                break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3:// 切换"光标"移动按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1: //窗口1 主菜单
                switch(ucWd1Part)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                    case 1:    //设置时间
                        ucWd1Part=2; //光标切换到下一行
                        ucWd1Part1Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part2Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 2:    //设置速度
                        ucWd1Part=3; //光标切换到下一行
                        ucWd1Part2Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part3Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 3:    //设置第3行参数
                        ucWd1Part=1; //光标返回到第一行
                        ucWd1Part3Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd1Part1Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                }
                break;
            case 2: //窗口2 设置时间
                switch(ucWd2Part)  //在窗口2下，根据不同的局部变量来设置不同的参数
                {
                    case 1:    //时间上限
                        ucWd2Part=2; //光标切换到下一行
                        ucWd2Part1Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd2Part2Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                    case 2:    //时间下限
                        ucWd2Part=1; //光标返回到第一行
```

```
                        ucWd2Part2Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                        ucWd2Part1Update=1; //更新显示下一行，     目的是更新反显光标的状态
                        break;
                }
                break;
        case 3: //窗口3 设置速度
                switch(ucWd3Part)  //在窗口3下，根据不同的局部变量来设置不同的参数
                {
                        case 1:   //速度上限
                                ucWd3Part=2; //光标切换到下一行
                                ucWd3Part1Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                                ucWd3Part2Update=1; //更新显示下一行，     目的是更新反显光标的状态
                                break;
                        case 2:   //速度下限
                                ucWd3Part=1; //光标返回到第一行
                                ucWd3Part2Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                                ucWd3Part1Update=1; //更新显示下一行，     目的是更新反显光标的状态
                                break;
                }
                break;
        case 4: //窗口4 设置频率
                switch(ucWd4Part)  //在窗口4下，根据不同的局部变量来设置不同的参数
                {
                        case 1:   //频率上限
                                ucWd4Part=2; //光标切换到下一行
                                ucWd4Part1Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                                ucWd4Part2Update=1; //更新显示下一行，     目的是更新反显光标的状态
                                break;
                        case 2:   //频率下限
                                ucWd4Part=1; //光标返回到第一行
                                ucWd4Part2Update=1; //更新显示原来那一行，目的是更新反显光标的状态
                                ucWd4Part1Update=1; //更新显示下一行，     目的是更新反显光标的状态
                                break;
                }
                break;
        }

        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 4: // 进入和退出按键  对应朱兆祺学习板的S13键，按一次进入选中的子菜单。再按一次退出子菜单。
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:  //窗口1
                switch(ucWd1Part)  //在窗口1下，根据不同的局部变量来设置不同的参数
                {
                        case 1:   //设置时间
                                                ucWd=2; //进入设置时间的窗口2
                                                        ucWd2Update=1; //窗口2整屏更新
```

```
                        break;
                    case 2:    //设置速度
                                                ucWd=3; //进入设置速度的窗口3
                                                        ucWd3Update=1; //窗口3整屏更新
                        break;
                    case 3:    //设置频率
                                                ucWd=4; //进入设置频率的窗口4
                                                        ucWd4Update=1; //窗口4整屏更新
                        break;
                }
                break;
            case 2:  //窗口2
                        ucWd=1;         //返回主菜单窗口1
                        ucWd1Update=1; //窗口1整屏更新
                break;
            case 3:  //窗口3
                        ucWd=1;         //返回主菜单窗口1
                        ucWd1Update=1; //窗口1整屏更新
                break;
             case 4:  //窗口4
                        ucWd=1;         //返回主菜单窗口1
                        ucWd1Update=1; //窗口1整屏更新
                break;
        }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
  }
}
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                    break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                    break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                    break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                    break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                    break;
            case 5:
```

```
            p_ucAnyNumber=Zf816_5;
                    break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                    break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                    break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                    break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                    break;
            case 10:
             p_ucAnyNumber=Zf816_nc;
                    break;
               default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                    break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:
              wd1();  //主菜单
              break;
        case 2:
              wd2();  //设置时间
              break;
        case 3:
              wd3();  //设置速度
              break;
        case 4:
              wd4();  //设置频率
              break;
        //本程序只有4个窗口，所以只有4个case ，如果要增加窗口，就直接增加 case 5, case 6...
    }
}
void wd1(void)  //窗口1  主菜单
{
    unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
/* 注释三:
* 把每一个窗口的内容分为两种类型，一种类型是那些不用经常刷新显示的内容，只有在切换窗口的时候
* 才需要更新，这种内容放在整屏更新显示的括号里，比如清屏操作等内容。另外一种是那些经常需要
* 刷新显示的内容，这种内容放在局部更新显示的括号里。
*/
```

```
    if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
    {
        ucWd1Update=0;  //及时清零，避免一直更新
        ucWd1Part1Update=1; //激活窗口1的第1个局部更新显示变量
        ucWd1Part2Update=1; //激活窗口1的第2个局部更新显示变量
        ucWd1Part3Update=1; //激活窗口1的第3个局部更新显示变量
        display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
        clear_all_canvas();  //把画布全部清零
        insert_buffer_to_canvas(0,0,Zf816_mao_hao,0,1,16);//把冒号的字模插入画布
        display_lattice(2,0,Hz1616_zhu,0,2,16,0);     //主菜单。这些内容不用经常更新，只有在切换窗口的时候才更新显示
        display_lattice(3,0,Hz1616_cai,0,2,16,0);
        display_lattice(4,0,Hz1616_dan,0,2,16,0);
    }
/* 注释四:
* 注意! 我前面讲数码管显示的时候有一句话讲错了，我那时说<局部更新应该写在整屏更新之前>,这是不对的。
* 按照现在的显示程序框架<即整屏显示更新括号里包含了所有局部变量的激活>，应该是<整屏更新应该写在局部更新之前>
* 这样才对。
*/
    if(ucWd1Part1Update==1) //窗口1的第1个局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
        ucWd1Part1Update=0; //及时清零，避免一直更新
        if(ucWd1Part==1) //被选中
        {
            ucCursorFlag=1; //反显 显示
        }
        else //没被选中
        {
            ucCursorFlag=0; //正常 显示
        }
               display_lattice(0,16,Hz1616_she,ucCursorFlag,2,16,0);     //设置时间范围
        display_lattice(1,16,Hz1616_zhi,ucCursorFlag,2,16,0);
        display_lattice(2,16,Hz1616_shi,ucCursorFlag,2,16,0);
        display_lattice(3,16,Hz1616_jian,ucCursorFlag,2,16,0);
        display_lattice(4,16,Hz1616_fan,ucCursorFlag,2,16,0);
        display_lattice(5,16,Hz1616_wei,ucCursorFlag,2,16,0);

    }
    if(ucWd1Part2Update==1) //窗口1的第2个局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
         ucWd1Part2Update=0; //及时清零，避免一直更新
         if(ucWd1Part==2) //被选中
         {
            ucCursorFlag=1; //反显 显示
         }
         else //没被选中
         {
            ucCursorFlag=0; //正常 显示
         }
```

```
        display_lattice(8,0,Hz1616_she,ucCursorFlag,2,16,0);       //设置速度范围
        display_lattice(9,0,Hz1616_zhi,ucCursorFlag,2,16,0);
        display_lattice(10,0,Hz1616_su,ucCursorFlag,2,16,0);
        display_lattice(11,0,Hz1616_du,ucCursorFlag,2,16,0);
        display_lattice(12,0,Hz1616_fan,ucCursorFlag,2,16,0);
        display_lattice(13,0,Hz1616_wei,ucCursorFlag,2,16,0);

    }
    if(ucWd1Part3Update==1) //窗口1的第3行局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
        ucWd1Part3Update=0; //及时清零，避免一直更新
        if(ucWd1Part==3) //被选中
        {
            ucCursorFlag=1; //反显 显示
        }
        else //没被选中
        {
            ucCursorFlag=0; //正常 显示
        }
        display_lattice(8,16,Hz1616_she,ucCursorFlag,2,16,0);     //设置频率范围
        display_lattice(9,16,Hz1616_zhi,ucCursorFlag,2,16,0);
        display_lattice(10,16,Hz1616_pin,ucCursorFlag,2,16,0);
        display_lattice(11,16,Hz1616_lv,ucCursorFlag,2,16,0);
        display_lattice(12,16,Hz1616_fan,ucCursorFlag,2,16,0);
        display_lattice(13,16,Hz1616_wei,ucCursorFlag,2,16,0);
    }

}
void wd2(void)  //窗口2 设置时间
{
    unsigned char ucAnyNumber_1; //分解变量的个位
    unsigned char ucAnyNumber_10; //分解变量的十位
    unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
    unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    if(ucWd2Update==1)  //窗口2整屏更新，里面只放那些不用经常刷新显示的内容
    {
        ucWd2Update=0;  //及时清零，避免一直更新
        ucWd2Part1Update=1; //激活窗口2的第1个局部更新显示变量，这里在前面数码管显示框架上有所改进
        ucWd2Part2Update=1; //激活窗口2的第2个局部更新显示变量，这里在前面数码管显示框架上有所改进
        display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
        clear_all_canvas();  //把画布全部清零
        insert_buffer_to_canvas(0,0,Zf816_mao_hao,0,1,16);//把冒号的字模插入画布
        display_lattice(2,0,Hz1616_she,0,2,16,0);     //设置时间。这些内容不用经常更新，只有在切换窗口的
时候才更新显示
        display_lattice(3,0,Hz1616_zhi,0,2,16,0);
        display_lattice(4,0,Hz1616_shi,0,2,16,0);
        display_lattice(5,0,Hz1616_jian,0,2,16,0);
                display_lattice(0,16,Hz1616_shi,0,2,16,0);     //时间上限
        display_lattice(1,16,Hz1616_jian,0,2,16,0);
```

```
    display_lattice(2,16,Hz1616_shang,0,2,16,0);
    display_lattice(3,16,Hz1616_xian,0,2,16,0);
    display_lattice(8,0,Hz1616_shi,0,2,16,0);  //时间下限
    display_lattice(9,0,Hz1616_jian,0,2,16,0);
    display_lattice(10,0,Hz1616_xia,0,2,16,0);
    display_lattice(11,0,Hz1616_xian,0,2,16,0);
}
if(ucWd2Part1Update==1) //窗口2的第1个局部更新显示变量,里面放一些经常需要刷新显示的内容
{
    ucWd2Part1Update=0; //及时清零, 避免一直更新
    if(ucWd2Part==1) //被选中
    {
        ucCursorFlag=1; //反显 显示
    }
    else //没被选中
    {
        ucCursorFlag=0; //正常 显示
    }
    if(ucTimeH>=10) //有2位数以上
    {
        ucAnyNumber_10=ucTimeH/10;  //十位
    }
    else //否则显示空
    {
        ucAnyNumber_10=10;  //在下面的转换函数中, 代码10表示空字模
    }
    ucAnyNumber_1=ucTimeH%10/1;  //个位

    p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
    p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
    insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
    insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
    display_lattice(4,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

}
if(ucWd2Part2Update==1) //窗口2的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
{
     ucWd2Part2Update=0; //及时清零, 避免一直更新
     if(ucWd2Part==2) //被选中
     {
         ucCursorFlag=1; //反显 显示
     }
     else //没被选中
     {
         ucCursorFlag=0; //正常 显示
     }
     if(ucTimeL>=10) //有2位数以上
     {
         ucAnyNumber_10=ucTimeL/10;  //十位
     }
```

```
        else //否则显示空
        {
            ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
        }
        ucAnyNumber_1=ucTimeL%10/1;  //个位

        p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
        p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
        insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
        insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
        display_lattice(12,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

    }

}
void wd3(void)  //窗口3 设置速度
{
    unsigned char ucAnyNumber_1; //分解变量的个位
    unsigned char ucAnyNumber_10; //分解变量的十位
    unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
    unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    if(ucWd3Update==1)  //窗口3整屏更新，里面只放那些不用经常刷新显示的内容
    {
        ucWd3Update=0;  //及时清零，避免一直更新
        ucWd3Part1Update=1; //激活窗口3的第1个局部更新显示变量，这里在前面数码管显示框架上有所改进
        ucWd3Part2Update=1; //激活窗口3的第2个局部更新显示变量，这里在前面数码管显示框架上有所改进
        display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
        clear_all_canvas();  //把画布全部清零
        insert_buffer_to_canvas(0,0,Zf816_mao_hao,0,1,16);//把冒号的字模插入画布
        display_lattice(2,0,Hz1616_she,0,2,16,0);     //设置速度。这些内容不用经常更新，只有在切换窗口的
时候才更新显示
        display_lattice(3,0,Hz1616_zhi,0,2,16,0);
        display_lattice(4,0,Hz1616_su,0,2,16,0);
        display_lattice(5,0,Hz1616_du,0,2,16,0);
                display_lattice(0,16,Hz1616_su,0,2,16,0);     //速度上限
        display_lattice(1,16,Hz1616_du,0,2,16,0);
        display_lattice(2,16,Hz1616_shang,0,2,16,0);
        display_lattice(3,16,Hz1616_xian,0,2,16,0);
        display_lattice(8,0,Hz1616_su,0,2,16,0);  //速度下限
        display_lattice(9,0,Hz1616_du,0,2,16,0);
        display_lattice(10,0,Hz1616_xia,0,2,16,0);
        display_lattice(11,0,Hz1616_xian,0,2,16,0);
    }
    if(ucWd3Part1Update==1) //窗口3的第1个局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
        ucWd3Part1Update=0; //及时清零，避免一直更新
        if(ucWd3Part==1) //被选中
        {
            ucCursorFlag=1; //反显 显示
```

```
    }
    else //没被选中
    {
        ucCursorFlag=0; //正常 显示
    }
    if(ucSpeedH>=10) //有2位数以上
    {
        ucAnyNumber_10=ucSpeedH/10;  //十位
    }
    else //否则显示空
    {
        ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
    }
    ucAnyNumber_1=ucSpeedH%10/1;  //个位

    p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
    p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
    insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
    insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
    display_lattice(4,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

}
if(ucWd3Part2Update==1) //窗口3的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
{
     ucWd3Part2Update=0; //及时清零，避免一直更新
     if(ucWd3Part==2) //被选中
     {
         ucCursorFlag=1; //反显 显示
     }
     else //没被选中
     {
         ucCursorFlag=0; //正常 显示
     }
     if(ucSpeedL>=10) //有2位数以上
     {
         ucAnyNumber_10=ucSpeedL/10;  //十位
     }
     else //否则显示空
     {
         ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
     }
     ucAnyNumber_1=ucSpeedL%10/1;  //个位

     p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
     p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
     insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
     insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
     display_lattice(12,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

 }
```

```
}
void wd4(void)  //窗口4 设置频率
{
    unsigned char ucAnyNumber_1; //分解变量的个位
    unsigned char ucAnyNumber_10; //分解变量的十位
    unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
    unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    if(ucWd4Update==1)  //窗口4整屏更新，里面只放那些不用经常刷新显示的内容
    {
        ucWd4Update=0;  //及时清零，避免一直更新
        ucWd4Part1Update=1; //激活窗口4的第1个局部更新显示变量，这里在前面数码管显示框架上有所改进
        ucWd4Part2Update=1; //激活窗口4的第2个局部更新显示变量，这里在前面数码管显示框架上有所改进
        display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
        clear_all_canvas();  //把画布全部清零
        insert_buffer_to_canvas(0,0,Zf816_mao_hao,0,1,16);//把冒号的字模插入画布
        display_lattice(2,0,Hz1616_she,0,2,16,0);     //设置频率。这些内容不用经常更新，只有在切换窗口的
时候才更新显示
        display_lattice(3,0,Hz1616_zhi,0,2,16,0);
        display_lattice(4,0,Hz1616_pin,0,2,16,0);
        display_lattice(5,0,Hz1616_lv,0,2,16,0);
                display_lattice(0,16,Hz1616_pin,0,2,16,0);     //频率上限
        display_lattice(1,16,Hz1616_lv,0,2,16,0);
        display_lattice(2,16,Hz1616_shang,0,2,16,0);
        display_lattice(3,16,Hz1616_xian,0,2,16,0);
        display_lattice(8,0,Hz1616_pin,0,2,16,0);  //频率下限
        display_lattice(9,0,Hz1616_lv,0,2,16,0);
        display_lattice(10,0,Hz1616_xia,0,2,16,0);
        display_lattice(11,0,Hz1616_xian,0,2,16,0);
    }
    if(ucWd4Part1Update==1) //窗口4的第1个局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
        ucWd4Part1Update=0; //及时清零，避免一直更新
        if(ucWd4Part==1) //被选中
        {
            ucCursorFlag=1; //反显 显示
        }
        else //没被选中
        {
            ucCursorFlag=0; //正常 显示
        }
        if(ucFreqH>=10) //有2位数以上
        {
            ucAnyNumber_10=ucFreqH/10;  //十位
        }
        else //否则显示空
        {
            ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
        }
```

```
        ucAnyNumber_1=ucFreqH%10/1;  //个位

        p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
        p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
        insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
        insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
        display_lattice(4,16,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

    }
    if(ucWd4Part2Update==1) //窗口4的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
         ucWd4Part2Update=0; //及时清零，避免一直更新
         if(ucWd4Part==2) //被选中
         {
             ucCursorFlag=1; //反显 显示
         }
         else //没被选中
         {
             ucCursorFlag=0; //正常 显示
         }
         if(ucFreqL>=10) //有2位数以上
         {
             ucAnyNumber_10=ucFreqL/10;  //十位
         }
         else //否则显示空
         {
             ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
         }
         ucAnyNumber_1=ucFreqL%10/1;  //个位

         p_ucAnyNumber_10=number_to_matrix(ucAnyNumber_10); //把数字转换成字模首地址
         p_ucAnyNumber_1=number_to_matrix(ucAnyNumber_1); //把数字转换成字模首地址
         insert_buffer_to_canvas(2,0,p_ucAnyNumber_10,ucCursorFlag,1,16);//把十的字模插入画布
         insert_buffer_to_canvas(3,0,p_ucAnyNumber_1,ucCursorFlag,1,16);//把个的字模插入画布
         display_lattice(12,0,ucCanvasBuffer,0,4,16,0);    //显示整屏的画布,最后的参数0是偏移量

    }

}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                  ucCanvasBuffer[j*4+i]=0x00;
      }
   }
```

```
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释五:
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至3，因为画布的横向只要4个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
                  ucTemp=ucArray[j*x_amount+i];
                  if(ucFbFlag==0)
                  {
             ucCanvasBuffer[(y+j)*4+x+i]=ucTemp; //这里的4代表画布每一行只有4个字节
                  }
                  else
                  {
             ucCanvasBuffer[(y+j)*4+x+i]=~ucTemp; //这里的4代表画布每一行只有4个字节
                  }
      }
   }
```

```
}
/* 注释六:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //         delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
```

```
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
```

}
复制代码
总结陈词:
我前面几节液晶屏程序的字模都是通过外围工具软件生成的，其实这款12864液晶模块本身就是自带字库，编程的时候只要在源代码里直接写入所需要的汉字或者字符，就可以自动调用相对应的字库了。但是细心的网友一定会问，为什么在源代码上直接写入某个汉字就可以调用到这个汉字的字库?在这个过程中，C51编译器到底还干了哪些鲜为人知的好事？欲知详情，请听下回分解-----液晶屏自带字库跟汉字机内码的关系。
（未完待续，下节更精彩，不要走开哦）

第八十节：调用液晶屏内部字库来显示汉字或字符的坐标体系和本质。

开场白:
前面章节讲的内容全部都是用自构字库的，相当于使用液晶屏的图像模式。其实这个款12864液晶屏的驱动芯片是st7920，它内部是自带16x16字库的，可以显示16x16的汉字或者8x16的字符。这一节开始就跟大家讲讲这方面的内容。
要教会大家四个知识点:
第一个：内部字库的真实坐标体系的本质。当我们用内部字库的时候，它的坐标体系跟前面讲的自造字库坐标不一样，不再是256x32的液晶屏。它还原成为128x64的液晶屏，横坐标x轴坐标没办法精确到每个点，只能以16个点(2个字节)为一个单位，因此128个点的x轴坐标范围是0至8。而y轴的坐标也是以16个点(2个字节)为一个单位，因此64个点的x轴坐标范围是0至3。把12864液晶屏分成4行8列，每个数代表一个坐标点。
第二个：在使用内部字库时，C51编译器暗地里干了啥？如果使用液晶屏内部自带字库，编程的时候只要在源代码里直接写入所需要的汉字或者字符，就可以自动调用相对应的字库了。但是细心的网友一定会问，为什么在源代码上直接写入某个汉字就可以调用到这个汉字的字库？其实，表面上我们写下具体的某个汉字或者字符，但是C51编译器会自动对数组内的汉字翻译成 机内码(2字节)，会自动对数组内的字符翻译成 ASCII码(1字节)。
第三个：12864的控制芯片st7920内部有两套驱动显示指令方式，一种是前面章节讲的自构字库模式，也是图像模式。另外一种就是本节讲的用内部字库模式。在切换模式的时候，发送命令字0x0c表示用内部字库模式，发送命令字0x36表示用自构字库模式。
第四个：12864整屏有4行8列，一共32个坐标点，每个坐标点可以显示一个16x16的汉字，但是在显示8x16字符时候，必须一次显示2个字符筹够16x16的点阵。例如，只想达到显示一个字符的时候，应该在另外一个空位置上显示空字符来填充。
具体内容，请看源代码讲解。
（1）硬件平台：基于朱兆祺51单片机学习板。
（2）实现功能:
开机上电后，液晶屏第一行调用直接汉字书写方式的数组来显示(馒头V5)的内容。第四行调用机内码和ASCII码的数组来显示(馒头V5)的内容。
（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(void); // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
void display_hz1616(unsigned int x,unsigned int y,const unsigned char  *ucArray);
void display_double_zf816(unsigned int x,unsigned int y,const unsigned char  *ucArray1,const unsigned char  *ucArray2);
void delay_short(unsigned int uiDelayshort); //延时
/* 注释一：内部字库的真实坐标体系的本质。
```

```
* 当我们用内部字库的时候，它的坐标体系跟前面讲的自造字库坐标不一样，不再是256x32的液晶屏。
* 它还原成为128x64的液晶屏，横坐标x轴坐标没办法精确到每个点，只能以16个点(2个字节)为一个单位，
* 因此128个点的x轴坐标范围是0至8。而y轴的坐标也是以16个点(2个字节)为一个单位，因此64个点的x轴
* 坐标范围是0至3。以下是坐标地址的位置编码。把12864液晶屏分成4行8列，每个数代表一个坐标点，
* 用深究具体含义，液晶驱动芯片ST7920的手册上有提到。
*/
code unsigned char  ucAddrTable[]=  //调用内部字库时，液晶屏的坐标体系，位置编码，是驱动内容，读者可以不用深究它的含义。
{
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,
0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
};
/* 注释二：在使用内部字库时，C51编译器暗地里干了啥?
* 如果使用液晶屏内部自带字库，以下编程的时候只要在源代码里直接写入所需要的汉字或者字符，
* 就可以自动调用相对应的字库了。但是细心的网友一定会问，为什么在源代码上直接写入某个汉字
* 就可以调用到这个汉字的字库？其实，表面上我们写下具体的某个汉字或者字符，但是C51编译器
* 会自动对数组内的汉字翻译成 机内码(2字节)，会自动对数组内的字符翻译成 ASCII码(1字节)。
* 本节程序会做这个实验来验证它。以下两种书写方式不一样，但本质是一样的。
*/
code unsigned char Hz1616_man[]="馒"; //对于数组内的汉字，编译会自动翻译成 机内码(2字节)
code unsigned char JN1616_man[]=  //机内码  馒  网上有很多把汉字或者字符转换成相关编码的工具软件
{
0xC2,
0xF8,
};
code unsigned char Hz1616_tou[]="头"; //对于数组内的汉字，编译会自动翻译成 机内码(2字节)
code unsigned char JN1616_tou[]=  //机内码  头  网上有很多把汉字或者字符转换成相关编码的工具软件
{
0xCD,
0xB7,
};
code unsigned char Zf816_V[]="V";      //对于数组内的字符，编译会自动翻译成 ASCII码(1字节)
code unsigned char ASCII816_V[]= //ASCII码  V  网上有很多把汉字或者字符转换成相关编码的工具软件
{
0x56,
};
code unsigned char Zf816_5[]="5";      //对于数组内的字符，编译会自动翻译成 ASCII码(1字节)
code unsigned char ASCII816_5[]= //ASCII码  5  网上有很多把汉字或者字符转换成相关编码的工具软件
{
0x35,
};
code unsigned char Zf816_nc[]=" ";      //对于数组内的字符，编译会自动翻译成 ASCII码(1字节)
code unsigned char ASCII816_nc[]= //ASCII码  空字符  网上有很多把汉字或者字符转换成相关编码的工具软件
{
0x20,
};
void main()
  {
```

```
        LCDInit(); //初始化12864 内部包含液晶模块的复位
/* 注释三:
* 12864的控制芯片st7920内部有两套驱动显示指令方式，一种是前面章节讲的自构字库模式，也是图像模式。
* 另外一种就是本节讲的用内部字库模式。以下是切换模式的命令，命令字0x0c表示用内部字库模式。
* 命令字0x36表示用自构字库模式。
*/
        WriteCommand(0x0C); //命令字0x0c表示用内部字库模式。命令字0x36表示用自构字库模式。
        display_clear(); // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
        display_hz1616(0,0,Hz1616_man);  //第一行，调用直接汉字书写方式的数组来显示(馒头V5),
        display_hz1616(1,0,Hz1616_tou);
        display_double_zf816(2,0,Zf816_V,Zf816_5);
        display_hz1616(0,3,JN1616_man);  //第四行，调用机内码和ASCII码的数组来显示(馒头V5),
        display_hz1616(1,3,JN1616_tou);
        display_double_zf816(2,3,ASCII816_V,Zf816_5);
        while(1)
        {
           ;
        }
}
/* 注释四: 在一个坐标点显示1个内部字库汉字的函数
* 第1，2个参数x,y是坐标体系。x的范围是0至8，y的范围是0至3.
* 第3个参数*ucArray是汉字机内码，是有2个字节的数组。
*/
void display_hz1616(unsigned int x,unsigned int y,const unsigned char  *ucArray)
{
    WriteCommand(0x30);   //基本指令集
        WriteCommand(ucAddrTable[8*y+x]);        //起始位置
        LCDWriteData(ucArray[0]);
        LCDWriteData(ucArray[1]);
}
/* 注释五: 在一个坐标点显示2个内部字库字符的函数
* 注意，由于一个坐标点是16x16点阵，而一个字符是8x16点阵的，所以务必要显示2个字符筹够1个坐标点。
* 第1，2个参数x,y是坐标体系。x的范围是0至8，y的范围是0至3.
* 第3个参数*ucArray1是左边第1个字符ASCII码，是有1个字节的数组。
* 第4个参数*ucArray2是右边第2个字符ASCII码，是有1个字节的数组。
*/
void display_double_zf816(unsigned int x,unsigned int y,const unsigned char *ucArray1,const unsigned
char  *ucArray2)
{
    WriteCommand(0x30);   //基本指令集
        WriteCommand(ucAddrTable[8*y+x]);        //起始位置
        LCDWriteData(ucArray1[0]);
        LCDWriteData(ucArray2[0]);
}
void display_clear(void) // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
{
    unsigned int i,j;
        for(i=0;i<4;i++)
        {
                for(j=0;j<8;j++)
```

```
                {
                    display_double_zf816(j,i,Zf816_nc,ASCII816_nc);  //Zf816_nc与ASCII816_nc本质是一样的
, 只是书写方式不一样。
                }
        }
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
```

```
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
```

总结陈词:

通过本节的实验，我们发现汉字的识别本质是机内码，字符的识别本质是ASCII码。不管是机内码还是ASCII码，这些都是16进制的数字，也就是我们手机平时接收和发送的信息本质都是这些数字编码，但是机内码是2个字节，ASCII码是1个字节，如果在一串随机的信息中，同时包含汉字和字符两种数字信息，我们的程序又该如何能筛选和识别它们，会不会把机内码和ASCII码搞混乱了？不会的。其实这两种编码都是有规律可以筛选识别的，欲知详情，请听下回分解-----液晶屏显示串口发送过来的任意汉字和字符。

（未完待续，下节更精彩，不要走开哦）

第八十一节：液晶屏显示串口发送过来的任意汉字和字符。

开场白:

通过上一节的学习，我们发现汉字的识别本质是机内码，字符的识别本质是ASCII码。不管是机内码还是ASCII码，这些都是16进制的数字，也就是我们手机平时接收和发送的信息本质都是这些数字编码，但是机内码是2个字节，ASCII码是1个字节，如果在一串随机的信息中，同时包含汉字和字符两种数字信息，我们的程序又该如何能筛选和识别它们，会不会把机内码和ASCII码搞混乱了？这一节要教大家三个知识点:

第一个: ASCII码与汉字机内码不一样的规律是，ASCII码都是小于128(0x80)的，根据这个特点可以编程序把它们区分开来。

第二个: 当任意一串信息中既包含汉字机内码，又包含字符ASCII码时，并且当ASCII码左右相邻个数是以奇数存在的时候，如何巧妙地插入填充空格字符0x20使它们能够符合一个坐标点显示2个字符的要求。

第三个: 本节程序串口部分是在第39节内容基础上移植修改而成，本节程序中多添加了如何通过结束标志0x0D 0x0A来提取有效数据的内容，读者可以学习一下其中的框架。

具体内容，请看源代码讲解。

（1）硬件平台: 基于朱兆祺51单片机学习板。

（2）实现功能:

开机上电后，液晶屏第1行显示“请发送信息”。 任意时刻，从电脑“串口调试助手”根据以下协议要求，发送一串不超过24个汉字或者字符的信息，液晶屏就实时把这些信息显示在第2,3,4行。并且蜂鸣器会鸣叫一声表示数据接收正确。

波特率是: 9600 。

通讯协议: EB 00 55 XX XX XXXX …XX XX 0D 0A

最前面3个字节EB 00 55 表示数据头。

最后面2个字节0D 0A表示信息的结束标志。

中间的XX是机内码和ASCII码信息。比如: 要发送“曹健1人学习51单片机”的信息，它们对应的指令是:

EB 00 55 B2 DC BD A1 31 C8 CB D1 A7 CF B0 3531 B5 A5 C6 AC BB FA 0D 0A

（3）源代码讲解如下:

```
#include "REG52.H"
/* 注释一:
* 本程序的串口那部分内容是从《第三十九节: 判断数据头来接收一串数据的串口通用程序框架。》
* 移植过来的，但是以下要把接收缓冲区的数据从10改成60.同时，协议后面多增加了数据结束标志0x0d 0x0a。
*/
#define const_rc_size  60  //接收串口中断数据的缓冲区数组大小
#define const_receive_time  5  //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完，这个时间根据实际情况来调整大小
#define const_voice_short  40   //蜂鸣器短叫的持续时间
```

```
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
void T0_time(void);  //定时中断函数
void usart_receive(void); //串口接收中断函数
void usart_service(void);  //串口服务程序,在main函数里
void display_service(void); //显示服务程序,在main函数里
void empty_diaplay_buffer(void); //把显示缓冲区全部填充空格字符0x20
void diaplay_all_buffer(void); //显示第2,3,4行全部缓冲区的内容
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(void); // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
void display_double_code(unsigned int x,unsigned int y,const unsigned char ucArray1,const unsigned char ucArray2); //在一个坐标点显示1个汉字或者2个字符的函数
void delay_short(unsigned int uiDelayshort); //延时
code unsigned char  ucAddrTable[]=  //调用内部字库时，液晶屏的坐标体系，位置编码，是驱动内容，读者可以不用深究它的含义。
{
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,
0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
};
code unsigned char JN1616_qing[]=  //机内码  请
{
0xC7,0xEB, //请
};
code unsigned char JN1616_fa[]=  //机内码  发
{
0xB7,0xA2,
};
code unsigned char JN1616_song[]=  //机内码  送
{
0xCB,0xCD,
};
code unsigned char JN1616_xin[]=  //机内码  信
{
0xD0,0xC5,
};
code unsigned char JN1616_xi[]=  //机内码  息
{
```

```
0xCF,0xA2,
};
unsigned int  uiSendCnt=0;      //用来识别串口是否接收完一串数据的计时器
unsigned char ucSendLock=1;     //串口服务程序的自锁变量，每次接收完一串数据只处理一次
unsigned int  uiRcregTotal=0;   //代表当前缓冲区已经接收了多少个数据
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned int  uiRcMoveIndex=0;  //用来解析数据协议的中间变量
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量       1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1个局部更新显示变量  1代表更新显示，响应函数内部会清零
unsigned char ucDispplayBuffer[48]; //第2,3,4行显示内容的缓冲区
void main()
  {
        initial_myself();
        delay_long(100);
        initial_peripheral();
        while(1)
        {
            usart_service();  //串口服务程序
                        display_service(); //显示服务程序
        }
}
/* 注释二: 在一个坐标点显示1个汉字或者2个字符的函数
* 第1，2个参数x,y是坐标体系。x的范围是0至8，y的范围是0至3.
* 第3个参数ucArray1是第1个汉字机内码或者ASCII码。
* 第4个参数ucArray2是第2个汉字机内码或者ASCII码。
*/
void display_double_code(unsigned int x,unsigned int y,const unsigned char ucArray1,const unsigned char
ucArray2)
{
    WriteCommand(0x30);   //基本指令集
    WriteCommand(ucAddrTable[8*y+x]);        //起始位置
    LCDWriteData(ucArray1);
    LCDWriteData(ucArray2);
}
void display_clear(void) // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
{
    unsigned int i,j;
        for(i=0;i<4;i++)
        {
                for(j=0;j<8;j++)
                {
                   display_double_code(j,i,0x20,0x20);  //0x20是空格的ASCII码
                }
        }
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
```

```
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void empty_diaplay_buffer(void) //把显示缓冲区全部填充空格字符0x20
{
   unsigned int i;
   for(i=0;i<48;i++)
   {
      ucDispplayBuffer[i]=0x20; //第2,3,4行显示内容的缓冲区全部填充0x20空格字符
   }
}
void diaplay_all_buffer(void) //显示第2,3,4行全部缓冲区的内容
{
   unsigned int i,j;
```

```
   for(i=0;i<3;i++) //i代表行数
   {
      for(j=0;j<8;j++) //j代表某行的某个坐标在第几列
      {
         display_double_code(j,i+1,ucDispplayBuffer[i*16+j*2],ucDispplayBuffer[i*16+j*2+1]); //这里的
16代表一行可以显示16个字符
      }
   }
}
void display_service(void) //显示服务程序,在main函数里
{
  if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
    {
        ucWd1Update=0;  //及时清零，避免一直更新
        ucWd1Part1Update=1; //激活窗口1的第1个局部更新显示变量
        display_clear(); // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
                //显示第一行固定的内容: 请发送信息
        display_double_code(1,0,JN1616_qing[0],JN1616_qing[1]);      //请
        display_double_code(2,0,JN1616_fa[0],JN1616_fa[1]);          //发
        display_double_code(3,0,JN1616_song[0],JN1616_song[1]);      //送
        display_double_code(4,0,JN1616_xin[0],JN1616_xin[1]);        //信
        display_double_code(5,0,JN1616_xi[0],JN1616_xi[1]);          //息
    }
    if(ucWd1Part1Update==1) //窗口1的第1个局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
        ucWd1Part1Update=0; //及时清零，避免一直更新
        diaplay_all_buffer(); //显示第2,3,4行全部缓冲区的内容
        }
}
/* 注释三:
* 以下有效信息截取和如何判断机内码与ASCII码是本程序的核心，请仔细看讲解。
* 凡是ASCII码都是小于0x80(128)的，根据这个特点可以把ASCII码和机内码分离出来，
* 同时，由于液晶屏的1个坐标必须显示2个编码，对于单个存在的ASCII码，我们要在
* 它的右边多插入一个空格字符0x20。至于如何插入空格0x20字符，请看以下代码。
*/
void usart_service(void)  //串口服务程序,在main函数里
{
    unsigned int i;
        unsigned int uiCodeCnt; //统计接收的有效编码数量
        unsigned int uiCodeYu;  //对uiCodeCnt求2的余数，方便识别是否是1个ASCII码相邻
    if(uiSendCnt>=const_receive_time&&ucSendLock==1) //说明超过了一定的时间内，再也没有新数据从串口来
    {
           ucSendLock=0;    //处理一次就锁起来，不用每次都进来,除非有新接收的数据
           uiRcMoveIndex=0; //由于是判断数据头，所以下标移动变量从数组的0开始向最尾端移动  这个变量是用
来抗干扰处理的
           while(uiRcregTotal>=6&&uiRcMoveIndex<=(uiRcregTotal-6)) //这里的6表示有3个字节的数据头，至少
1个有效数据，2个数据结束标志0x0d 0x0a
           {
if(ucRcregBuf[uiRcMoveIndex+0]==0xeb&&ucRcregBuf[uiRcMoveIndex+1]==0x00&&ucRcregBuf[uiRcMoveIndex+2]==0x
55)  //数据头eb 00 55的判断
```

```
                {
                                empty_diaplay_buffer(); //把显示缓冲区全部填充空格字符0x20
                                    uiCodeCnt=0; //统计接收的有效编码数量清零
                            for(i=0;i<(uiRcregTotal-uiRcMoveIndex-3)&&i<48;i++)//这里的3表示有3个字节的数据头。48表示最大只能接收24个汉字，一共48个字节的机内码.
                                            {
                                    if(ucRcregBuf[uiRcMoveIndex+3+i]==0x0d&&ucRcregBuf[uiRcMoveIndex+4+i]==0x0a)  //结束标志0x0d 0x0a的判断
                                    {
                                         uiVoiceCnt=const_voice_short; //蜂鸣器发出声音,表示数据接收正确完毕
                                                        ucWd1Part1Update=1; //及时更新显示第2,3,4行内容的信息
                                                        break; //退出for循环
                                    }
                                                else  //收集有效信息编码进入显示缓冲区
                                                   {
                                                       uiCodeYu=uiCodeCnt%2; //对2求余数，用来识别相信的2个是否是机内码，否则要进行插入填充0x20处理
                                                           if(uiCodeYu==1)
                                                           {
                                                if(ucRcregBuf[uiRcMoveIndex+3+i]>=0x80&&ucRcregBuf[uiRcMoveIndex+3+i-1]<0x80) //如果当前的是机内码，而上一个不是机内码
                                                                   {
                                                                        ucDispplayBuffer[uiCodeCnt]=0x20; //当前的先填充插入空格字符0x20
                                                                       uiCodeCnt++;    //统计接收的有效编码数量
                                                                   }
                                                           }
                                                       ucDispplayBuffer[uiCodeCnt]=ucRcregBuf[uiRcMoveIndex+3+i]; //收集有效信息编码进入显示缓冲区
                                                       uiCodeCnt++;    //统计接收的有效编码数量
                                                   }
                                            }
                                    break;    //退出while循环
                                }
                                uiRcMoveIndex++; //因为是判断数据头，游标向着数组最尾端的方向移动
                        }

                        uiRcregTotal=0;  //清空缓冲的下标，方便下次重新从0下标开始接受新数据

                }

}
void T0_time(void) interrupt 1    //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(uiSendCnt<const_receive_time)    //如果超过这个时间没有串口数据过来，就认为一串数据已经全部接收完
  {
          uiSendCnt++;     //表面上这个数据不断累加，但是在串口中断里，每接收一个字节它都会被清零，除非这个中间没有串口数据过来
```

```
      ucSendLock=1;      //开自锁标志
  }
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xfe;    //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void usart_receive(void) interrupt 4                    //串口接收数据中断
{
   if(RI==1)
   {
        RI = 0;
            ++uiRcregTotal;
        if(uiRcregTotal>const_rc_size)  //超过缓冲区
        {
           uiRcregTotal=const_rc_size;
        }
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        uiSendCnt=0;  //及时喂狗，虽然main函数那边不断在累加，但是只要串口的数据还没发送完毕，那么它永远
也长不大,因为每个中断都被清零。

   }
   else  //我在其它单片机上都不用else这段代码的，可能在51单片机上多增加" TI = 0;"稳定性会更好吧。
   {
        TI = 0;
   }

}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
```

```
    {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
    }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  //配置定时器
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
  IP =0x10;  //把串口中断设置为最高优先级，必须的。
  TH1=TL1=-(11059200L/12/32/9600);  //这段配置代码具体是什么意思，我也不太清楚，反正是跟串口波特率有关。
  TR1=1;
}
void initial_peripheral(void) //第二区 初始化外围
{
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
   LCDInit(); //初始化12864 内部包含液晶模块的复位
   WriteCommand(0x0C); //命令字0x0c表示用内部字库模式。命令字0x36表示用自构字库模式。
   empty_diaplay_buffer(); //把显示缓冲区全部填充空格字符0x20
}
```

复制代码

总结陈词:

我们现在是调用液晶屏内部字库来显示内容，如果要某行内容反显或者光标闪烁改怎么编程？欲知详情，请听下回分解-----如何在调用液晶屏内部字库时让某行内容反显或者光标闪烁。

（未完待续，下节更精彩，不要走开哦）

第八十二节：如何通过调用液晶屏内部字库把一个任意数值的变量显示出来。

开场白:

本来这一节打算开始讲调用液晶屏内部字库时的反显程序，但是我担心跳跃太大，恐怕很多初学者跟不上，所以多插入这一节讲讲后面菜单程序中经常用到的基本功能，在调用内部字库的情况下，如何把一个任意数值的变量显示在液晶屏上。这一节的功能需求跟前面第76节是一模一样的，只不过前面的不是用自带字库，现在的是用自带字库而已。我们还是需要做一个变量转换成ASCII码的函数，以后只要调用这个转换函数就可以了。这一节就要把这个转换函数和框架思路教给大家。

具体内容，请看源代码讲解。

（1）硬件平台:

基于朱兆祺51单片机学习板。

（2）实现功能：我们定义一个char型的全局变量，把它默认初始化为218，开机上电后，能看到正中间恰好显示这个全局变量的数值218。大家也可以试着更改它的默认初始值，只要不超过char型最大数值255范围，我们就会看到它上电后显示的就是这个初始值。

（3）源代码讲解如下:

```
#include "REG52.H"
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
void initial_myself(void);
void initial_peripheral(void);
void delay_long(unsigned int uiDelaylong);
unsigned char *number_to_ASCII(unsigned char  ucBitNumber);
void display_service(void); //显示服务程序,在main函数里
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(void); // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
void display_double_code(unsigned int x,unsigned int y,const unsigned char ucArray1,const unsigned char ucArray2); //在一个坐标点显示1个汉字或者2个字符的函数
void delay_short(unsigned int uiDelayshort); //延时
code unsigned char  ucAddrTable[]=  //调用内部字库时，液晶屏的坐标体系，位置编码，是驱动内容，读者可以不用深究它的含义。
{
0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,
0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,
0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,
0x98,0x99,0x9a,0x9b,0x9c,0x9d,0x9e,0x9f,
};
code unsigned char ASCII816_0[]="0";   //0  对于数组内的字符，编译会自动翻译成 ASCII码(1字节)
code unsigned char ASCII816_1[]="1";   //1
code unsigned char ASCII816_2[]="2";   //2
code unsigned char ASCII816_3[]="3";   //3
code unsigned char ASCII816_4[]="4";   //4
code unsigned char ASCII816_5[]="5";   //5
code unsigned char ASCII816_6[]="6";   //6
code unsigned char ASCII816_7[]="7";   //7
code unsigned char ASCII816_8[]="8";   //8
code unsigned char ASCII816_9[]="9";   //9
code unsigned char ASCII816_nc[]=" ";  //空格
/* 注释一:
* 以下变量就是本程序的任意变量，网友可以自己更改它的大小来测试本程序，不要超过255.
*/
unsigned char ucAnyNumber=218;  //任意变量默认初始化为218。
unsigned char ucWd1Part1Update=1; //窗口1的第1个局部更新显示变量  1代表更新显示，响应函数内部会清零
void main()
  {
        initial_myself();
        delay_long(100);
        initial_peripheral();
```

```
        while(1)
        {
           display_service(); //显示服务程序
        }
}
/* 注释二: 在一个坐标点显示1个汉字或者2个字符的函数
* 第1，2个参数x,y是坐标体系。x的范围是0至8，y的范围是0至3.
* 第3个参数ucArray1是第1个汉字机内码或者ASCII码。
* 第4个参数ucArray2是第2个汉字机内码或者ASCII码。
*/
void display_double_code(unsigned int x,unsigned int y,const unsigned char ucArray1,const unsigned char
ucArray2)
{
    WriteCommand(0x30);    //基本指令集
    WriteCommand(ucAddrTable[8*y+x]);         //起始位置
    LCDWriteData(ucArray1);
    LCDWriteData(ucArray2);
}
void display_clear(void) // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
{
    unsigned int i,j;
        for(i=0;i<4;i++)
        {
                for(j=0;j<8;j++)
                {
                   display_double_code(j,i,0x20,0x20);  //0x20是空格的ASCII码
                }
        }
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
```

```
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
/* 注释三:
* 本程序的核心转换函数。
* 是可以把一位任意数字变量的函数转换成对应的ASCII码，由于ASCII码放在数组里，所以返回的是指针，代表数组的首地址。
*/
unsigned char *number_to_ASCII(unsigned char  ucBitNumber)
{
        unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的ASCII码。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的ASCII码。
        {
            case 0:
                  p_ucAnyNumber=ASCII816_0;
                  break;
            case 1:
                  p_ucAnyNumber=ASCII816_1;
                  break;
            case 2:
                  p_ucAnyNumber=ASCII816_2;
                  break;
            case 3:
                  p_ucAnyNumber=ASCII816_3;
                  break;
            case 4:
                  p_ucAnyNumber=ASCII816_4;
                  break;
            case 5:
                  p_ucAnyNumber=ASCII816_5;
                  break;
```

```
            case 6:
                  p_ucAnyNumber=ASCII816_6;
                  break;
            case 7:
                  p_ucAnyNumber=ASCII816_7;
                  break;
            case 8:
                  p_ucAnyNumber=ASCII816_8;
                  break;
            case 9:
                  p_ucAnyNumber=ASCII816_9;
                  break;
            case 10:
                  p_ucAnyNumber=ASCII816_nc;
                  break;
            default:   //如果上面的条件都不符合，那么默认指向空格ASCII码
                  p_ucAnyNumber=ASCII816_nc;
                  break;
        }
        return p_ucAnyNumber;  //返回转换结束后的指针
}
void display_service(void) //显示服务程序,在main函数里
{
/* 注释四:
* 这里的局部变量用static关键词修饰，是因为这个函数一直在主函数while(1)里循环扫描，我不希望它每次进来这个
函数
* 都多花几条指令去初始化这些局部变量，这样会多耗掉几个指令，所以我就用static关键字避免了这种情况，让这些
局部变量
* 只在上电那一刻就初始化了，以后每次进来这个函数不用再初始化这些变量。
*/
    static unsigned char ucAnyNumber_1; //分解变量的个位
    static unsigned char ucAnyNumber_10; //分解变量的十位
    static unsigned char ucAnyNumber_100; //分解变量的百位
    static unsigned char *p_ucAnyNumber_1; //经过数字转换成字模后，分解变量的个位字模首地址
    static unsigned char *p_ucAnyNumber_10; //经过数字转换成字模后，分解变量的十位字模首地址
    static unsigned char *p_ucAnyNumber_100; //经过数字转换成字模后，分解变量的百位字模首地址
    if(ucWd1Part1Update==1) //窗口1的第1个局部更新显示变量,里面放一些经常需要刷新显示的内容
    {
        ucWd1Part1Update=0; //及时清零，避免一直更新
        if(ucAnyNumber>=100) //有3位数以上
        {
           ucAnyNumber_100=ucAnyNumber/100; //百位
        }
        else //否则显示空
        {
           ucAnyNumber_100=10;  //在下面的转换函数中，代码10表示空字模
        }
        if(ucAnyNumber>=10) //有2位数以上
        {
           ucAnyNumber_10=ucAnyNumber%100/10;  //十位
```

```
        }
        else //否则显示空
        {
           ucAnyNumber_10=10;  //在下面的转换函数中，代码10表示空字模
        }
        ucAnyNumber_1=ucAnyNumber%10/1;  //个位
        p_ucAnyNumber_100=number_to_ASCII(ucAnyNumber_100); //把数字转换成字符ASCII码
        p_ucAnyNumber_10=number_to_ASCII(ucAnyNumber_10);   //把数字转换成字符ASCII码
        p_ucAnyNumber_1=number_to_ASCII(ucAnyNumber_1);     //把数字转换成字符ASCII码
        display_double_code(2,1,ASCII816_nc[0],p_ucAnyNumber_100[0]);//液晶屏的显示驱动函数  这里的
ASCII816_nc[0]代表填充显示一个空格字符
        display_double_code(3,1,p_ucAnyNumber_10[0],p_ucAnyNumber_1[0]);//液晶屏的显示驱动函数
    }
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
}
void initial_peripheral(void) //第二区 初始化外围
{
   LCDInit(); //初始化12864 内部包含液晶模块的复位
   WriteCommand(0x0C); //命令字0x0c表示用内部字库模式。命令字0x36表示用自构字库模式。
   display_clear(); // 清屏。4行8列的坐标点全部显示2个空字符相当于清屏了。
}
```

总结陈词:

在液晶屏程序里，经常要用到反显的功能来表示选中某一项菜单。在调用内部字库时，这样的驱动程序又该怎么写？欲知详情，请听下回分解-----如何在调用液晶屏内部字库时让某行内容反显。

（未完待续，下节更精彩，不要走开哦）

第八十三节：矩阵键盘输入任意数字或小数点的液晶屏显示程序。

开场白：

本来这节打算讲调用液晶屏内部字库时让某行内容反显的，但是在昨天调试过程中，发现一个很奇怪的问题，当调用内部字库时，按照数据手册，我执行一条反显指令时，应该是仅仅某一行反显，但是却同时出现两行反显。比如，当我执行

```
WriteCommand(0x34); //扩充指令集
WriteCommand(0x04); //第1行反显
```

指令时，发现第一行和第三行反显，后来想想，我猜测这种12864的屏应该是25632折成左右半屏，左半屏在上面，右半屏在下面。经过这次经验，我觉得大家以后尽量不要用液晶屏的内部字库模式，应该用自构字库的模式（图形模式）。因为我觉得用内部字库模式的时候，这个集成的反显扩展指令不好用。而用自构字库的模式（图形模式），却可以顺心所欲的灵活运用，适合做菜单程序。

既然发现内部字库不好用，所以不再讲内部字库模式，这节仅仅接着前面第79节内容，继续讲在自构字库的模式（图形模式）下，如何通过矩阵键盘直接输入数字和小数点，就像普通的计算器一样键盘输入。这个功能表面简单，其实有以下四个地方值得注意：

第一：如何用数组接收按键输入的BCD码数据。

第二：如何限制输入参数的小数点个数和数组的有效个数。

第三：如果第0个位置是0，那么继续输入的数据直接覆盖0，否则就移位再输入。

第四：如果第0个位置是0，那么继续输入的小数点要移位输入。

要仔细了解以上提到的关键点，必须好好研究本程序中的void set_data(...)函数。同时也要温习一下之前讲的自构字库模式的液晶屏显示内容，尤其是插入画布显示的内容。

具体内容，请看源代码讲解。

（1）　　硬件平台：

基于朱兆祺51单片机学习板。数字1键对应S1键，数字2键对应S2键，数字3键对应S3键…. 数字9键对应S9键，数字0键对应S10键。小数键对应S11，清零键对应S16，其它按键不用。

（2）　　实现功能：

用矩阵键盘输入任意数字或小数点。小数点不能超过2位，一旦超过2位，再按其它按键则输入无效。有效数字也不能超过6位（包括小数点），一旦超过6位，再按其它按键则输入无效。

想重新输入，必须按S16清零按键才能重新输入。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  10    //按键去抖动延时的时间
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
```

```
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00  全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void key_number_input(unsigned char ucKeyNumber); //输入数字按键
void set_data(unsigned char ucKeyNumberTemp,unsigned char ucDotBitMax,unsigned char
ucDataCntMax,unsigned char *p_ucDotCnt,unsigned char *p_ucDotBitS,unsigned char *p_ucWdPartCnt,unsigned
char *p_ucSetDataBuffer);
void key_delete_input(void); //删除按键
void T0_time(); //定时中断函数
void key_service();
void key_scan(); //按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
```

```
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Zf816_dot[]=  //小数点
{
/*--  文字:  .  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x60,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_yi[]=
{
/*--  文字:  一  --*/
```

```
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_xiang[]=
{
/*--  文字:  项  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x03,0xFE,0xFC,0x20,0x10,0x40,0x11,0xFC,0x11,0x04,0x11,0x24,0x11,0x24,
0x11,0x24,0x11,0x24,0x1D,0x24,0xE1,0x34,0x00,0x48,0x01,0x86,0x06,0x02,0x00,0x00,
};
code unsigned char Hz1616_shu[]=
{
/*--  文字:  数  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x20,0x49,0x30,0x2A,0x20,0x1C,0x20,0xFF,0x7E,0x1C,0x44,0x2B,0x44,0x48,0xC4,
0x08,0x28,0xFF,0x28,0x12,0x10,0x34,0x10,0x0C,0x28,0x32,0x4E,0xC0,0x84,0x00,0x00,
};
code unsigned char Hz1616_zhu[]=
{
/*--  文字:  组  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x00,0x19,0xF8,0x11,0x08,0x25,0x08,0x25,0x08,0x79,0xF8,0x09,0x08,0x11,0x08,
0x21,0x08,0x7D,0xF8,0x01,0x08,0x01,0x08,0x0D,0x08,0x73,0xFE,0x00,0x00,0x00,0x00,
};
/* 注释一:
* 以下是画布显示数组。横向是6个字节，纵向16行，可以显示3个16x16的汉字.
*  注意，这节内容的画布跟前面章节的画布大小不一样，前面章节的横向是4个字节，这节的横向是6个字节。
*/
unsigned char ucCanvasBuffer[]=
{
0x00,0x00,0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
};
/* 注释二:
```

```
* 以下4个变量记录一个参数的4种信息，包括小数点的数量，个数，数据的位置，数组具体值.
*/
unsigned char ucDotCnt_1=0;  //记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效
unsigned char ucDotBitS_1=0; //记录当前输入的小数点个数，如果小数点的个量如果超过规定2位，此时再按任何输入按键则无效
unsigned char ucWdPartCnt_1=0; //记录当前输入的数据在数组中的位置。
unsigned char ucDataBuffer_1[6]={0,10,10,10,10,10}; //一项的BCD码数组缓冲
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucPart=1; //局部变量 0代表没有选中任何一行，其它数值1到4代表选中某一行
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量       1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会清零
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                    key_service(); //按键服务程序
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan();//按键扫描函数 放在定时中断里
  if(uiVoiceCnt!=0)
  {
```

```
    uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
    ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
         beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
    case 1:   //按键扫描输出第ucRowRecord列低电平
             if(ucRowRecord==1)  //第一列输出低电平
                 {
            key_dr1=0;
            key_dr2=1;
            key_dr3=1;
            key_dr4=1;
                 }
             else if(ucRowRecord==2)  //第二列输出低电平
                 {
            key_dr1=1;
            key_dr2=0;
            key_dr3=1;
            key_dr4=1;
                 }
             else if(ucRowRecord==3)  //第三列输出低电平
                 {
            key_dr1=1;
            key_dr2=1;
            key_dr3=0;
            key_dr4=1;
                 }
             else   //第四列输出低电平
                 {
            key_dr1=1;
            key_dr2=1;
            key_dr3=1;
            key_dr4=0;
                 }
         uiKeyTimeCnt=0;  //延时计数器清零
         ucKeyStep++;     //切换到下一个运行步骤
             break;
    case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
         uiKeyTimeCnt++;
```

```
            if(uiKeyTimeCnt>1)
            {
               uiKeyTimeCnt=0;
        ucKeyStep++;      //切换到下一个运行步骤
            }
         break;
case 3:
     if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
     {
        ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
        ucKeyLock=0;  //按键自锁标志清零
        uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                ucRowRecord++;  //输出下一列
                if(ucRowRecord>4)
                {
                   ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                }
     }
            else if(ucKeyLock==0)  //有按键按下，且是第一次触发
            {
               if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                   {
                      uiKeyTimeCnt++;  //去抖动延时计数器
                          if(uiKeyTimeCnt>const_key_time)
                          {
                             uiKeyTimeCnt=0;
                             ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                 if(ucRowRecord==1)  //第一列输出低电平
                     {
                                ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                      }
                 else if(ucRowRecord==2)  //第二列输出低电平
                     {
                                ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                      }
                 else if(ucRowRecord==3)  //第三列输出低电平
                     {
                                ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                      }
                 else   //第四列输出低电平
                     {
                                ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                      }
                          }

                   }
               else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                   {
```

```
            uiKeyTimeCnt++;  //去抖动延时计数器
                if(uiKeyTimeCnt>const_key_time)
                {
                   uiKeyTimeCnt=0;
                   ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
      }
  else   //第四列输出低电平
      {
                ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
      }
           }

    }
else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
    {
       uiKeyTimeCnt++;  //去抖动延时计数器
           if(uiKeyTimeCnt>const_key_time)
           {
              uiKeyTimeCnt=0;
              ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S10键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
      }
  else   //第四列输出低电平
      {
                ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
      }
           }
```

```
                        }
                    else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                      if(ucRowRecord==1)  //第一列输出低电平
                          {
                                    ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                          }
                      else if(ucRowRecord==2)  //第二列输出低电平
                          {
                                    ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                          }
                      else if(ucRowRecord==3)  //第三列输出低电平
                          {
                                    ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                          }
                      else   //第四列输出低电平
                          {
                                    ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                          }
                               }

                      }

                }
            break;
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 数字1 对应朱兆祺学习板的S1键
          key_number_input(1); //输入数字按键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 数字2 对应朱兆祺学习板的S2键
          key_number_input(2); //输入数字按键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 数字3 对应朱兆祺学习板的S3键
          key_number_input(3); //输入数字按键
```

```
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 数字4 对应朱兆祺学习板的S4键
        key_number_input(4); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 5:// 数字5 对应朱兆祺学习板的S5键
        key_number_input(5); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 6:// 数字6 对应朱兆祺学习板的S6键
        key_number_input(6); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 7:// 数字7 对应朱兆祺学习板的S7键
        key_number_input(7); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 8: //数字8 对应朱兆祺学习板的S8键
        key_number_input(8); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 9:// 数字9 对应朱兆祺学习板的S9键
        key_number_input(9); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 10:// 数字0  对应朱兆祺学习板的S10键
        key_number_input(0); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 11:// 小数点按键 对应朱兆祺学习板的S11键
        key_number_input(11); //输入数字按键  11代表小数点
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 12:// 本节暂时不用 对应朱兆祺学习板的S12键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 13:// 本节暂时不用 对应朱兆祺学习板的S13键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
```

```
            break;
    case 14:// 本节暂时不用  对应朱兆祺学习板的S14键

            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 15:// 本节暂时不用 对应朱兆祺学习板的S15键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 16:// 清除按键 对应朱兆祺学习板的S16键
            key_delete_input(); //删除按键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
  }
}
void key_number_input(unsigned char ucKeyNumber) //输入数字按键
{
        switch(ucWd)
        {
          case 1:    //第1窗口。本节程序只有1个窗口
                    switch(ucPart)
                    {
              case 1:  //1窗口第1项
                     set_data(ucKeyNumber,2,6,&ucDotCnt_1,&ucDotBitS_1,&ucWdPartCnt_1,ucDataBuffer_1);
//设置参数，请看本函数具体内容。本节的核心内容，值得好好研究!
                                    ucWd1Part1Update=1;//更新显示
                                    break;
          }

                     break;
    }

}
/* 注释三:
* 本节的核心函数，值得好好研究!
* 涉及到参数的4种信息，包括小数点的数量，个数，数据的位置，数组具体值。以及它们之间的相互作用关系。
* 以下参数，指针类型的参数是让代入的全局变量在退出函数后维持它当前最新更改的数值不变。
* 第1个参数ucKeyNumberTemp是当前按键输入的数值。
* 第2个参数ucDotBitMax是限定被设置参数的小数点最大位数。
* 第3个参数ucDataCntMax是限定被设置参数的最大数组个数。
* 第4个参数*p_ucDotCnt是记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时
再按小数点按键则无效。
* 第5个参数*p_ucDotBitS是记录当前输入的小数点个数，如果小数点的个量如果超过规定2位，此时再按任何输入按键
则无效
* 第6个参数*p_ucWdPartCnt是记录当前输入的数据在数组中的位置，方便锁定每次按键输入的数字显示位置。
* 第7个参数*p_ucSetDataBuffer是BCD码数组缓冲的具体数字内容。
*/
void set_data(unsigned char ucKeyNumberTemp,unsigned char ucDotBitMax,unsigned char
```

```
ucDataCntMax,unsigned char *p_ucDotCnt,unsigned char *p_ucDotBitS,unsigned char *p_ucWdPartCnt,unsigned char *p_ucSetDataBuffer)
{
                    unsigned int i;
                    if(ucKeyNumberTemp==11) //等于小数点
                    {
                       if(ucDotBitMax==0) //如果限定的小数点最大数是0，就意味着此数据不允许带小数点，必须是整数。
                       {
                           return; //直接返回退出
                       }
                       else if(*p_ucDotCnt>0)  //小数点个数大于0，意味着当前数组已经包含了小数点，此时再输入小数点则无效。
                       {
                           return; //直接返回退出
                       }
                       else  //否则有效，记录当前已经包含一个小数点的信息。
                       {
                           *p_ucDotCnt=1;  //只能包含一个小数点
                       }
                    }
                    else if(*p_ucDotCnt==1) //如果输入的不是小数点，并且之前已经输入了一个小数点，那么此时输入的数字就是小数点后的数据
                    {
                        if(*p_ucDotBitS<ucDotBitMax) //如果小数点位数还没超过最大限制位数，则继续加1记录当前小数点位数。
                        {
                            *p_ucDotBitS=(*p_ucDotBitS)+1;
                        }
                        else //如果小数点位数已经超过允许的范围，则输入的按键无效，直接退出。
                        {
                            return; //直接返回退出
                        }
                    }

                                    if(*p_ucWdPartCnt<(ucDataCntMax-1))  //当输入的有效BCD码不超过最大数组缓冲时
                                    {
                       if(*p_ucWdPartCnt==0&&p_ucSetDataBuffer[0]==0&&ucKeyNumberTemp!=11)  //如果当前默认位置是第0个位置，并且默认第0个数据是0，并且当前的按键输入不是小数点，则不用移位
                       {
                           ;
                       }
                       else  //否则，移位
                       {
                           for(i=0;i<(ucDataCntMax-1);i++)  //移位
                           {
                              p_ucSetDataBuffer[ucDataCntMax-1-i]=p_ucSetDataBuffer[ucDataCntMax-2-i];
                           }
                                                *p_ucWdPartCnt=(*p_ucWdPartCnt)+1;
```

```
                        }
                        p_ucSetDataBuffer[0]=ucKeyNumberTemp; //当前输入的数字或者小数点永远在第右边第0个
位置。

                    }
}
void key_delete_input(void) //删除按键
{
        static unsigned int i;
        switch(ucWd)
        {
          case 1:    //第1窗口。本节程序只有1个窗口
                    switch(ucPart)
                    {
             case 1:  //1窗口第1项

                                //清零
                     ucDotBitS_1=0;
                     ucDotCnt_1=0;
                     ucWdPartCnt_1=0;
                     for(i=0;i<6;i++)
                     {
                        ucDataBuffer_1[i]=10;
                     }
                                    ucDataBuffer_1[0]=0; //第0个位置填入0

                                ucWd1Part1Update=1;//更新显示
                                break;

          }

                     break;

    }

}
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                     break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                     break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                     break;
```

```
            case 3:
             p_ucAnyNumber=Zf816_3;
                    break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                    break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                    break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                    break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                    break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                    break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                    break;
            case 10:  //空格
             p_ucAnyNumber=Zf816_nc;
                    break;
                       case 11:    //小数点
             p_ucAnyNumber=Zf816_dot;
                    break;
               default:    //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                    break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    static unsigned char *p_ucAnyNumber; //经过数字转换成字模后，分解变量的某位字模首地址
    static unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    static unsigned int i;
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:    //显示窗口1的数据
               if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
               {
                    ucWd1Update=0;  //及时清零，避免一直更新
                    ucWd1Part1Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
                    clear_all_canvas();  //把画布全部清零
                    display_lattice(0,0,Hz1616_yi,0,2,16,0);     //一窗口一行，这些内容不用经常更新，只
有在切换窗口的时候才更新显示
```

```
                    display_lattice(1,0,Hz1616_xiang,0,2,16,0);
                    display_lattice(2,0,Hz1616_shu,0,2,16,0);
                    display_lattice(3,0,Hz1616_zhu,0,2,16,0);
                    display_lattice(4,0,Zf816_mao_hao,0,1,16,0); //冒号

                }
                if(ucWd1Part1Update==1) //窗口1的第1行局部更新显示变量,里面放一些经常需要刷新显示的内容
                {
                        ucWd1Part1Update=0; //及时清零，避免一直更新
                        if(ucPart==1) //被选中
                        {
                           ucCursorFlag=1; //反显 显示
                        }
                        else //没被选中
                        {
                            ucCursorFlag=0; //正常 显示
                        }

                                                for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                        {
                                                    p_ucAnyNumber=number_to_matrix(ucDataBuffer_1[5-i]);
                          insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地址
，一共可以显示6个字符，因此取值范围是0到5
                        }
                        display_lattice(5,0,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最后
的参数0是偏移量
                }

                break;
        //本程序只有1个窗口，所以只有一个case 1，如果要增加窗口，就直接增加 case 2，case 3...
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                  ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
```

```
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释四:
* 注意，这节内容的画布跟前面章节的画布大小不一样，前面章节的横向是4个字节，这节的横向是6个字节。
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至5，因为画布的横向只要6个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
               ucTemp=ucArray[j*x_amount+i];
               if(ucFbFlag==0)
               {
                  ucCanvasBuffer[(y+j)*6+x+i]=ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向
是4个字节，要稍微注意的。
               }
               else
               {
                  ucCanvasBuffer[(y+j)*6+x+i]=~ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向
是4个字节，要稍微注意的。
               }
      }
   }
}
/* 注释五:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
```

```
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
```

```
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

总结陈词:

这节讲的是键盘输入数字或者小数点的BCD码用来显示，实际项目中，我们经常要知道所输入的BCD码数组到底有效数值

是多少，这个该怎么办？欲知详情，请听下回分解----

第八十四节：实时同步把键盘输入的BCD码数组转换成数值的液晶屏显示程序。

开场白：

键盘直接输入的是带小数点的BCD码数组，要把它们转换成具体的数值才可以更好的在程序里运算或者处理。如何把BCD码数组实时同步转换成数值？这一节主要跟大家讲这方面的算法程序。另外，有一个地方值得注意：上一节键盘输入的小数点个数可以限制成最大2位，但是整数部分没有限制。这节为了也能限制整数部分的最大个数为3位，我修改了上一节的void set_data(...)函数。所以这节的void set_data(...)函数跟上一节的void set_data(...)函数有点不一样，需要特别注意。

具体内容，请看源代码讲解。

（1）　　硬件平台：

基于朱兆祺51单片机学习板。数字1键对应S1键，数字2键对应S2键，数字3键对应S3键…. 数字9键对应S9键，数字0键对应S10键。小数键对应S11，清零键对应S16，其它按键不用。

（2）　　实现功能：

用矩阵键盘输入任意数字或小数点。小数点不能超过2位，一旦超过2位，再按其它按键则输入无效。整数部分不能超过3位，一旦超过3位，再按其它按键则输入无效。想重新输入，必须按S16清零按键才能重新输入。每次键盘输入的第一行BCD码数组会同步更新显示在第二行的数值上。

（3）源代码讲解如下：

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  10    //按键去抖动延时的时间
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void key_number_input(unsigned char ucKeyNumber); //输入数字按键
void set_data(unsigned char ucKeyNumberTemp, //设置参数
```

```
                unsigned char ucDotBitMax,
                unsigned char ucDataCntMax,
                unsigned char *p_ucDotCnt,
                unsigned char *p_ucDotBitS,
                            unsigned char *p_ucWdPartCnt,
                            unsigned char *p_ucSetDataBuffer,
                            unsigned char ucIntCntMax,
                            unsigned char *p_ucIntCnt);
unsigned long buffer_to_data(unsigned char ucConverDataSize,unsigned char ucConverDotCnt,unsigned char
*p_ucConverBuffer); //把带小数点的BCD数组转换成long类型的数值。
void key_delete_input(void); //删除按键
void T0_time(); //定时中断函数
void key_service();
void key_scan(); //按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
```

```
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Zf816_dot[]=  //小数点
{
/*--  文字:  .  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x60,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_yi[]=
{
/*--  文字:  一  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
};
code unsigned char Hz1616_xiang[]=
{
/*--  文字:  项  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x03,0xFE,0xFC,0x20,0x10,0x40,0x11,0xFC,0x11,0x04,0x11,0x24,0x11,0x24,
0x11,0x24,0x11,0x24,0x1D,0x24,0xE1,0x34,0x00,0x48,0x01,0x86,0x06,0x02,0x00,0x00,
};
code unsigned char Hz1616_shu[]=
{
/*--  文字:  数  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x20,0x49,0x30,0x2A,0x20,0x1C,0x20,0xFF,0x7E,0x1C,0x44,0x2B,0x44,0x48,0xC4,
0x08,0x28,0xFF,0x28,0x12,0x10,0x34,0x10,0x0C,0x28,0x32,0x4E,0xC0,0x84,0x00,0x00,
};
code unsigned char Hz1616_zhu[]=
{
/*--  文字:  组  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x00,0x19,0xF8,0x11,0x08,0x25,0x08,0x25,0x08,0x79,0xF8,0x09,0x08,0x11,0x08,
0x21,0x08,0x7D,0xF8,0x01,0x08,0x01,0x08,0x0D,0x08,0x73,0xFE,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_zhi[]=
{
/*--  文字:  值  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x40,0x18,0x60,0x17,0xFC,0x10,0x40,0x20,0x80,0x33,0xF8,0x62,0x08,0xA3,0xF8,
0x22,0x08,0x23,0xF8,0x22,0x08,0x23,0xF8,0x22,0x08,0x22,0x08,0x2F,0xFE,0x20,0x00,
};
/* 注释一:
* 以下是画布显示数组。横向是6个字节，纵向16行，可以显示3个16x16的汉字.
*  注意，这节内容的画布跟前面79章节的画布大小不一样，79节前面的横向是4个字节，这节的横向是6个字节。
*/
unsigned char ucCanvasBuffer[]=
{
0x00,0x00,0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
};
/* 注释二:
* 以下5个变量记录一个参数的5种信息，包括小数点的数量，小数点个数，数据的位置，数组具体值，整数个数
*/
unsigned char ucDotCnt_1=0;  //记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效
unsigned char ucDotBitS_1=0; //记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任何输入按键则无效
unsigned char ucWdPartCnt_1=0; //记录当前输入的数据在数组中的位置。
unsigned char ucDataBuffer_1[6]={0,10,10,10,10,10}; //一项的BCD码数组缓冲
unsigned char ucIntCnt_1=0; //记录当前输入的整数个数，如果整数的个数如果超过规定ucIntCntMax位，此时再按任何输入按键则无效
unsigned long ulData_1=0; //用一个long变量表示BCD码的具体数值。
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucPart=1; //局部变量 0代表没有选中任何一行，其它数值1到4代表选中某一行
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量       1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
unsigned char ucWd1Part2Update=0; //窗口1的第2行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                    key_service(); //按键服务程序
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;      //开总中断
```

```
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan();//按键扫描函数 放在定时中断里
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;    //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
     case 1:   //按键扫描输出第ucRowRecord列低电平
               if(ucRowRecord==1)  //第一列输出低电平
                   {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==2)  //第二列输出低电平
                   {
              key_dr1=1;
              key_dr2=0;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==3)  //第三列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=0;
              key_dr4=1;
                   }
               else   //第四列输出低电平
                   {
```

```
            key_dr1=1;
            key_dr2=1;
            key_dr3=1;
            key_dr4=0;
                 }
          uiKeyTimeCnt=0;  //延时计数器清零
          ucKeyStep++;     //切换到下一个运行步骤
              break;
    case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
          uiKeyTimeCnt++;
                 if(uiKeyTimeCnt>1)
                 {
                    uiKeyTimeCnt=0;
            ucKeyStep++;     //切换到下一个运行步骤
                 }
              break;
    case 3:
          if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
          {
            ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
            ucKeyLock=0;  //按键自锁标志清零
            uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                        ucRowRecord++;  //输出下一列
                        if(ucRowRecord>4)
                        {
                           ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                        }
          }
                 else if(ucKeyLock==0)  //有按键按下，且是第一次触发
                 {
                    if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                        if(ucRowRecord==1)  //第一列输出低电平
                            {
                                       ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                            }
                        else if(ucRowRecord==2)  //第二列输出低电平
                            {
                                       ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                            }
                        else if(ucRowRecord==3)  //第三列输出低电平
                            {
                                       ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
```

```
        }
    else   //第四列输出低电平
        {
                ucKeySec=4;   //触发4号键 对应朱兆祺学习板的S4键
        }
            }

      }
  else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
      {
        uiKeyTimeCnt++;   //去抖动延时计数器
            if(uiKeyTimeCnt>const_key_time)
            {
               uiKeyTimeCnt=0;
               ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
    if(ucRowRecord==1)  //第一列输出低电平
        {
                ucKeySec=5;   //触发5号键 对应朱兆祺学习板的S5键
        }
    else if(ucRowRecord==2)  //第二列输出低电平
        {
                ucKeySec=6;   //触发6号键 对应朱兆祺学习板的S6键
        }
    else if(ucRowRecord==3)  //第三列输出低电平
        {
                ucKeySec=7;   //触发7号键 对应朱兆祺学习板的S7键
        }
    else   //第四列输出低电平
        {
                ucKeySec=8;   //触发8号键 对应朱兆祺学习板的S8键
        }
            }

      }
  else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
      {
        uiKeyTimeCnt++;   //去抖动延时计数器
            if(uiKeyTimeCnt>const_key_time)
            {
               uiKeyTimeCnt=0;
               ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
    if(ucRowRecord==1)  //第一列输出低电平
        {
                ucKeySec=9;   //触发9号键 对应朱兆祺学习板的S9键
        }
    else if(ucRowRecord==2)  //第二列输出低电平
        {
                ucKeySec=10;   //触发10号键 对应朱兆祺学习板的S10键
```

```
                        }
                    else if(ucRowRecord==3)  //第三列输出低电平
                        {
                                ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
                        }
                    else   //第四列输出低电平
                        {
                                ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
                        }
                            }

                      }
                  else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                      {
                         uiKeyTimeCnt++;  //去抖动延时计数器
                             if(uiKeyTimeCnt>const_key_time)
                             {
                                uiKeyTimeCnt=0;
                                ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                    if(ucRowRecord==1)  //第一列输出低电平
                        {
                                ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                        }
                    else if(ucRowRecord==2)  //第二列输出低电平
                        {
                                ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                        }
                    else if(ucRowRecord==3)  //第三列输出低电平
                        {
                                ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                        }
                    else   //第四列输出低电平
                        {
                                ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                        }
                            }

                      }

                }
            break;
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 数字1 对应朱兆祺学习板的S1键
          key_number_input(1); //输入数字按键
```

```
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 2:// 数字2 对应朱兆祺学习板的S2键
        key_number_input(2); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 数字3 对应朱兆祺学习板的S3键
        key_number_input(3); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 数字4 对应朱兆祺学习板的S4键
        key_number_input(4); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 5:// 数字5 对应朱兆祺学习板的S5键
        key_number_input(5); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 6:// 数字6 对应朱兆祺学习板的S6键
        key_number_input(6); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 7:// 数字7 对应朱兆祺学习板的S7键
        key_number_input(7); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 8: //数字8 对应朱兆祺学习板的S8键
        key_number_input(8); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 9:// 数字9 对应朱兆祺学习板的S9键
        key_number_input(9); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 10:// 数字0  对应朱兆祺学习板的S10键
        key_number_input(0); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 11:// 小数点按键 对应朱兆祺学习板的S11键
        key_number_input(11); //输入数字按键  11代表小数点
```

```
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 12:// 本节暂时不用 对应朱兆祺学习板的S12键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 13:// 本节暂时不用 对应朱兆祺学习板的S13键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 14:// 本节暂时不用  对应朱兆祺学习板的S14键

            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 15:// 本节暂时不用 对应朱兆祺学习板的S15键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 16:// 清除按键 对应朱兆祺学习板的S16键
            key_delete_input(); //删除按键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
  }
}
void key_number_input(unsigned char ucKeyNumber) //输入数字按键
{
    switch(ucWd)
    {
       case 1:   //第1窗口。本节程序只有1个窗口
             switch(ucPart)
             {
                 case 1:  //1窗口第1项
                      set_data(ucKeyNumber,  //本函数跟前面第83节内容有所改动，请看本函数具体内容。本节的核心内容，值得好好研究!
                                                    2,  //小数点最大个数
                                                        6,  //数组缓冲最大个数
                                                    &ucDotCnt_1,
                                                        &ucDotBitS_1,
                                                        &ucWdPartCnt_1,
                                                        ucDataBuffer_1,
                                                        3, //整数部分的最大个数
                                                        &ucIntCnt_1);
                                              ulData_1=buffer_to_data(6,2,ucDataBuffer_1); //把带小数点的BCD码数组转换成long数值。
                      ucWd1Part1Update=1;//第一行局部更新显示
                                              ucWd1Part2Update=1;//第二行局部更新显示
                      break;
```

```
            }

            break;
    }

}
/* 注释三:
* 本函数在前面第83节内容的函数上有改动，为了限制整数部分的个数，多添加了第8和第9这两个参数。
* 本节的核心函数，值得好好研究!
* 涉及到参数的4种信息，包括小数点的数量，小数点的个数，数据的位置，数组具体值，整数的数量，整数的个数
，以及它们之间的相互作用关系。
* 以下参数，指针类型的参数是让代入的全局变量在退出函数后维持它当前最新更改的数值不变。
* 第1个参数ucKeyNumberTemp是当前按键输入的数值。
* 第2个参数ucDotBitMax是限定被设置参数的小数点最大位数。
* 第3个参数ucDataCntMax是限定被设置参数的最大数组个数。
* 第4个参数*p_ucDotCnt是记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时
再按小数点按键则无效。
* 第5个参数*p_ucDotBitS是记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任
何输入按键则无效
* 第6个参数*p_ucWdPartCnt是记录当前输入的数据在数组中的位置，方便锁定每次按键输入的数字显示位置。
* 第7个参数*p_ucSetDataBuffer是BCD码数组缓冲的具体数字内容。
* 第8个参数ucIntCntMax是限定被设置参数的整数部分的最大位数。
* 第9个参数*p_ucIntCnt是记录当前输入的整数部分个数，如果整数部分的个数如果超过规定ucIntCntMax位，此时再
按任何输入按键则无效
*/
void set_data(unsigned char ucKeyNumberTemp,
              unsigned char ucDotBitMax,
              unsigned char ucDataCntMax,
              unsigned char *p_ucDotCnt,
              unsigned char *p_ucDotBitS,
                          unsigned char *p_ucWdPartCnt,
                          unsigned char *p_ucSetDataBuffer,
                          unsigned char ucIntCntMax,
                          unsigned char *p_ucIntCnt)
{
                    unsigned int i;
                    if(ucKeyNumberTemp==11) //等于小数点
                    {
                       if(ucDotBitMax==0) //如果限定的小数点最大数是0，就意味着此数据不允许带小数点，必
须是整数。
                          {
                              return; //直接返回退出
                          }
                          else if(*p_ucDotCnt>0)  //小数点个数大于0，意味着当前数组已经包含了小数点，此时再
输入小数点则无效。
                          {
                              return; //直接返回退出
                          }
                          else  //否则有效，记录当前已经包含一个小数点的信息。
                          {
```

```
                    *p_ucDotCnt=1;  //只能包含一个小数点
                }
            }
            else  //如果输入的不是小数点
            {
                if(*p_ucDotCnt==1) //如果之前已经输入了一个小数点，那么此时输入的数字就是小数点后的数据
                                    {
                    if(*p_ucDotBitS<ucDotBitMax) //如果小数点位数还没超过最大限制位数，则继续加1记录当前小数点位数。
                    {
                        *p_ucDotBitS=(*p_ucDotBitS)+1;
                    }
                    else //如果小数点位数已经超过允许的范围，则输入的按键无效，直接退出。
                    {
                       return; //直接返回退出
                    }
                                }
                                    else if(*p_ucIntCnt<ucIntCntMax)//如果之前没有输入小数点，那么输入的就是整数个数超，整数个数没有超过极限
                                    {
                                        *p_ucIntCnt=(*p_ucIntCnt)+1;
                                    }
                                    else //整数个数超过极限
                                    {
                      return; //直接返回退出
                                    }
            }


            if(*p_ucWdPartCnt==0&&p_ucSetDataBuffer[0]==0&&ucKeyNumberTemp!=11)  //如果当前默认位置是第0个位置，并且默认第0个数据是0，并且当前的按键输入不是小数点，则不用移位
            {
                ;
            }
            else  //否则，移位
            {
               for(i=0;i<(ucDataCntMax-1);i++)  //移位
               {
                  p_ucSetDataBuffer[ucDataCntMax-1-i]=p_ucSetDataBuffer[ucDataCntMax-2-i];
               }
               *p_ucWdPartCnt=(*p_ucWdPartCnt)+1;
            }
            p_ucSetDataBuffer[0]=ucKeyNumberTemp; //当前输入的数字或者小数点永远在第右边第0个位置。


}
/* 注释四:
* 本节的核心函数，值得好好研究!
```

```
* 功能: 把一个带小数点的BCD码数组转换成一个long类型的数值。
* 第1个参数ucConverDataSize是这个数组的最大有效个数。
* 第2个参数ucConverDotCnt是这个数组要转换成的long数值带几个小数点
* 第3个参数*p_ucConverBuffer是具体此数组的数据
* 函数最后返回被转换的long数值。
*/
unsigned long buffer_to_data(unsigned char ucConverDataSize,unsigned char ucConverDotCnt,unsigned char
*p_ucConverBuffer)
{
   unsigned long ulConverResult=0;
   unsigned long ulConverResultTemp=0;
   unsigned char ucConverResultBuffer[6]; //因为本节内容的ucConverDataSize是6，所以取6.
   unsigned char i;
   unsigned char j;
   unsigned char ucConverFlag;
   for(i=0;i<ucConverDataSize;i++)
   {
      ucConverResultBuffer[i]=0;  //先把临时缓冲区清零
   }
   j=0;
   ucConverFlag=0;
   for(i=0;i<ucConverDataSize;i++)
   {
       if(p_ucConverBuffer[i]==11) //小数点
       {
          ucConverFlag=i; //记录小数点的位置
       }
       else if(p_ucConverBuffer[i]<10)
       {
          ucConverResultBuffer[j]=p_ucConverBuffer[i];  //提取数组中的有效数字
          j++;
       }
   }
   for(i=0;i<ucConverDataSize;i++)    //通过处理每一位从而合成一个long类型的数值
   {
       ulConverResultTemp=0;
       ulConverResultTemp=ucConverResultBuffer[i];
       for(j=0;j<i;j++)
       {
           ulConverResultTemp=ulConverResultTemp*10;  //把每一位对应的进位扩大到对应的倍数
       }
       ulConverResult=ulConverResult+ulConverResultTemp;
   }
   for(i=ucConverFlag;i<ucConverDotCnt;i++) //根据数组小数点的位置和实际要转换成的小数点个数，来扩大到对
应的倍数。
   {
      ulConverResult=ulConverResult*10;
   }
   return ulConverResult;
}
```

```
void key_delete_input(void) //删除按键
{
    static unsigned int i;
   switch(ucWd)
   {
      case 1:   //第1窗口。本节程序只有1个窗口
           switch(ucPart)
           {
              case 1:  //1窗口第1项
                    //清零
                                      ulData_1=0; //long数值清零
                                       ucIntCnt_1=0;
                    ucDotBitS_1=0;
                    ucDotCnt_1=0;
                    ucWdPartCnt_1=0;
                    for(i=0;i<6;i++)
                    {
                       ucDataBuffer_1[i]=10;
                    }
                    ucDataBuffer_1[0]=0; //第0个位置填入0

                    ucWd1Part1Update=1;//第一行局部更新显示
                                   ucWd1Part2Update=1;//第二行局部更新显示
                    break;

           }

           break;

   }

}
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                    break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                    break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                    break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                    break;
            case 4:
```

```
            p_ucAnyNumber=Zf816_4;
                    break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                    break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                    break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                    break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                    break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                    break;
            case 10:  //空格
             p_ucAnyNumber=Zf816_nc;
                    break;
                       case 11:   //小数点
             p_ucAnyNumber=Zf816_dot;
                    break;
               default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                    break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    static unsigned char *p_ucAnyNumber; //经过数字转换成字模后，分解变量的某位字模首地址
    static unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    static unsigned int i;
        static unsigned char ucDataBuffer_temp[6]; //分解一个10进制的long类型数据的每一位
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:   //显示窗口1的数据
               if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
               {
                    ucWd1Update=0;  //及时清零，避免一直更新
                    ucWd1Part1Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    ucWd1Part2Update=1; //激活窗口1的第2行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
                    clear_all_canvas();  //把画布全部清零
                    display_lattice(0,0,Hz1616_yi,0,2,16,0);     //一项数组
                    display_lattice(1,0,Hz1616_xiang,0,2,16,0);
                    display_lattice(2,0,Hz1616_shu,0,2,16,0);
```

```
            display_lattice(3,0,Hz1616_zhu,0,2,16,0);
            display_lattice(4,0,Zf816_mao_hao,0,1,16,0); //冒号
            display_lattice(0,16,Hz1616_yi,0,2,16,0);     //一项数值
            display_lattice(1,16,Hz1616_xiang,0,2,16,0);
            display_lattice(2,16,Hz1616_shu,0,2,16,0);
            display_lattice(3,16,Hz1616_zhi,0,2,16,0);
            display_lattice(4,16,Zf816_mao_hao,0,1,16,0); //冒号
        }
        if(ucWd1Part1Update==1) //窗口1的第1行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
            ucWd1Part1Update=0; //及时清零，避免一直更新
            if(ucPart==1) //被选中
            {
               ucCursorFlag=1; //反显 显示
            }
            else //没被选中
            {
                ucCursorFlag=0; //正常 显示
            }

            for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
            {
                p_ucAnyNumber=number_to_matrix(ucDataBuffer_1[5-i]);
                insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地
址，一共可以显示6个字符，因此取值范围是0到5
            }
            display_lattice(5,0,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最后
的参数0是偏移量
        }
        if(ucWd1Part2Update==1) //窗口1的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
            ucWd1Part2Update=0; //及时清零，避免一直更新
            if(ucPart==2) //被选中
            {
               ucCursorFlag=1; //反显 显示
            }
            else //没被选中
            {
                ucCursorFlag=0; //正常 显示
            }
            if(ulData_1>=10000)
                                {
                                   ucDataBuffer_temp[5]=ulData_1%100000/10000;
                                }
                                else
                                {
                                   ucDataBuffer_temp[5]=10; //空格
                                }
            if(ulData_1>=1000)
                                {
```

```
                                        ucDataBuffer_temp[4]=ulData_1%10000/1000;
                        }
                                    else
                                    {
                                        ucDataBuffer_temp[4]=10; //空格
                        }
                                    ucDataBuffer_temp[3]=ulData_1%1000/100;
                                    ucDataBuffer_temp[2]=11;  //11代表小数点
                                    ucDataBuffer_temp[1]=ulData_1%100/10;
                                    ucDataBuffer_temp[0]=ulData_1%10/1;

                        for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                        {
                            p_ucAnyNumber=number_to_matrix(ucDataBuffer_temp[5-i]);
                            insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地
址，一共可以显示6个字符，因此取值范围是0到5
                        }
                        display_lattice(5,16,ucCanvasBuffer,ucCursorFlag,6,16,0);   //显示整屏的画布,最
后的参数0是偏移量
                }

                break;
            //本程序只有1个窗口，所以只有一个case 1，如果要增加窗口，就直接增加 case 2，case 3...
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                  ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);          //垂直地址
         WriteCommand(0x80);            //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
```

```
            LCDWriteData(ucFillDate);
        }
        y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释五:
* 注意，这节内容的画布跟第79节前面的画布大小不一样，第79节前面的横向是4个字节，这节的横向是6个字节。
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至5，因为画布的横向只要6个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
              ucTemp=ucArray[j*x_amount+i];
              if(ucFbFlag==0)
              {
                 ucCanvasBuffer[(y+j)*6+x+i]=ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向
是4个字节，要稍微注意的。
              }
              else
              {
                 ucCanvasBuffer[(y+j)*6+x+i]=~ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向
是4个字节，要稍微注意的。
              }
      }
   }
}
/* 注释六:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
```

```
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
```

```
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
```

总结陈词:

这节讲了把BCD码数组同步实时转换成数值的算法程序，相反，把数值转换成BCD码数组的逆运算程序应该怎么写？欲知详情，请听下回分解----实时同步把加减按键输入的数值转换成BCD码数组的液晶屏显示程序。

（未完待续，下节更精彩，不要走开哦）

第八十五节: 实时同步把加减按键输入的数值转换成BCD码数组的液晶屏显示程序。

开场白:

把运算处理完的数值转换成BCD码数组才可以更好方便显示和数字按键的输入编辑。这一节主要跟大家讲这方面的算法程序。本节的核心转换函数是void data_to_buffer(...)。

具体内容，请看源代码讲解。

（1）　硬件平台:
基于朱兆祺51单片机学习板。数字1键对应S1键，数字2键对应S2键，数字3键对应S3键…. 数字9键对应S9键，数字0键对应S10键。小数键对应S11，S13按键是加按键，S14按键是减按键，清零键对应S16，其它按键不用。
（2）　实现功能:
通过S13,S14这两个加减按键更改第2行显示的数值，此数值会同步更新显示在第1行的BCD码数组上。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  10    //按键去抖动延时的时间
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void key_number_input(unsigned char ucKeyNumber); //输入数字按键
void set_data(unsigned char ucKeyNumberTemp, //设置参数
              unsigned char ucDotBitMax,
              unsigned char ucDataCntMax,
              unsigned char *p_ucDotCnt,
              unsigned char *p_ucDotBitS,
                          unsigned char *p_ucWdPartCnt,
                          unsigned char *p_ucSetDataBuffer,
                          unsigned char ucIntCntMax,
                          unsigned char *p_ucIntCnt);
void data_to_buffer(unsigned long ulWillConverData,  //把数值转换成数组
                    unsigned char ucConverDotCnt,
                                  unsigned char ucConverDataSize,
                                  unsigned char *p_ucDotCnt,
```

```
                                        unsigned char *p_ucDotBitS,
                                        unsigned char *p_ucWdPartCnt,
                                        unsigned char *p_ucConverBuffer);
unsigned long buffer_to_data(unsigned char ucConverDataSize,unsigned char ucConverDotCnt,unsigned char
*p_ucConverBuffer); //把带小数点的BCD数组转换成long类型的数值。
void key_delete_input(void); //删除按键
void T0_time(); //定时中断函数
void key_service();
void key_scan(); //按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
```

```
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Zf816_dot[]=  //小数点
{
/*--  文字:  .  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x60,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_yi[]=
{
/*--  文字:  一  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_xiang[]=
{
/*--  文字:  项  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
```

```
0x00,0x00,0x03,0xFE,0xFC,0x20,0x10,0x40,0x11,0xFC,0x11,0x04,0x11,0x24,0x11,0x24,
0x11,0x24,0x11,0x24,0x1D,0x24,0xE1,0x34,0x00,0x48,0x01,0x86,0x06,0x02,0x00,0x00,
};
code unsigned char Hz1616_shu[]=
{
/*--  文字:  数  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x20,0x49,0x30,0x2A,0x20,0x1C,0x20,0xFF,0x7E,0x1C,0x44,0x2B,0x44,0x48,0xC4,
0x08,0x28,0xFF,0x28,0x12,0x10,0x34,0x10,0x0C,0x28,0x32,0x4E,0xC0,0x84,0x00,0x00,
};
code unsigned char Hz1616_zhu[]=
{
/*--  文字:  组  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x00,0x19,0xF8,0x11,0x08,0x25,0x08,0x25,0x08,0x79,0xF8,0x09,0x08,0x11,0x08,
0x21,0x08,0x7D,0xF8,0x01,0x08,0x01,0x08,0x0D,0x08,0x73,0xFE,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_zhi[]=
{
/*--  文字:  值  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x40,0x18,0x60,0x17,0xFC,0x10,0x40,0x20,0x80,0x33,0xF8,0x62,0x08,0xA3,0xF8,
0x22,0x08,0x23,0xF8,0x22,0x08,0x23,0xF8,0x22,0x08,0x22,0x08,0x2F,0xFE,0x20,0x00,
};
/* 注释一:
* 以下是画布显示数组。横向是6个字节，纵向16行，可以显示3个16x16的汉字.
*  注意，这节内容的画布跟前面79章节的画布大小不一样，79节前面的横向是4个字节，这节的横向是6个字节。
*/
unsigned char ucCanvasBuffer[]=
{
0x00,0x00,0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
};
/* 注释二:
* 以下5个变量记录一个参数的5种信息，包括小数点的数量，小数点个数，数据的位置，数组具体值，整数个数
```

```
*/
unsigned char ucDotCnt_1=0;  //记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效
unsigned char ucDotBitS_1=0; //记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任何输入按键则无效
unsigned char ucWdPartCnt_1=0; //记录当前输入的数据在数组中的位置。
unsigned char ucDataBuffer_1[6]={0,10,10,10,10,10}; //一项的BCD码数组缓冲
unsigned char ucIntCnt_1=0; //记录当前输入的整数个数，如果整数的个数如果超过规定ucIntCntMax位，此时再按任何输入按键则无效
unsigned long ulData_1=0; //用一个long变量表示BCD码的具体数值。
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucPart=2; //局部变量 0代表没有选中任何一行，其它数值1到4代表选中某一行
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量      1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
unsigned char ucWd1Part2Update=0; //窗口1的第2行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                  key_service(); //按键服务程序
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
void T0_time() interrupt 1
{
```

```
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan();//按键扫描函数 放在定时中断里
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
         beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
           beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
     case 1:   //按键扫描输出第ucRowRecord列低电平
              if(ucRowRecord==1)  //第一列输出低电平
                   {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==2)  //第二列输出低电平
                   {
              key_dr1=1;
              key_dr2=0;
              key_dr3=1;
              key_dr4=1;
                   }
               else if(ucRowRecord==3)  //第三列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=0;
              key_dr4=1;
                   }
               else   //第四列输出低电平
                   {
              key_dr1=1;
              key_dr2=1;
              key_dr3=1;
              key_dr4=0;
                   }
```

```
        uiKeyTimeCnt=0;  //延时计数器清零
        ucKeyStep++;     //切换到下一个运行步骤
            break;
    case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
        uiKeyTimeCnt++;
                if(uiKeyTimeCnt>1)
                {
                   uiKeyTimeCnt=0;
           ucKeyStep++;     //切换到下一个运行步骤
                }
            break;
    case 3:
        if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
        {
           ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
           ucKeyLock=0;  //按键自锁标志清零
           uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                       ucRowRecord++;  //输出下一列
                       if(ucRowRecord>4)
                       {
                          ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                       }
        }
                else if(ucKeyLock==0)  //有按键按下，且是第一次触发
                {
                   if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                       {
                          uiKeyTimeCnt++;  //去抖动延时计数器
                              if(uiKeyTimeCnt>const_key_time)
                              {
                                 uiKeyTimeCnt=0;
                                 ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                     if(ucRowRecord==1)  //第一列输出低电平
                         {
                                    ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                           }
                     else if(ucRowRecord==2)  //第二列输出低电平
                         {
                                    ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                           }
                     else if(ucRowRecord==3)  //第三列输出低电平
                         {
                                    ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                           }
                     else   //第四列输出低电平
                         {
                                    ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                           }
```

```
            }

        }
    else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
        {
           uiKeyTimeCnt++;  //去抖动延时计数器
               if(uiKeyTimeCnt>const_key_time)
               {
                  uiKeyTimeCnt=0;
                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
      if(ucRowRecord==1)  //第一列输出低电平
          {
                     ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
          }
      else if(ucRowRecord==2)  //第二列输出低电平
          {
                     ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
          }
      else if(ucRowRecord==3)  //第三列输出低电平
          {
                     ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
          }
      else   //第四列输出低电平
          {
                     ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
          }
               }

        }
    else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
        {
           uiKeyTimeCnt++;  //去抖动延时计数器
               if(uiKeyTimeCnt>const_key_time)
               {
                  uiKeyTimeCnt=0;
                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
      if(ucRowRecord==1)  //第一列输出低电平
          {
                     ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
          }
      else if(ucRowRecord==2)  //第二列输出低电平
          {
                     ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S10键
          }
      else if(ucRowRecord==3)  //第三列输出低电平
          {
                     ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
          }
```

```
                        else   //第四列输出低电平
                            {
                                    ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
                            }
                                }

                      }
                    else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                        if(ucRowRecord==1)  //第一列输出低电平
                            {
                                    ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                            }
                        else if(ucRowRecord==2)  //第二列输出低电平
                            {
                                    ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                            }
                        else if(ucRowRecord==3)  //第三列输出低电平
                            {
                                    ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                            }
                        else   //第四列输出低电平
                            {
                                    ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                            }
                                }

                      }

                }
            break;
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 数字1 对应朱兆祺学习板的S1键
          key_number_input(1); //输入数字按键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 数字2 对应朱兆祺学习板的S2键
          key_number_input(2); //输入数字按键
```

```
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 3:// 数字3 对应朱兆祺学习板的S3键
        key_number_input(3); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 数字4 对应朱兆祺学习板的S4键
        key_number_input(4); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 5:// 数字5 对应朱兆祺学习板的S5键
        key_number_input(5); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 6:// 数字6 对应朱兆祺学习板的S6键
        key_number_input(6); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 7:// 数字7 对应朱兆祺学习板的S7键
        key_number_input(7); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 8: //数字8 对应朱兆祺学习板的S8键
        key_number_input(8); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 9:// 数字9 对应朱兆祺学习板的S9键
        key_number_input(9); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 10:// 数字0  对应朱兆祺学习板的S10键
        key_number_input(0); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 11:// 小数点按键 对应朱兆祺学习板的S11键
        key_number_input(11); //输入数字按键  11代表小数点
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 12:// 本节暂时不用 对应朱兆祺学习板的S12键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
```

```
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 13:// 加按键 对应朱兆祺学习板的S13键
            ulData_1++;
            if(ulData_1>99999)
            {
                ulData_1=99999;
            }
            data_to_buffer(ulData_1,  //把数值转换成数组，这是本节核心函数，请好好研究此函数的具体功能。
                                      2,  //小数点最大个数
                                          6,  //数组缓冲最大个数
                                          &ucDotCnt_1,
                                          &ucDotBitS_1,
                                          &ucWdPartCnt_1,
                                          ucDataBuffer_1);  //被转换成的数组
            ucWd1Part1Update=1;   //实时更新显示数组
            ucWd1Part2Update=1;   //实时更新显示数值
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 14:// 减按键  对应朱兆祺学习板的S14键
            ulData_1--;
            if(ulData_1>99999) //unsigned long类型的变量0减去1会变成0xffffffff
            {
                ulData_1=0;
            }
            data_to_buffer(ulData_1,  //把数值转换成数组，这是本节核心函数，请好好研究此函数的具体功能。
                                      2,  //小数点最大个数
                                          6,  //数组缓冲最大个数
                                          &ucDotCnt_1,
                                          &ucDotBitS_1,
                                          &ucWdPartCnt_1,
                                          ucDataBuffer_1);  //被转换成的数组
            ucWd1Part1Update=1;   //实时更新显示数组
            ucWd1Part2Update=1;   //实时更新显示数值
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 15:// 本节暂时不用 对应朱兆祺学习板的S15键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 16:// 清除按键 对应朱兆祺学习板的S16键
            key_delete_input(); //删除按键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    }
}
void key_number_input(unsigned char ucKeyNumber) //输入数字按键
```

```
{
    switch(ucWd)
    {
       case 1:    //第1窗口。本节程序只有1个窗口
             switch(ucPart)
             {
                 case 1:  //1窗口第1项
                      set_data(ucKeyNumber,
                                                  2,  //小数点最大个数
                                                      6,  //数组缓冲最大个数
                                                  &ucDotCnt_1,
                                                      &ucDotBitS_1,
                                                      &ucWdPartCnt_1,
                                                      ucDataBuffer_1,
                                                      3, //整数部分的最大个数
                                                      &ucIntCnt_1);
                                              ulData_1=buffer_to_data(6,2,ucDataBuffer_1); //把带小数点的
BCD码数组转换成long数值。
                      ucWd1Part1Update=1;//第一行局部更新显示
                                              ucWd1Part2Update=1;//第二行局部更新显示
                      break;
             }

             break;
    }

}
/* 注释三:
* 涉及到参数的4种信息，包括小数点的数量，小数点的个数，数据的位置，数组具体值，整数的数量，整数的个数
，以及它们之间的相互作用关系。
* 以下参数，指针类型的参数是让代入的全局变量在退出函数后维持它当前最新更改的数值不变。
* 第1个参数ucKeyNumberTemp是当前按键输入的数值。
* 第2个参数ucDotBitMax是限定被设置参数的小数点最大位数。
* 第3个参数ucDataCntMax是限定被设置参数的最大数组个数。
* 第4个参数*p_ucDotCnt是记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时
再按小数点按键则无效。
* 第5个参数*p_ucDotBitS是记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任
何输入按键则无效
* 第6个参数*p_ucWdPartCnt是记录当前输入的数据在数组中的位置，方便锁定每次按键输入的数字显示位置。
* 第7个参数*p_ucSetDataBuffer是BCD码数组缓冲的具体数字内容。
* 第8个参数ucIntCntMax是限定被设置参数的整数部分的最大位数。
* 第9个参数*p_ucIntCnt是记录当前输入的整数部分个数，如果整数部分的个数如果超过规定ucIntCntMax位，此时再
按任何输入按键则无效
*/
void set_data(unsigned char ucKeyNumberTemp,
              unsigned char ucDotBitMax,
              unsigned char ucDataCntMax,
              unsigned char *p_ucDotCnt,
              unsigned char *p_ucDotBitS,
                        unsigned char *p_ucWdPartCnt,
```

```
                        unsigned char *p_ucSetDataBuffer,
                        unsigned char ucIntCntMax,
                        unsigned char *p_ucIntCnt)
{
                    unsigned int i;
                    if(ucKeyNumberTemp==11) //等于小数点
                    {
                       if(ucDotBitMax==0) //如果限定的小数点最大数是0，就意味着此数据不允许带小数点，必
须是整数。
                       {
                           return; //直接返回退出
                       }
                       else if(*p_ucDotCnt>0)  //小数点个数大于0，意味着当前数组已经包含了小数点，此时再
输入小数点则无效。
                       {
                           return; //直接返回退出
                       }
                       else  //否则有效，记录当前已经包含一个小数点的信息。
                       {
                           *p_ucDotCnt=1;  //只能包含一个小数点
                       }
                    }
                    else  //如果输入的不是小数点
                    {
                        if(*p_ucDotCnt==1) //如果之前已经输入了一个小数点，那么此时输入的数字就是小数点
后的数据
                                           {
                           if(*p_ucDotBitS<ucDotBitMax) //如果小数点位数还没超过最大限制位数，则继续加
1记录当前小数点位数。
                           {
                               *p_ucDotBitS=(*p_ucDotBitS)+1;
                           }
                           else //如果小数点位数已经超过允许的范围，则输入的按键无效，直接退出。
                           {
                              return; //直接返回退出
                           }
                                       }
                                           else if(*p_ucIntCnt<ucIntCntMax)//如果之前没有输入小数点
，那么输入的就是整数个数超，整数个数没有超过极限
                                           {
                                               *p_ucIntCnt=(*p_ucIntCnt)+1;
                                           }
                                           else //整数个数超过极限
                                           {
                             return; //直接返回退出
                                           }
                    }


                    if(*p_ucWdPartCnt==0&&p_ucSetDataBuffer[0]==0&&ucKeyNumberTemp!=11)  //如果当前默认
```

位置是第0个位置，并且默认第0个数据是0，并且当前的按键输入不是小数点，则不用移位

```
                    {
                        ;
                    }
                    else  //否则，移位
                    {
                       for(i=0;i<(ucDataCntMax-1);i++)  //移位
                       {
                          p_ucSetDataBuffer[ucDataCntMax-1-i]=p_ucSetDataBuffer[ucDataCntMax-2-i];
                       }
                       *p_ucWdPartCnt=(*p_ucWdPartCnt)+1;
                    }
                    p_ucSetDataBuffer[0]=ucKeyNumberTemp; //当前输入的数字或者小数点永远在第右边第0个位
```

置。

```
}
/* 注释四:
* 功能: 把一个带小数点的BCD码数组转换成一个long类型的数值。
* 第1个参数ucConverDataSize是这个数组的最大有效个数。
* 第2个参数ucConverDotCnt是这个数组要转换成的long数值带几个小数点
* 第3个参数*p_ucConverBuffer是具体此数组的数据
* 函数最后返回被转换的long数值。
*/
unsigned long buffer_to_data(unsigned char ucConverDataSize,unsigned char ucConverDotCnt,unsigned char
*p_ucConverBuffer)
{
   unsigned long ulConverResult=0;
   unsigned long ulConverResultTemp=0;
   unsigned char ucConverResultBuffer[6]; //因为本节内容的ucConverDataSize是6，所以取6.
   unsigned char i;
   unsigned char j;
   unsigned char ucConverFlag;
   for(i=0;i<ucConverDataSize;i++)
   {
      ucConverResultBuffer[i]=0;  //先把临时缓冲区清零
   }
   j=0;
   ucConverFlag=0;
   for(i=0;i<ucConverDataSize;i++)
   {
       if(p_ucConverBuffer[i]==11) //小数点
       {
          ucConverFlag=i; //记录小数点的位置
       }
       else if(p_ucConverBuffer[i]<10)
       {
          ucConverResultBuffer[j]=p_ucConverBuffer[i];  //提取数组中的有效数字
          j++;
       }
```

```
    }
    for(i=0;i<ucConverDataSize;i++)    //通过处理每一位从而合成一个long类型的数值
    {
        ulConverResultTemp=0;
        ulConverResultTemp=ucConverResultBuffer[i];
        for(j=0;j<i;j++)
        {
            ulConverResultTemp=ulConverResultTemp*10;  //把每一位对应的进位扩大到对应的倍数
        }
        ulConverResult=ulConverResult+ulConverResultTemp;
    }
    for(i=ucConverFlag;i<ucConverDotCnt;i++) //根据数组小数点的位置和实际要转换成的小数点个数，来扩大到对应的倍数。
    {
       ulConverResult=ulConverResult*10;
    }
    return ulConverResult;
}
/* 注释五:
* 本节的核心函数，值得好好研究!
* 功能: 把一个long类型的数值转换成一个带小数点的BCD码数组
* 第1个参数ulWillConverData是即将被转换的unsigned long类型数值。
* 第2个参数ucConverDotCnt是这个数值带几个小数点
* 第3个参数ucConverDataSize是这个数组的最大有效个数。
* 第4个参数*p_ucDotCnt是记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效。
* 第5个参数*p_ucDotBitS是记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任何输入按键则无效
* 第6个参数*p_ucWdPartCnt是记录当前输入的数据在数组中的位置，方便锁定每次按键输入的数字显示位置。
* 第7个参数*p_ucConverBuffer是具体此数组的数据。
*/
void data_to_buffer(unsigned long ulWillConverData,
                    unsigned char ucConverDotCnt,
                                  unsigned char ucConverDataSize,
                                  unsigned char *p_ucDotCnt,
                                  unsigned char *p_ucDotBitS,
                                  unsigned char *p_ucWdPartCnt,
                                  unsigned char *p_ucConverBuffer)
{
    unsigned char ucConverResultBuffer[6]; //因为本节内容的ucConverDataSize是6，所以取6.
    unsigned char i;
    unsigned char ucValidaDotCnt=0;
    if(ucConverDotCnt==0)  //没有小数点
    {
        *p_ucDotCnt=0;    //当前没有输入小数点的标志
        *p_ucDotBitS=0;   //当前输入的小数点个数是0
        ucConverResultBuffer[5]=10;  //没有小数点的时候，第5位必然是显示空格
            //以下是具体把数值转换成数组，不需要显示的高位填入10表示显示空格
        if(ulWillConverData>=10000)
        {
```

```
        ucConverResultBuffer[4]=u1WillConverData%100000/10000;
    }
    else
    {
        ucConverResultBuffer[4]=10;
    }
    if(u1WillConverData>=1000)
    {
        ucConverResultBuffer[3]=u1WillConverData%10000/1000;
    }
    else
    {
        ucConverResultBuffer[3]=10;
    }
    if(u1WillConverData>=100)
    {
        ucConverResultBuffer[2]=u1WillConverData%1000/100;
    }
    else
    {
        ucConverResultBuffer[2]=10;
    }
    if(u1WillConverData>=10)
    {
        ucConverResultBuffer[1]=u1WillConverData%100/10;
    }
    else
    {
        ucConverResultBuffer[1]=10;
    }
    ucConverResultBuffer[0]=u1WillConverData%10;
}
else if(ucConverDotCnt==1)  //1位小数点
{
    *p_ucDotCnt=1;  //当前已经有输入小数点的标志
    *p_ucDotBitS=1; //当前输入的小数点个数是1
    ucConverResultBuffer[1]=11;  //第1位填入小数点11
        //以下是具体把数值转换成数组，不需要显示的高位填入10表示显示空格
    if(u1WillConverData>=10000)
    {
        ucConverResultBuffer[5]=u1WillConverData%100000/10000;
    }
    else
    {
        ucConverResultBuffer[5]=10;
    }
    if(u1WillConverData>=1000)
    {
        ucConverResultBuffer[4]=u1WillConverData%10000/1000;
    }
```

```
    else
    {
       ucConverResultBuffer[4]=10;
    }
    if(ulWillConverData>=100)
    {
       ucConverResultBuffer[3]=ulWillConverData%1000/100;
    }
    else
    {
       ucConverResultBuffer[3]=10;
    }
    ucConverResultBuffer[2]=ulWillConverData%100/10;
    ucConverResultBuffer[0]=ulWillConverData%10;
}
else if(ucConverDotCnt==2)  //2位小数点
{
    *p_ucDotCnt=1;  //当前已经有输入小数点的标志
    *p_ucDotBitS=2; //当前输入的小数点个数是2
    ucConverResultBuffer[2]=11; //第2位填入小数点11
        //以下是具体把数值转换成数组，不需要显示的高位填入10表示显示空格
    if(ulWillConverData>=10000)
    {
       ucConverResultBuffer[5]=ulWillConverData%100000/10000;
    }
    else
    {
       ucConverResultBuffer[5]=10;
    }
    if(ulWillConverData>=1000)
    {
       ucConverResultBuffer[4]=ulWillConverData%10000/1000;
    }
    else
    {
       ucConverResultBuffer[4]=10;
    }
    ucConverResultBuffer[3]=ulWillConverData%1000/100;
    ucConverResultBuffer[1]=ulWillConverData%100/10;
    ucConverResultBuffer[0]=ulWillConverData%10;
}
ucValidaDotCnt=0;
for(i=0;i<ucConverDataSize;i++)
{
    if(ucConverResultBuffer[i]!=10)  //统计数组有效的BCD码位数
    {
       ucValidaDotCnt++;
    }
    p_ucConverBuffer[i]=ucConverResultBuffer[i];  //把转换的结果传输给实际的数组用来外部显示
```

```
   }
   *p_ucWdPartCnt=ucValidaDotCnt-1;  //当前显示的实际位置
}
void key_delete_input(void) //删除按键
{
    static unsigned int i;
   switch(ucWd)
   {
      case 1:   //第1窗口。本节程序只有1个窗口
           switch(ucPart)
           {
              case 1:  //1窗口第1行
                    //清零
                                    ulData_1=0; //long数值清零
                                     ucIntCnt_1=0;
                    ucDotBitS_1=0;
                    ucDotCnt_1=0;
                    ucWdPartCnt_1=0;
                    for(i=0;i<6;i++)
                    {
                       ucDataBuffer_1[i]=10;
                    }
                    ucDataBuffer_1[0]=0; //第0个位置填入0

                    ucWd1Part1Update=1;//第一行局部更新显示
                                   ucWd1Part2Update=1;//第二行局部更新显示
                    break;
              case 2:  //1窗口第2行
                    //清零
                                    ulData_1=0; //long数值清零
                                     ucIntCnt_1=0;
                    ucDotBitS_1=0;
                    ucDotCnt_1=0;
                    ucWdPartCnt_1=0;
                    for(i=0;i<6;i++)
                    {
                       ucDataBuffer_1[i]=10;
                    }
                    ucDataBuffer_1[0]=0; //第0个位置填入0

                    ucWd1Part1Update=1;//第一行局部更新显示
                                   ucWd1Part2Update=1;//第二行局部更新显示
                    break;
           }

           break;

   }

}
```

```
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                     break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                     break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                     break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                     break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                     break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                     break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                     break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                     break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                     break;
            case 9:
             p_ucAnyNumber=Zf816_9;
                     break;
            case 10:  //空格
             p_ucAnyNumber=Zf816_nc;
                     break;
                        case 11:   //小数点
             p_ucAnyNumber=Zf816_dot;
                     break;
                   default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                     break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    static unsigned char *p_ucAnyNumber; //经过数字转换成字模后，分解变量的某位字模首地址
```

```
    static unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    static unsigned int i;
        static unsigned char ucDataBuffer_temp[6]; //分解一个10进制的long类型数据的每一位
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:   //显示窗口1的数据
                if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
                {
                      ucWd1Update=0;  //及时清零，避免一直更新
                      ucWd1Part1Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上有所改进
                      ucWd1Part2Update=1; //激活窗口1的第2行局部更新显示变量，这里在前面数码管显示框架上有所改进
                      display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
                      clear_all_canvas();  //把画布全部清零
                      display_lattice(0,0,Hz1616_yi,0,2,16,0);     //一项数组
                      display_lattice(1,0,Hz1616_xiang,0,2,16,0);
                      display_lattice(2,0,Hz1616_shu,0,2,16,0);
                      display_lattice(3,0,Hz1616_zhu,0,2,16,0);
                      display_lattice(4,0,Zf816_mao_hao,0,1,16,0); //冒号
                      display_lattice(0,16,Hz1616_yi,0,2,16,0);     //一项数值
                      display_lattice(1,16,Hz1616_xiang,0,2,16,0);
                      display_lattice(2,16,Hz1616_shu,0,2,16,0);
                      display_lattice(3,16,Hz1616_zhi,0,2,16,0);
                      display_lattice(4,16,Zf816_mao_hao,0,1,16,0); //冒号
                }
                if(ucWd1Part1Update==1) //窗口1的第1行局部更新显示变量,里面放一些经常需要刷新显示的内容
                {
                        ucWd1Part1Update=0; //及时清零，避免一直更新
                        if(ucPart==1) //被选中
                        {
                           ucCursorFlag=1; //反显 显示
                        }
                        else //没被选中
                        {
                            ucCursorFlag=0; //正常 显示
                        }

                        for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                        {
                            p_ucAnyNumber=number_to_matrix(ucDataBuffer_1[5-i]);
                            insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地址，一共可以显示6个字符，因此取值范围是0到5
                        }
                        display_lattice(5,0,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最后的参数0是偏移量
                }
                if(ucWd1Part2Update==1) //窗口1的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
                {
                        ucWd1Part2Update=0; //及时清零，避免一直更新
```

```
                        if(ucPart==2) //被选中
                        {
                           ucCursorFlag=1; //反显 显示
                        }
                        else //没被选中
                        {
                            ucCursorFlag=0; //正常 显示
                        }
                        if(ulData_1>=10000)
                                                {
                                                   ucDataBuffer_temp[5]=ulData_1%100000/10000;
                                                }
                                                else
                                                {
                                                   ucDataBuffer_temp[5]=10; //空格
                                                }
                        if(ulData_1>=1000)
                                                {
                                                      ucDataBuffer_temp[4]=ulData_1%10000/1000;
                        }
                                                else
                                                {
                                                      ucDataBuffer_temp[4]=10; //空格
                        }
                                                ucDataBuffer_temp[3]=ulData_1%1000/100;
                                                ucDataBuffer_temp[2]=11;  //11代表小数点
                                                ucDataBuffer_temp[1]=ulData_1%100/10;
                                                ucDataBuffer_temp[0]=ulData_1%10/1;

                        for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                        {
                            p_ucAnyNumber=number_to_matrix(ucDataBuffer_temp[5-i]);
                            insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地
址，一共可以显示6个字符，因此取值范围是0到5
                        }
                        display_lattice(5,16,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最
后的参数0是偏移量
                }

                break;
        //本程序只有1个窗口，所以只有一个case 1，如果要增加窗口，就直接增加 case 2，case 3...
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
```

```
      {
                  ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释六:
* 注意，这节内容的画布跟第79节前面的画布大小不一样，第79节前面的横向是4个字节，这节的横向是6个字节。
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至5，因为画布的横向只要6个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
              ucTemp=ucArray[j*x_amount+i];
              if(ucFbFlag==0)
              {
                 ucCanvasBuffer[(y+j)*6+x+i]=ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向是4个字节，要稍微注意的。
              }
```

```
            else
            {
               ucCanvasBuffer[(y+j)*6+x+i]=~ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向是4个字节，要稍微注意的。
            }
      }
   }
}
/* 注释七:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //          delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
```

```
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块
的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
```

```
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
```

总结陈词:
前面两节都讲了数组和数值的相互转换函数，结合前面的基础，下一节讲数字键盘与液晶菜单的综合程序，欲知详情，请听下回分解----数字键盘与液晶菜单的综合程序。
（未完待续，下节更精彩，不要走开哦）

第八十六节: 数字键盘与液晶菜单的综合程序。

开场白:
前面已经介绍完数值跟BCD码数组相互转换的算法，但是按键只能设置一项数据。如果多增加一项数据，变成两项数据，按键与显示菜单之间该如何关联起来，这样的程序框架是什么样的？其实很简单很有规律的，只需要在前面的基础上，在按键和显示函数里，根据不同的uPart行变量添加进不同的代码，即可完成。这就是鸿哥写的程序代码，不管添加多少代码，都是有一个“道”可循，非常有规律性。

具体内容，请看源代码讲解。
（1）　硬件平台:
基于朱兆祺51单片机学习板。数字1键对应S1键，数字2键对应S2键，数字3键对应S3键…. 数字9键对应S9键，数字0键对应S10键。小数键对应S11，S12按键是光标移动按键，S13按键是加按键，S14按键是减按键，清零键对应S16，其它按键不用。
（2）　实现功能:
通过S12光标移动按键，可以把负显光标切换到不同的行里面，根据光标所在位置，
通过S13,S14这两个加减按键可以更改对应的数。第1行和第2行的数据会彼此有关联，只要改其中一个，另外一个就会同步被更新。同理，第3行和第4行的数据也会彼此有关联，只要改其中一个，另外一个也会同步被更新。
（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time  10    //按键去抖动延时的时间
sbit key_sr1=P0^0; //第一行输入
sbit key_sr2=P0^1; //第二行输入
sbit key_sr3=P0^2; //第三行输入
sbit key_sr4=P0^3; //第四行输入
sbit key_dr1=P0^4; //第一列输出
sbit key_dr2=P0^5; //第二列输出
sbit key_dr3=P0^6; //第三列输出
sbit key_dr4=P0^7; //第四列输出
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit  LCDCS_dr  = P1^6;  //片选线
sbit  LCDSID_dr = P1^7;  //串行数据线
sbit  LCDCLK_dr = P3^2;  //串行时钟线
sbit  LCDRST_dr = P3^4;  //复位线
void SendByteToLcd(unsigned char ucData);  //发送一个字节数据到液晶模块
void SPIWrite(unsigned char ucWData, unsigned char ucWRS); //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
void WriteCommand(unsigned char ucCommand); //发送一个字节的命令给液晶模块
```

```
void LCDWriteData(unsigned char ucData);   //发送一个字节的数据给液晶模块
void LCDInit(void);  //初始化  函数内部包括液晶模块的复位
void display_clear(unsigned char ucFillDate); // 清屏 全部显示空填充0x00   全部显示点阵用0xff
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount);//把字模插入画布.
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr); //显示任意点阵函数
unsigned char *number_to_matrix(unsigned char  ucBitNumber); //把一位数字转换成字模首地址的函数
void delay_short(unsigned int uiDelayshort); //延时
void delay_long(unsigned int uiDelayLong);
void key_number_input(unsigned char ucKeyNumber); //输入数字按键
void set_data(unsigned char ucKeyNumberTemp, //设置参数
              unsigned char ucDotBitMax,
              unsigned char ucDataCntMax,
              unsigned char *p_ucDotCnt,
              unsigned char *p_ucDotBitS,
                          unsigned char *p_ucWdPartCnt,
                          unsigned char *p_ucSetDataBuffer,
                          unsigned char ucIntCntMax,
                          unsigned char *p_ucIntCnt);
void data_to_buffer(unsigned long ulWillConverData,  //把数值转换成数组
                    unsigned char ucConverDotCnt,
                                        unsigned char ucConverDataSize,
                                        unsigned char *p_ucDotCnt,
                                        unsigned char *p_ucDotBitS,
                                        unsigned char *p_ucWdPartCnt,
                                        unsigned char *p_ucConverBuffer);
unsigned long buffer_to_data(unsigned char ucConverDataSize,unsigned char ucConverDotCnt,unsigned char
*p_ucConverBuffer); //把带小数点的BCD数组转换成long类型的数值。
void key_delete_input(void); //删除按键
void T0_time(); //定时中断函数
void key_service();
void key_scan(); //按键扫描函数 放在定时中断里
void initial_myself();
void initial_peripheral();
void lcd_display_service(void); //应用层面的液晶屏显示程序
void clear_all_canvas(void);  //把画布全部清零
code unsigned char Zf816_0[]=
{
/*--  文字:  0  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x42,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_1[]=
{
/*--  文字:  1  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x10,0x70,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x7C,0x00,0x00,
};
code unsigned char Zf816_2[]=
```

```
{
/*--  文字:  2  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x04,0x04,0x08,0x10,0x20,0x42,0x7E,0x00,0x00,
};
code unsigned char Zf816_3[]=
{
/*--  文字:  3  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x04,0x18,0x04,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_4[]=
{
/*--  文字:  4  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x04,0x0C,0x14,0x24,0x24,0x44,0x44,0x7E,0x04,0x04,0x1E,0x00,0x00,
};
code unsigned char Zf816_5[]=
{
/*--  文字:  5  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x40,0x40,0x40,0x58,0x64,0x02,0x02,0x42,0x44,0x38,0x00,0x00,
};
code unsigned char Zf816_6[]=
{
/*--  文字:  6  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x1C,0x24,0x40,0x40,0x58,0x64,0x42,0x42,0x42,0x24,0x18,0x00,0x00,
};
code unsigned char Zf816_7[]=
{
/*--  文字:  7  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x7E,0x44,0x44,0x08,0x08,0x10,0x10,0x10,0x10,0x10,0x10,0x00,0x00,
};
code unsigned char Zf816_8[]=
{
/*--  文字:  8  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x3C,0x42,0x42,0x42,0x24,0x18,0x24,0x42,0x42,0x42,0x3C,0x00,0x00,
};
code unsigned char Zf816_9[]=
{
/*--  文字:  9  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x18,0x24,0x42,0x42,0x42,0x26,0x1A,0x02,0x02,0x24,0x38,0x00,0x00,
};
code unsigned char Zf816_nc[]=  //空字模
{
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
};
code unsigned char Zf816_dot[]=  //小数点
{
/*--  文字:  .  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x60,0x60,0x00,0x00,
};
code unsigned char Zf816_mao_hao[]=  //冒号
{
/*--  文字:  :  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=8x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
};
code unsigned char Hz1616_yi[]=
{
/*--  文字:  一  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_er[]=
{
/*--  文字:  二  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x00,0x10,0x3F,0xF8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x7F,0xFE,0x00,0x00,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_xiang[]=
{
/*--  文字:  项  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x00,0x00,0x03,0xFE,0xFC,0x20,0x10,0x40,0x11,0xFC,0x11,0x04,0x11,0x24,0x11,0x24,
0x11,0x24,0x11,0x24,0x1D,0x24,0xE1,0x34,0x00,0x48,0x01,0x86,0x06,0x02,0x00,0x00,
};
code unsigned char Hz1616_shu[]=
{
/*--  文字:  数  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x08,0x20,0x49,0x30,0x2A,0x20,0x1C,0x20,0xFF,0x7E,0x1C,0x44,0x2B,0x44,0x48,0xC4,
0x08,0x28,0xFF,0x28,0x12,0x10,0x34,0x10,0x0C,0x28,0x32,0x4E,0xC0,0x84,0x00,0x00,
};
code unsigned char Hz1616_zhu[]=
{
/*--  文字:  组  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x00,0x19,0xF8,0x11,0x08,0x25,0x08,0x25,0x08,0x79,0xF8,0x09,0x08,0x11,0x08,
0x21,0x08,0x7D,0xF8,0x01,0x08,0x01,0x08,0x0D,0x08,0x73,0xFE,0x00,0x00,0x00,0x00,
};
code unsigned char Hz1616_zhi[]=
{
```

```
/*--  文字:  值  --*/
/*--  宋体12;  此字体下对应的点阵为: 宽x高=16x16   --*/
0x10,0x40,0x18,0x60,0x17,0xFC,0x10,0x40,0x20,0x80,0x33,0xF8,0x62,0x08,0xA3,0xF8,
0x22,0x08,0x23,0xF8,0x22,0x08,0x23,0xF8,0x22,0x08,0x22,0x08,0x2F,0xFE,0x20,0x00,
};
/* 注释一:
* 以下是画布显示数组。横向是6个字节，纵向16行，可以显示3个16x16的汉字.
*  注意，这节内容的画布跟前面79章节的画布大小不一样，79节前面的横向是4个字节，这节的横向是6个字节。
*/
unsigned char ucCanvasBuffer[]=
{
0x00,0x00,0x00,0x00,0x00,0x00,  //上半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
//------------上半屏和下半屏的分割线-----------
0x00,0x00,0x00,0x00,0x00,0x00,  //下半屏
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,
};
/* 注释二:
* 以下5个变量记录一个参数的5种信息，包括小数点的数量，小数点个数，数据的位置，数组具体值，整数个数
*/
unsigned char ucDotCnt_1=0;  //记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效
unsigned char ucDotBitS_1=0; //记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任何输入按键则无效
unsigned char ucWdPartCnt_1=0; //记录当前输入的数据在数组中的位置。
unsigned char ucDataBuffer_1[6]={0,10,10,10,10,10}; //一项的BCD码数组缓冲
unsigned char ucIntCnt_1=0; //记录当前输入的整数个数，如果整数的个数如果超过规定ucIntCntMax位，此时再按任何输入按键则无效
unsigned long ulData_1=0; //用一个long变量表示BCD码的具体数值。
unsigned char ucDotCnt_2=0;  //记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效
unsigned char ucDotBitS_2=0; //记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任何输入按键则无效
unsigned char ucWdPartCnt_2=0; //记录当前输入的数据在数组中的位置。
unsigned char ucDataBuffer_2[6]={0,10,10,10,10,10}; //一项的BCD码数组缓冲
unsigned char ucIntCnt_2=0; //记录当前输入的整数个数，如果整数的个数如果超过规定ucIntCntMax位，此时再按任何输入按键则无效
unsigned long ulData_2=0; //用一个long变量表示BCD码的具体数值。
```

```
unsigned char ucKeyStep=1;  //按键扫描步骤变量
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiKeyTimeCnt=0; //按键去抖动延时计数器
unsigned char ucKeyLock=0; //按键触发后自锁的变量标志
unsigned char ucRowRecord=1; //记录当前扫描到第几列了
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char ucWd=1; //窗口变量
unsigned char ucPart=1; //局部变量 0代表没有选中任何一行，其它数值1到4代表选中某一行
unsigned char ucWd1Update=1; //窗口1的整屏更新显示变量       1代表更新显示，响应函数内部会清零
unsigned char ucWd1Part1Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
unsigned char ucWd1Part2Update=0; //窗口1的第2行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
unsigned char ucWd1Part3Update=0; //窗口1的第1行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
unsigned char ucWd1Part4Update=0; //窗口1的第2行局部更新显示变量  1代表更新显示，响应函数内部会自动把它清零
void main()
  {
        initial_myself();       //第一区,上电后马上初始化
        delay_long(100);        //一线，延时线。延时一段时间
        initial_peripheral();  //第二区,上电后延时一段时间再初始化
        while(1)   //第三区
        {
                    key_service(); //按键服务程序
            lcd_display_service(); //应用层面的液晶屏显示程序
        }
}
void initial_myself()  //第一区 上电后马上初始化
{
   beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
   TMOD=0x01;  //设置定时器0为工作方式1
   TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
   TL0=0x2f;
}
void initial_peripheral() //第二区 上电后延时一段时间再初始化
{
    LCDInit(); //初始化12864 内部包含液晶模块的复位
    EA=1;      //开总中断
    ET0=1;     //允许定时中断
    TR0=1;     //启动定时中断
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  key_scan();//按键扫描函数 放在定时中断里
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
```

```
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
         beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  TH0=0xf8;   //重装初始值(65535-2000)=63535=0xf82f
  TL0=0x2f;
  TR0=1;  //开中断
}
void key_scan()//按键扫描函数 放在定时中断里
{
  switch(ucKeyStep)
  {
     case 1:   //按键扫描输出第ucRowRecord列低电平
              if(ucRowRecord==1)  //第一列输出低电平
                  {
              key_dr1=0;
              key_dr2=1;
              key_dr3=1;
              key_dr4=1;
                  }
               else if(ucRowRecord==2)  //第二列输出低电平
                  {
              key_dr1=1;
              key_dr2=0;
              key_dr3=1;
              key_dr4=1;
                  }
               else if(ucRowRecord==3)  //第三列输出低电平
                  {
              key_dr1=1;
              key_dr2=1;
              key_dr3=0;
              key_dr4=1;
                  }
               else   //第四列输出低电平
                  {
              key_dr1=1;
              key_dr2=1;
              key_dr3=1;
              key_dr4=0;
                  }
          uiKeyTimeCnt=0;  //延时计数器清零
          ucKeyStep++;     //切换到下一个运行步骤
              break;
     case 2:     //此处的小延时用来等待刚才列输出信号稳定，再判断输入信号。不是去抖动延时。
          uiKeyTimeCnt++;
                  if(uiKeyTimeCnt>1)
```

```
                {
                   uiKeyTimeCnt=0;
            ucKeyStep++;      //切换到下一个运行步骤
                }
             break;
    case 3:
        if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==1)
        {
           ucKeyStep=1;  //如果没有按键按下，返回到第一个运行步骤重新开始扫描
           ucKeyLock=0;  //按键自锁标志清零
           uiKeyTimeCnt=0; //按键去抖动延时计数器清零，此行非常巧妙

                      ucRowRecord++;  //输出下一列
                      if(ucRowRecord>4)
                      {
                         ucRowRecord=1; //依次输出完四列之后，继续从第一列开始输出低电平
                      }
        }
                else if(ucKeyLock==0)  //有按键按下，且是第一次触发
                {
                   if(key_sr1==0&&key_sr2==1&&key_sr3==1&&key_sr4==1)
                       {
                          uiKeyTimeCnt++;  //去抖动延时计数器
                              if(uiKeyTimeCnt>const_key_time)
                              {
                                 uiKeyTimeCnt=0;
                                 ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零

                     if(ucRowRecord==1)  //第一列输出低电平
                         {
                                    ucKeySec=1;  //触发1号键 对应朱兆祺学习板的S1键
                         }
                     else if(ucRowRecord==2)  //第二列输出低电平
                         {
                                    ucKeySec=2;  //触发2号键 对应朱兆祺学习板的S2键
                         }
                     else if(ucRowRecord==3)  //第三列输出低电平
                         {
                                    ucKeySec=3;  //触发3号键 对应朱兆祺学习板的S3键
                         }
                     else   //第四列输出低电平
                         {
                                    ucKeySec=4;  //触发4号键 对应朱兆祺学习板的S4键
                         }
                              }

                     }
                   else if(key_sr1==1&&key_sr2==0&&key_sr3==1&&key_sr4==1)
                       {
                          uiKeyTimeCnt++;  //去抖动延时计数器
```

```
                if(uiKeyTimeCnt>const_key_time)
                {
                   uiKeyTimeCnt=0;
                   ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=5;  //触发5号键 对应朱兆祺学习板的S5键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=6;  //触发6号键 对应朱兆祺学习板的S6键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=7;  //触发7号键 对应朱兆祺学习板的S7键
      }
  else   //第四列输出低电平
      {
                ucKeySec=8;  //触发8号键 对应朱兆祺学习板的S8键
      }
           }

    }
else if(key_sr1==1&&key_sr2==1&&key_sr3==0&&key_sr4==1)
    {
       uiKeyTimeCnt++;  //去抖动延时计数器
           if(uiKeyTimeCnt>const_key_time)
           {
              uiKeyTimeCnt=0;
              ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
  if(ucRowRecord==1)  //第一列输出低电平
      {
                ucKeySec=9;  //触发9号键 对应朱兆祺学习板的S9键
      }
  else if(ucRowRecord==2)  //第二列输出低电平
      {
                ucKeySec=10;  //触发10号键 对应朱兆祺学习板的S10键
      }
  else if(ucRowRecord==3)  //第三列输出低电平
      {
                ucKeySec=11;  //触发11号键 对应朱兆祺学习板的S11键
      }
  else   //第四列输出低电平
      {
                ucKeySec=12;  //触发12号键 对应朱兆祺学习板的S12键
      }
           }
```

```
                        }
                    else if(key_sr1==1&&key_sr2==1&&key_sr3==1&&key_sr4==0)
                        {
                           uiKeyTimeCnt++;  //去抖动延时计数器
                               if(uiKeyTimeCnt>const_key_time)
                               {
                                  uiKeyTimeCnt=0;
                                  ucKeyLock=1;//自锁按键置位,避免一直触发,只有松开按键,此标志位才会被清
零
                      if(ucRowRecord==1)  //第一列输出低电平
                          {
                                    ucKeySec=13;  //触发13号键 对应朱兆祺学习板的S13键
                          }
                      else if(ucRowRecord==2)  //第二列输出低电平
                          {
                                    ucKeySec=14;  //触发14号键 对应朱兆祺学习板的S14键
                          }
                      else if(ucRowRecord==3)  //第三列输出低电平
                          {
                                    ucKeySec=15;  //触发15号键 对应朱兆祺学习板的S15键
                          }
                      else   //第四列输出低电平
                          {
                                    ucKeySec=16;  //触发16号键 对应朱兆祺学习板的S16键
                          }
                               }

                        }

                }
            break;
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 数字1 对应朱兆祺学习板的S1键
          key_number_input(1); //输入数字按键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 2:// 数字2 对应朱兆祺学习板的S2键
          key_number_input(2); //输入数字按键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3:// 数字3 对应朱兆祺学习板的S3键
          key_number_input(3); //输入数字按键
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
```

```
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 4:// 数字4 对应朱兆祺学习板的S4键
        key_number_input(4); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 5:// 数字5 对应朱兆祺学习板的S5键
        key_number_input(5); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 6:// 数字6 对应朱兆祺学习板的S6键
        key_number_input(6); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 7:// 数字7 对应朱兆祺学习板的S7键
        key_number_input(7); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 8: //数字8 对应朱兆祺学习板的S8键
        key_number_input(8); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 9:// 数字9 对应朱兆祺学习板的S9键
        key_number_input(9); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 10:// 数字0  对应朱兆祺学习板的S10键
        key_number_input(0); //输入数字按键
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 11:// 小数点按键 对应朱兆祺学习板的S11键
        key_number_input(11); //输入数字按键  11代表小数点
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
case 12:// 光标移动按键 对应朱兆祺学习板的S12键
        //ucPart++;
                //if(ucPart>4)
                //{
                //   ucPart=1;
                //}
                switch(ucPart)  //根据不同的行来进行不同的操作
                {
```

```
                case 1:
                        ucPart=2;

                ucWd1Part1Update=1;   //实时更新显示数组
                ucWd1Part2Update=1;   //实时更新显示数值
                        break;
                case 2:
                        ucPart=3;
                ucWd1Part2Update=1;   //实时更新显示数组
                ucWd1Part3Update=1;   //实时更新显示数值
                        break;
                case 3:
                        ucPart=4;
                ucWd1Part3Update=1;   //实时更新显示数组
                ucWd1Part4Update=1;   //实时更新显示数值
                        break;
                case 4:
                        ucPart=1;
                ucWd1Part4Update=1;   //实时更新显示数组
                ucWd1Part1Update=1;   //实时更新显示数值
                        break;
                }
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 13:// 加按键 对应朱兆祺学习板的S13键
                switch(ucPart)  //根据不同的行来进行不同的操作
                {
                case 2:
                ulData_1++;
                if(ulData_1>99999)
                {
                ulData_1=99999;
                }
                data_to_buffer(ulData_1,  //把数值转换成数组，这是本节核心函数，请好好研究此函数的具体
功能。
                                                2,  //小数点最大个数
                                                6,  //数组缓冲最大个数
                                                &ucDotCnt_1,
                                                &ucDotBitS_1,
                                                &ucWdPartCnt_1,
                                                ucDataBuffer_1);  //被转换成的数组
                ucWd1Part1Update=1;   //实时更新显示数组
                ucWd1Part2Update=1;   //实时更新显示数值
                        break;
                case 4:
                ulData_2++;
                if(ulData_2>99999)
                {
                ulData_2=99999;
```

```
                }
                data_to_buffer(ulData_2,  //把数值转换成数组，这是本节核心函数，请好好研究此函数的具体
功能。
                                          2,  //小数点最大个数
                                              6,  //数组缓冲最大个数
                                              &ucDotCnt_2,
                                              &ucDotBitS_2,
                                              &ucWdPartCnt_2,
                                              ucDataBuffer_2);  //被转换成的数组
                ucWd1Part3Update=1;   //实时更新显示数组
                ucWd1Part4Update=1;   //实时更新显示数值
                            break;
                }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 14:// 减按键  对应朱兆祺学习板的S14键
                switch(ucPart)  //根据不同的行来进行不同的操作
                {
                   case 2:
                ulData_1--;
                if(ulData_1>99999) //unsigned long类型的变量0减去1会变成0xffffffff
                {
                   ulData_1=0;
                }
                data_to_buffer(ulData_1,  //把数值转换成数组，这是本节核心函数，请好好研究此函数的具体
功能。
                                          2,  //小数点最大个数
                                              6,  //数组缓冲最大个数
                                              &ucDotCnt_1,
                                                &ucDotBitS_1,
                                              &ucWdPartCnt_1,
                                              ucDataBuffer_1);  //被转换成的数组
                ucWd1Part1Update=1;   //实时更新显示数组
                ucWd1Part2Update=1;   //实时更新显示数值
                            break;
                   case 4:
                ulData_2--;
                if(ulData_2>99999) //unsigned long类型的变量0减去1会变成0xffffffff
                {
                   ulData_2=0;
                }
                data_to_buffer(ulData_2,  //把数值转换成数组，这是本节核心函数，请好好研究此函数的具体
功能。
                                          2,  //小数点最大个数
                                              6,  //数组缓冲最大个数
                                              &ucDotCnt_2,
                                                &ucDotBitS_2,
                                              &ucWdPartCnt_2,
                                              ucDataBuffer_2);  //被转换成的数组
```

```
                    ucWd1Part3Update=1;    //实时更新显示数组
                    ucWd1Part4Update=1;    //实时更新显示数值
                                break;
                    }
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 15:// 本节暂时不用 对应朱兆祺学习板的S15键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
    case 16:// 清除按键 对应朱兆祺学习板的S16键
            key_delete_input(); //删除按键
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
  }
}
void key_number_input(unsigned char ucKeyNumber) //输入数字按键
{
    switch(ucWd)
    {
       case 1:    //第1窗口。本节程序只有1个窗口
             switch(ucPart)
             {
                 case 1:  //1窗口第1行
                      set_data(ucKeyNumber,
                                                  2,  //小数点最大个数
                                                      6,  //数组缓冲最大个数
                                                  &ucDotCnt_1,
                                                      &ucDotBitS_1,
                                                      &ucWdPartCnt_1,
                                                      ucDataBuffer_1,
                                                      3, //整数部分的最大个数
                                                      &ucIntCnt_1);
                                              ulData_1=buffer_to_data(6,2,ucDataBuffer_1); //把带小数点的
BCD码数组转换成long数值。
                      ucWd1Part1Update=1;//第一行局部更新显示
                                              ucWd1Part2Update=1;//第二行局部更新显示
                      break;
                 case 3:  //1窗口第3行
                      set_data(ucKeyNumber,
                                                  2,  //小数点最大个数
                                                      6,  //数组缓冲最大个数
                                                  &ucDotCnt_2,
                                                      &ucDotBitS_2,
                                                      &ucWdPartCnt_2,
                                                      ucDataBuffer_2,
                                                      3, //整数部分的最大个数
                                                      &ucIntCnt_2);
```

```
                                        ulData_2=buffer_to_data(6,2,ucDataBuffer_2); //把带小数点的
BCD码数组转换成long数值。
                        ucWd1Part3Update=1;//第三行局部更新显示
                                        ucWd1Part4Update=1;//第四行局部更新显示
                        break;
                }

                break;
    }

}
/* 注释三:
* 涉及到参数的4种信息，包括小数点的数量，小数点的个数，数据的位置，数组具体值，整数的数量，整数的个数
，以及它们之间的相互作用关系。
* 以下参数，指针类型的参数是让代入的全局变量在退出函数后维持它当前最新更改的数值不变。
* 第1个参数ucKeyNumberTemp是当前按键输入的数值。
* 第2个参数ucDotBitMax是限定被设置参数的小数点最大位数。
* 第3个参数ucDataCntMax是限定被设置参数的最大数组个数。
* 第4个参数*p_ucDotCnt是记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时
再按小数点按键则无效。
* 第5个参数*p_ucDotBitS是记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任
何输入按键则无效
* 第6个参数*p_ucWdPartCnt是记录当前输入的数据在数组中的位置，方便锁定每次按键输入的数字显示位置。
* 第7个参数*p_ucSetDataBuffer是BCD码数组缓冲的具体数字内容。
* 第8个参数ucIntCntMax是限定被设置参数的整数部分的最大位数。
* 第9个参数*p_ucIntCnt是记录当前输入的整数部分个数，如果整数部分的个数如果超过规定ucIntCntMax位，此时再
按任何输入按键则无效
*/
void set_data(unsigned char ucKeyNumberTemp,
              unsigned char ucDotBitMax,
              unsigned char ucDataCntMax,
              unsigned char *p_ucDotCnt,
              unsigned char *p_ucDotBitS,
                          unsigned char *p_ucWdPartCnt,
                          unsigned char *p_ucSetDataBuffer,
                          unsigned char ucIntCntMax,
                          unsigned char *p_ucIntCnt)
{
                    unsigned int i;
                    if(ucKeyNumberTemp==11) //等于小数点
                    {
                       if(ucDotBitMax==0) //如果限定的小数点最大数是0，就意味着此数据不允许带小数点，必
须是整数。
                       {
                           return; //直接返回退出
                       }
                       else if(*p_ucDotCnt>0)  //小数点个数大于0，意味着当前数组已经包含了小数点，此时再
输入小数点则无效。
                       {
                           return; //直接返回退出
```

```
                    }
                    else  //否则有效，记录当前已经包含一个小数点的信息。
                    {
                        *p_ucDotCnt=1;  //只能包含一个小数点
                    }
                }
                else  //如果输入的不是小数点
                {
                    if(*p_ucDotCnt==1) //如果之前已经输入了一个小数点，那么此时输入的数字就是小数点后的数据
                                        {
                        if(*p_ucDotBitS<ucDotBitMax) //如果小数点位数还没超过最大限制位数，则继续加1记录当前小数点位数。
                        {
                            *p_ucDotBitS=(*p_ucDotBitS)+1;
                        }
                        else //如果小数点位数已经超过允许的范围，则输入的按键无效，直接退出。
                        {
                           return; //直接返回退出
                        }
                                    }
                                        else if(*p_ucIntCnt<ucIntCntMax)//如果之前没有输入小数点，那么输入的就是整数个数超，整数个数没有超过极限
                                        {
                                            *p_ucIntCnt=(*p_ucIntCnt)+1;
                                        }
                                        else //整数个数超过极限
                                        {
                          return; //直接返回退出
                                        }
                }


                if(*p_ucWdPartCnt==0&&p_ucSetDataBuffer[0]==0&&ucKeyNumberTemp!=11)  //如果当前默认位置是第0个位置，并且默认第0个数据是0，并且当前的按键输入不是小数点，则不用移位
                {
                    ;
                }
                else  //否则，移位
                {
                   for(i=0;i<(ucDataCntMax-1);i++)  //移位
                   {
                      p_ucSetDataBuffer[ucDataCntMax-1-i]=p_ucSetDataBuffer[ucDataCntMax-2-i];
                   }
                   *p_ucWdPartCnt=(*p_ucWdPartCnt)+1;
                }
                p_ucSetDataBuffer[0]=ucKeyNumberTemp; //当前输入的数字或者小数点永远在第右边第0个位置。
```

```
}
/* 注释四:
* 功能: 把一个带小数点的BCD码数组转换成一个long类型的数值。
* 第1个参数ucConverDataSize是这个数组的最大有效个数。
* 第2个参数ucConverDotCnt是这个数组要转换成的long数值带几个小数点
* 第3个参数*p_ucConverBuffer是具体此数组的数据
* 函数最后返回被转换的long数值。
*/
unsigned long buffer_to_data(unsigned char ucConverDataSize,unsigned char ucConverDotCnt,unsigned char
*p_ucConverBuffer)
{
   unsigned long ulConverResult=0;
   unsigned long ulConverResultTemp=0;
   unsigned char ucConverResultBuffer[6]; //因为本节内容的ucConverDataSize是6，所以取6.
   unsigned char i;
   unsigned char j;
   unsigned char ucConverFlag;
   for(i=0;i<ucConverDataSize;i++)
   {
      ucConverResultBuffer[i]=0;  //先把临时缓冲区清零
   }
   j=0;
   ucConverFlag=0;
   for(i=0;i<ucConverDataSize;i++)
   {
       if(p_ucConverBuffer[i]==11) //小数点
       {
          ucConverFlag=i; //记录小数点的位置
       }
       else if(p_ucConverBuffer[i]<10)
       {
          ucConverResultBuffer[j]=p_ucConverBuffer[i];  //提取数组中的有效数字
          j++;
       }
   }
   for(i=0;i<ucConverDataSize;i++)   //通过处理每一位从而合成一个long类型的数值
   {
       ulConverResultTemp=0;
       ulConverResultTemp=ucConverResultBuffer[i];
       for(j=0;j<i;j++)
       {
           ulConverResultTemp=ulConverResultTemp*10;  //把每一位对应的进位扩大到对应的倍数
       }
       ulConverResult=ulConverResult+ulConverResultTemp;
   }
   for(i=ucConverFlag;i<ucConverDotCnt;i++) //根据数组小数点的位置和实际要转换成的小数点个数，来扩大到对
应的倍数。
   {
      ulConverResult=ulConverResult*10;
   }
```

```
   return ulConverResult;
}
/* 注释五:
* 本节的核心函数，值得好好研究!
* 功能: 把一个long类型的数值转换成一个带小数点的BCD码数组
* 第1个参数ulWillConverData是即将被转换的unsigned long类型数值。
* 第2个参数ucConverDotCnt是这个数值带几个小数点
* 第3个参数ucConverDataSize是这个数组的最大有效个数。
* 第4个参数*p_ucDotCnt是记录当前输入的小数点数量，如果小数点的数量不为0，说明当前数组已包含小数点，此时再按小数点按键则无效。
* 第5个参数*p_ucDotBitS是记录当前输入的小数点个数，如果小数点的个数如果超过规定ucDotBitMax位，此时再按任何输入按键则无效
* 第6个参数*p_ucWdPartCnt是记录当前输入的数据在数组中的位置，方便锁定每次按键输入的数字显示位置。
* 第7个参数*p_ucConverBuffer是具体此数组的数据。
*/
void data_to_buffer(unsigned long ulWillConverData,
                    unsigned char ucConverDotCnt,
                                  unsigned char ucConverDataSize,
                                  unsigned char *p_ucDotCnt,
                                  unsigned char *p_ucDotBitS,
                                  unsigned char *p_ucWdPartCnt,
                                  unsigned char *p_ucConverBuffer)
{
   unsigned char ucConverResultBuffer[6]; //因为本节内容的ucConverDataSize是6，所以取6.
   unsigned char i;
   unsigned char ucValidaDotCnt=0;
   if(ucConverDotCnt==0)  //没有小数点
   {
      *p_ucDotCnt=0;   //当前没有输入小数点的标志
      *p_ucDotBitS=0;  //当前输入的小数点个数是0
      ucConverResultBuffer[5]=10;  //没有小数点的时候，第5位必然是显示空格
          //以下是具体把数值转换成数组，不需要显示的高位填入10表示显示空格
      if(ulWillConverData>=10000)
      {
         ucConverResultBuffer[4]=ulWillConverData%100000/10000;
      }
      else
      {
         ucConverResultBuffer[4]=10;
      }
      if(ulWillConverData>=1000)
      {
         ucConverResultBuffer[3]=ulWillConverData%10000/1000;
      }
      else
      {
         ucConverResultBuffer[3]=10;
      }
      if(ulWillConverData>=100)
      {
```

```
        ucConverResultBuffer[2]=u1WillConverData%1000/100;
    }
    else
    {
        ucConverResultBuffer[2]=10;
    }
    if(u1WillConverData>=10)
    {
        ucConverResultBuffer[1]=u1WillConverData%100/10;
    }
    else
    {
        ucConverResultBuffer[1]=10;
    }
    ucConverResultBuffer[0]=u1WillConverData%10;
}
else if(ucConverDotCnt==1)  //1位小数点
{
    *p_ucDotCnt=1;  //当前已经有输入小数点的标志
    *p_ucDotBitS=1; //当前输入的小数点个数是1
    ucConverResultBuffer[1]=11;  //第1位填入小数点11
        //以下是具体把数值转换成数组，不需要显示的高位填入10表示显示空格
    if(u1WillConverData>=10000)
    {
        ucConverResultBuffer[5]=u1WillConverData%100000/10000;
    }
    else
    {
        ucConverResultBuffer[5]=10;
    }
    if(u1WillConverData>=1000)
    {
        ucConverResultBuffer[4]=u1WillConverData%10000/1000;
    }
    else
    {
        ucConverResultBuffer[4]=10;
    }
    if(u1WillConverData>=100)
    {
        ucConverResultBuffer[3]=u1WillConverData%1000/100;
    }
    else
    {
        ucConverResultBuffer[3]=10;
    }
    ucConverResultBuffer[2]=u1WillConverData%100/10;
    ucConverResultBuffer[0]=u1WillConverData%10;
}
else if(ucConverDotCnt==2)  //2位小数点
```

```
    {
       *p_ucDotCnt=1;  //当前已经有输入小数点的标志
       *p_ucDotBitS=2; //当前输入的小数点个数是2
       ucConverResultBuffer[2]=11; //第2位填入小数点11
           //以下是具体把数值转换成数组，不需要显示的高位填入10表示显示空格
       if(ulWillConverData>=10000)
       {
          ucConverResultBuffer[5]=ulWillConverData%100000/10000;
       }
       else
       {
          ucConverResultBuffer[5]=10;
       }
       if(ulWillConverData>=1000)
       {
          ucConverResultBuffer[4]=ulWillConverData%10000/1000;
       }
       else
       {
          ucConverResultBuffer[4]=10;
       }
       ucConverResultBuffer[3]=ulWillConverData%1000/100;
       ucConverResultBuffer[1]=ulWillConverData%100/10;
       ucConverResultBuffer[0]=ulWillConverData%10;
    }
    ucValidaDotCnt=0;
    for(i=0;i<ucConverDataSize;i++)
    {
       if(ucConverResultBuffer[i]!=10)  //统计数组有效的BCD码位数
       {
          ucValidaDotCnt++;
       }
       p_ucConverBuffer[i]=ucConverResultBuffer[i];  //把转换的结果传输给实际的数组用来外部显示

    }
    *p_ucWdPartCnt=ucValidaDotCnt-1;  //当前显示的实际位置
}
void key_delete_input(void) //删除按键
{
     static unsigned int i;
    switch(ucWd)
    {
       case 1:   //第1窗口。本节程序只有1个窗口
            switch(ucPart)
            {
               case 1:  //1窗口第1行
                     //清零
                                       ulData_1=0; //long数值清零
                                        ucIntCnt_1=0;
                     ucDotBitS_1=0;
```

```
        ucDotCnt_1=0;
        ucWdPartCnt_1=0;
        for(i=0;i<6;i++)
        {
           ucDataBuffer_1[i]=10;
        }
        ucDataBuffer_1[0]=0; //第0个位置填入0

        ucWd1Part1Update=1;//第一行局部更新显示
                        ucWd1Part2Update=1;//第二行局部更新显示
        break;
case 2:  //1窗口第2行
        //清零
                            ulData_1=0; //long数值清零
                             ucIntCnt_1=0;
        ucDotBitS_1=0;
        ucDotCnt_1=0;
        ucWdPartCnt_1=0;
        for(i=0;i<6;i++)
        {
           ucDataBuffer_1[i]=10;
        }
        ucDataBuffer_1[0]=0; //第0个位置填入0

        ucWd1Part1Update=1;//第一行局部更新显示
                        ucWd1Part2Update=1;//第二行局部更新显示
        break;
case 3:  //1窗口第3行
        //清零
                            ulData_2=0; //long数值清零
                             ucIntCnt_2=0;
        ucDotBitS_2=0;
        ucDotCnt_2=0;
        ucWdPartCnt_2=0;
        for(i=0;i<6;i++)
        {
           ucDataBuffer_2[i]=10;
        }
        ucDataBuffer_2[0]=0; //第0个位置填入0

        ucWd1Part3Update=1;//第三行局部更新显示
                        ucWd1Part4Update=1;//第四行局部更新显示
        break;
case 4:  //1窗口第4行
        //清零
                            ulData_2=0; //long数值清零
                             ucIntCnt_2=0;
        ucDotBitS_2=0;
        ucDotCnt_2=0;
        ucWdPartCnt_2=0;
```

```
                    for(i=0;i<6;i++)
                    {
                       ucDataBuffer_2[i]=10;
                    }
                    ucDataBuffer_2[0]=0; //第0个位置填入0

                    ucWd1Part3Update=1;//第三行局部更新显示
                                    ucWd1Part4Update=1;//第四行局部更新显示
                    break;
            }

            break;

    }

}
unsigned char *number_to_matrix(unsigned char  ucBitNumber)
{
    unsigned char *p_ucAnyNumber;  //此指针根据ucBitNumber数值的大小，分别调用不同的字库。
        switch(ucBitNumber)  //根据ucBitNumber数值的大小，分别调用不同的字库。
        {
            case 0:
             p_ucAnyNumber=Zf816_0;
                     break;
            case 1:
             p_ucAnyNumber=Zf816_1;
                     break;
            case 2:
             p_ucAnyNumber=Zf816_2;
                     break;
            case 3:
             p_ucAnyNumber=Zf816_3;
                     break;
            case 4:
             p_ucAnyNumber=Zf816_4;
                     break;
            case 5:
             p_ucAnyNumber=Zf816_5;
                     break;
            case 6:
             p_ucAnyNumber=Zf816_6;
                     break;
            case 7:
             p_ucAnyNumber=Zf816_7;
                     break;
            case 8:
             p_ucAnyNumber=Zf816_8;
                     break;
            case 9:
             p_ucAnyNumber=Zf816_9;
```

```
                break;
            case 10:  //空格
             p_ucAnyNumber=Zf816_nc;
                break;
                   case 11:   //小数点
             p_ucAnyNumber=Zf816_dot;
                break;
               default:   //如果上面的条件都不符合，那么默认指向空字模
             p_ucAnyNumber=Zf816_nc;
                break;
        }
    return p_ucAnyNumber;  //返回转换结束后的指针
}
void lcd_display_service(void) //应用层面的液晶屏显示程序
{
    static unsigned char *p_ucAnyNumber; //经过数字转换成字模后，分解变量的某位字模首地址
    static unsigned char ucCursorFlag;  //光标标志,也就是反显的标志,它是根据局部变量ucPart来定的
    static unsigned int i;
        static unsigned char ucDataBuffer_temp[6]; //分解一个10进制的long类型数据的每一位
    switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
    {
        case 1:   //显示窗口1的数据
               if(ucWd1Update==1)  //窗口1整屏更新，里面只放那些不用经常刷新显示的内容
               {
                    ucWd1Update=0;  //及时清零，避免一直更新
                    ucWd1Part1Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    ucWd1Part2Update=1; //激活窗口1的第2行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    ucWd1Part3Update=1; //激活窗口1的第1行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    ucWd1Part4Update=1; //激活窗口1的第2行局部更新显示变量，这里在前面数码管显示框架上
有所改进
                    display_clear(0x00); // 清屏操作， 全部显示空填充0x00，全部显示点阵用0xff。
                    clear_all_canvas();  //把画布全部清零
                    display_lattice(0,0,Hz1616_yi,0,2,16,0);    //一项数组
                    display_lattice(1,0,Hz1616_xiang,0,2,16,0);
                    display_lattice(2,0,Hz1616_shu,0,2,16,0);
                    display_lattice(3,0,Hz1616_zhu,0,2,16,0);
                    display_lattice(4,0,Zf816_mao_hao,0,1,16,0); //冒号
                    display_lattice(0,16,Hz1616_yi,0,2,16,0);    //一项数值
                    display_lattice(1,16,Hz1616_xiang,0,2,16,0);
                    display_lattice(2,16,Hz1616_shu,0,2,16,0);
                    display_lattice(3,16,Hz1616_zhi,0,2,16,0);
                    display_lattice(4,16,Zf816_mao_hao,0,1,16,0); //冒号

                    display_lattice(8,0,Hz1616_er,0,2,16,0);    //二项数组
                    display_lattice(9,0,Hz1616_xiang,0,2,16,0);
                    display_lattice(10,0,Hz1616_shu,0,2,16,0);
                    display_lattice(11,0,Hz1616_zhu,0,2,16,0);
```

```
            display_lattice(12,0,Zf816_mao_hao,0,1,16,0); //冒号
            display_lattice(8,16,Hz1616_er,0,2,16,0);     //二项数值
            display_lattice(9,16,Hz1616_xiang,0,2,16,0);
            display_lattice(10,16,Hz1616_shu,0,2,16,0);
            display_lattice(11,16,Hz1616_zhi,0,2,16,0);
            display_lattice(12,16,Zf816_mao_hao,0,1,16,0); //冒号
        }
        if(ucWd1Part1Update==1) //窗口1的第1行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
            ucWd1Part1Update=0; //及时清零，避免一直更新
            if(ucPart==1) //被选中
            {
               ucCursorFlag=1; //反显 显示
            }
            else //没被选中
            {
                ucCursorFlag=0; //正常 显示
            }

            for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
            {
                p_ucAnyNumber=number_to_matrix(ucDataBuffer_1[5-i]);
                insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地
址，一共可以显示6个字符，因此取值范围是0到5
            }
            display_lattice(5,0,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最后
的参数0是偏移量
        }
        if(ucWd1Part2Update==1) //窗口1的第2行局部更新显示变量,里面放一些经常需要刷新显示的内容
        {
            ucWd1Part2Update=0; //及时清零，避免一直更新
            if(ucPart==2) //被选中
            {
               ucCursorFlag=1; //反显 显示
            }
            else //没被选中
            {
                ucCursorFlag=0; //正常 显示
            }
            if(ulData_1>=10000)
                                {
                                   ucDataBuffer_temp[5]=ulData_1%100000/10000;
                                }
                                else
                                {
                                   ucDataBuffer_temp[5]=10; //空格
                                }
            if(ulData_1>=1000)
                                {
                                      ucDataBuffer_temp[4]=ulData_1%10000/1000;
```

```
                }
                                        else
                                        {
                                              ucDataBuffer_temp[4]=10; //空格
                }
                                        ucDataBuffer_temp[3]=ulData_1%1000/100;
                                        ucDataBuffer_temp[2]=11;  //11代表小数点
                                        ucDataBuffer_temp[1]=ulData_1%100/10;
                                        ucDataBuffer_temp[0]=ulData_1%10/1;

                for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                {
                    p_ucAnyNumber=number_to_matrix(ucDataBuffer_temp[5-i]);
                    insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地址，一共可以显示6个字符，因此取值范围是0到5
                }
                display_lattice(5,16,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最后的参数0是偏移量
            }
            if(ucWd1Part3Update==1) //窗口1的第3行局部更新显示变量,里面放一些经常需要刷新显示的内容
            {
                ucWd1Part3Update=0; //及时清零，避免一直更新
                if(ucPart==3) //被选中
                {
                   ucCursorFlag=1; //反显 显示
                }
                else //没被选中
                {
                    ucCursorFlag=0; //正常 显示
                }

                for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                {
                    p_ucAnyNumber=number_to_matrix(ucDataBuffer_2[5-i]);
                    insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地址，一共可以显示6个字符，因此取值范围是0到5
                }
                display_lattice(13,0,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最后的参数0是偏移量
            }
            if(ucWd1Part4Update==1) //窗口1的第4行局部更新显示变量,里面放一些经常需要刷新显示的内容
            {
                ucWd1Part4Update=0; //及时清零，避免一直更新
                if(ucPart==4) //被选中
                {
                   ucCursorFlag=1; //反显 显示
                }
                else //没被选中
                {
                    ucCursorFlag=0; //正常 显示
```

```
                        }
                        if(ulData_2>=10000)
                                                {
                                                   ucDataBuffer_temp[5]=ulData_2%100000/10000;
                                                }
                                                else
                                                {
                                                   ucDataBuffer_temp[5]=10; //空格
                                                }
                        if(ulData_2>=1000)
                                                {
                                                      ucDataBuffer_temp[4]=ulData_2%10000/1000;
                        }
                                                else
                                                {
                                                      ucDataBuffer_temp[4]=10; //空格
                        }
                                                ucDataBuffer_temp[3]=ulData_2%1000/100;
                                                ucDataBuffer_temp[2]=11;  //11代表小数点
                                                ucDataBuffer_temp[1]=ulData_2%100/10;
                                                ucDataBuffer_temp[0]=ulData_2%10/1;

                        for(i=0;i<6;i++) //把每个数组缓冲的字模依次插入画布
                        {
                            p_ucAnyNumber=number_to_matrix(ucDataBuffer_temp[5-i]);
                            insert_buffer_to_canvas(i,0,p_ucAnyNumber,0,1,16);  //这里的i是画布的横向地
址，一共可以显示6个字符，因此取值范围是0到5
                        }
                        display_lattice(13,16,ucCanvasBuffer,ucCursorFlag,6,16,0);    //显示整屏的画布,最
后的参数0是偏移量
                }

                break;
        //本程序只有1个窗口，所以只有一个case 1，如果要增加窗口，就直接增加 case 2，case 3...
    }
}
void clear_all_canvas(void)  //把画布全部清零
{
   unsigned int j=0;
   unsigned int i=0;
   for(j=0;j<16;j++)  //这里的16表示画布有16行
   {
      for(i=0;i<4;i++) //这里的4表示画布每行有4个字节
      {
                  ucCanvasBuffer[j*4+i]=0x00;
      }
   }
}
void display_clear(unsigned char ucFillDate) // 清屏  全部显示空填充0x00   全部显示点阵用0xff
{
```

```
    unsigned char x,y;
    WriteCommand(0x34);  //关显示缓冲指令
    WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是这样写的
    y=0;
    while(y<32)  //y轴的范围0至31
    {
         WriteCommand(y+0x80);         //垂直地址
         WriteCommand(0x80);           //水平地址
         for(x=0;x<32;x++)  //256个横向点，有32个字节
         {
            LCDWriteData(ucFillDate);
         }
         y++;
    }
    WriteCommand(0x36); //开显示缓冲指令
}
/* 注释六:
* 注意，这节内容的画布跟第79节前面的画布大小不一样，第79节前面的横向是4个字节，这节的横向是6个字节。
* 把字模插入画布的函数.
* 这是本节的核心函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是在画布中的坐标体系。
* x的范围是0至5，因为画布的横向只要6个字节。y的范围是0至15，因为画布的纵向只有16行。
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
*/
void insert_buffer_to_canvas(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char ucFbFlag,unsigned int x_amount,unsigned int y_amount)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
   for(j=0;j<y_amount;j++)
   {
      for(i=0;i<x_amount;i++)
      {
              ucTemp=ucArray[j*x_amount+i];
              if(ucFbFlag==0)
              {
                 ucCanvasBuffer[(y+j)*6+x+i]=ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向是4个字节，要稍微注意的。
              }
              else
              {
                 ucCanvasBuffer[(y+j)*6+x+i]=~ucTemp; //这里的6代表画布每一行只有6个字节。前面章节的横向是4个字节，要稍微注意的。
              }
      }
   }
```

```
}
/* 注释七:
* 显示任意点阵函数.
* 注意，本函数在前几节的基础上多增加了第7个参数uiOffSetAddr，它是偏移地址。
* 对于这个函数，读者尤其要搞懂x_amount和y_amount对应的显示关系。
* 第1，2个参数x,y是坐标体系。x的范围是0至15，y的范围是0至31.
* 第3个参数*ucArray是字模的数组。
* 第4个参数ucFbFlag是反白显示标志。0代表正常显示，1代表反白显示。
* 第5，6个参数x_amount，y_amount分别代表字模数组的横向有多少个字节，纵向有几横。
* 第7个参数uiOffSetAddr是偏移地址，代表字模数组的从第几个数据开始显示。
*/
void display_lattice(unsigned int x,unsigned int y,const unsigned char  *ucArray,unsigned char
ucFbFlag,unsigned int x_amount,unsigned int y_amount,unsigned int uiOffSetAddr)
{
   unsigned int j=0;
   unsigned int i=0;
   unsigned char ucTemp;
//注意，要把以下两行指令屏蔽，否则屏幕在更新显示时会整屏闪动
//  WriteCommand(0x34);  //关显示缓冲指令
//  WriteCommand(0x34);  //关显示缓冲指令  故意写2次，怕1次关不了 这个是因为我参考到某厂家的驱动程序也是
这样写的
   for(j=0;j<y_amount;j++) //y_amount代表y轴有多少横
   {
       WriteCommand(y+j+0x80);         //垂直地址
       WriteCommand(x+0x80);           //水平地址
       for(i=0;i<x_amount;i++) //x_amount代表x轴有多少列
       {
           ucTemp=ucArray[j*x_amount+i+uiOffSetAddr]; //uiOffSetAddr是字模数组的偏移地址
           if(ucFbFlag==1)  //反白显示
           {
               ucTemp=~ucTemp;
           }
           LCDWriteData(ucTemp);
          //         delay_short(30000);  //把上一节这个延时函数去掉，加快刷屏速度
      }
   }
   WriteCommand(0x36); //开显示缓冲指令
}
void SendByteToLcd(unsigned char ucData)  //发送一个字节数据到液晶模块
{
        unsigned char i;
        for ( i = 0; i < 8; i++ )
        {
                if ( (ucData << i) & 0x80 )
                {
                        LCDSID_dr = 1;
                }
                else
                {
                        LCDSID_dr = 0;
```

```
                }
                LCDCLK_dr = 0;
                LCDCLK_dr = 1;
        }
}
void SPIWrite(unsigned char ucWData, unsigned char ucWRS) //模拟SPI发送一个字节的命令或者数据给液晶模块的底层驱动
{
        SendByteToLcd( 0xf8 + (ucWRS << 1) );
        SendByteToLcd( ucWData & 0xf0 );
        SendByteToLcd( (ucWData << 4) & 0xf0);
}
void WriteCommand(unsigned char ucCommand) //发送一个字节的命令给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucCommand, 0);
        delay_short(90);
}
void LCDWriteData(unsigned char ucData)  //发送一个字节的数据给液晶模块
{
        LCDCS_dr = 0;
        LCDCS_dr = 1;
        SPIWrite(ucData, 1);
}
void LCDInit(void) //初始化  函数内部包括液晶模块的复位
{
        LCDRST_dr = 1;  //复位
        LCDRST_dr = 0;
        LCDRST_dr = 1;
}
void delay_short(unsigned int uiDelayShort) //延时函数
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
```

}

总结陈词:

液晶屏显示的内容到这一节为止基本讲完。前面第38节到第45节是讲串口的，我的串口程序大部分都是通过靠时间来识别每一串数据是否接收完毕，只要第41节内容不是靠时间来判断，而是根据特定关键字来快速识别数据串是否接收完毕，下一节我打算结合我最新的一个项目经验，继续讲一个这方面的例子。欲知详情，请听下回分解----当主机连续不断地发送一串串数据给从机时，从机串口如何快速截取有效数据串。

（未完待续，下节更精彩，不要走开哦）

第八十七节：郑文显捐赠的工控项目源代码。

开场白:

根据上一节的预告，本来这节要讲关于串口的一个小项目，但是今天中午的时候，有个厦门客户的出现，让我决定先插入这节内容。

他叫郑文显，是做PLC开发的。今天中午他要我帮他写一个工控程序让他来学习，也是基于朱兆祺51单片机学习板的，他想把这个源代码经过自己修改后移植到他自己做的工控板上。我一开始报价4000元，被他砍价到1000元，我看一下也不算很难就答应了下来。刚才下午花了3个小时终于做好了。郑文显爽快的付了款，并且在电话那里跟我讲，他说独乐乐不如众乐乐，资源只有分享才能发挥它的最大价值，因此他决定要把这个源代码捐赠出来给大家一起学。非常感谢他的慈善壮举。种善因，得善果。好人一生平安。他的这个项目不难，跟我第25节内容很类似，略加修改就可以了。具体功能需求请看以下第（2）点。

（1）　　硬件平台:

基于朱兆祺51单片机学习板。

（2）　　实现功能:

他的系统要控制2个气缸，没有任何传感器。第1个气缸先伸出去，1秒钟后再收回来。然后第2个气缸再伸出去，1秒钟后再收回来，算完成一个过程，然后重头开始循环下去。每一个过程要计数加1显示在右边的4位数码管上，左边的4位数码管显示设定的最大计数上限，一旦超过这个计数上限就自动停止。有4个按键，一个按键用来启动，一个按键用来急停。另外两个按键是加减按键，用来设置左边显示的最大计数上限。断电要求数据不丢失。如果同时按下加减两个按键，可以清零当前计数的内容。

这4个按键都是独立按键。S1键是加键，S5键是减键，S9键是启动键，S13键是急停键。其中74HC595驱动丝印为D1的LED灯模拟第1个气缸，丝印为D2的LED灯模拟第2个气缸。

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1    20    //按键去抖动延时的时间
#define const_key_time2    20    //按键去抖动延时的时间
#define const_key_time3    20    //按键去抖动延时的时间
#define const_key_time4    20    //按键去抖动延时的时间
#define const_key_time12   20   //按键去抖动延时的时间
#define const_1s  500  //1秒钟大概的定时中断次数
void start24(void);  //开始位
void ack24(void);  //确认位
void stop24(void);  //停止位
unsigned char read24(void);  //读取一个字节的时序
void write24(unsigned char dd); //发送一个字节的时序
unsigned char read_eeprom(unsigned int address);   //从一个地址读取出一个字节数据
void write_eeprom(unsigned int address,unsigned char dd); //往一个地址存入一个字节数据
unsigned int read_eeprom_int(unsigned int address);   //从一个地址读取出一个int类型的数据
void write_eeprom_int(unsigned int address,unsigned int uiWriteData); //往一个地址存入一个int类型的数据
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
```

```
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time();  //定时中断函数
void key_service(); //按键服务的应用程序
void key_scan();//按键扫描函数 放在定时中断里
void run(); //设备自动控制程序
void left_to_right();  //从左边移动到右边
void right_to_left(); //从右边返回到左边
void up_to_down();   //从上边移动到下边
void down_to_up();    //从下边返回到上边
void led_update();  //LED更新函数
void delay_timer(unsigned int uiDelayTimerTemp);
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //作为中途暂停指示灯 亮的时候表示中途暂停
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit eeprom_scl_dr=P3^7;    //时钟线
sbit eeprom_sda_dr_sr=P3^6; //数据的输出线和输入线
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
unsigned char ucKeySec=0;   //被触发的按键编号
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
```

```
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Part1Update=1; //左边4位数码管更新显示标志
unsigned char ucWd1Part2Update=1; //右边4位数码管更新显示标志
unsigned int  uiSetData=18;  //需要被设置的计数上限
unsigned int  uiRunCnt=0; //实际运行的计数值
unsigned char ucRunTimeFlag=0; //延时计数器的开关
unsigned int  uiRunTimeCnt=0;  //运动中的时间延时计数器变量
unsigned char ucRunStep=1;  //运动控制的步骤变量
unsigned char ucRunFlag=0;  //是否启动运行的标志   1代表运行
unsigned char ucDelayTimerFlag=0; //计时器的开关
unsigned int  uiDelayTimer=0;
unsigned char ucLed_dr1=0;   //代表16个灯的亮灭状态，0代表灭，1代表亮
unsigned char ucLed_dr2=0;
unsigned char ucLed_dr3=0;
unsigned char ucLed_dr4=0;
unsigned char ucLed_dr5=0;
unsigned char ucLed_dr6=0;
unsigned char ucLed_dr7=0;
unsigned char ucLed_dr8=0;
unsigned char ucLed_dr9=0;
unsigned char ucLed_dr10=0;
unsigned char ucLed_dr11=0;
unsigned char ucLed_dr12=0;
unsigned char ucLed_dr13=0;
unsigned char ucLed_dr14=0;
unsigned char ucLed_dr15=0;
unsigned char ucLed_dr16=0;
unsigned char ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
```

```
   while(1)
   {
      key_service(); //按键服务的应用程序
          run(); //设备自动控制程序
      display_service(); //显示的窗口菜单服务程序
      led_update();  //LED更新函数
   }
}
void left_to_right()  //从左边移动到右边
{
   ucLed_dr1=1;   // 1代表左右气缸从左边移动到右边
   ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void right_to_left() //从右边返回到左边
{
   ucLed_dr1=0;   // 0代表左右气缸从右边返回到左边
   ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void up_to_down()   //从上边移动到下边
{
   ucLed_dr2=1;   // 1代表上下气缸从上边移动到下边
   ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void down_to_up()    //从下边返回到上边
{
   ucLed_dr2=0;   // 0代表上下气缸从下边返回到上边
   ucLed_update=1;  //刷新变量。每次更改LED灯的状态都要更新一次。
}
void run() //设备自动控制程序
{
   if(ucRunFlag==1)  //是否启动运行的标志
   {
       switch(ucRunStep)
       {
          case 1:    //机械手从左边往右边移动
               left_to_right();
                          ucRunTimeFlag=0; //延时计数器关  在清零uiRunTimeCnt变量前，最好先关闭计时器开
关，起到跟中断互锁作用
                            uiRunTimeCnt=0;  //时间计数器清零，为接下来延时1秒钟做准备
               ucRunTimeFlag=1; //延时计数器开    感谢郑文显捐助本节源代码
               ucRunStep=2;  //这就是鸿哥传说中的怎样灵活控制步骤变量
               break;
          case 2:    //延时1秒
               if(uiRunTimeCnt>const_1s)  //延时1秒
               {
                  ucRunStep=3;  //这就是鸿哥传说中的怎样灵活控制步骤变量
               }
               break;
          case 3:    //机械手从右边往左边移动
               right_to_left();
```

```
                    ucRunTimeFlag=0; //延时计数器关  在清零uiRunTimeCnt变量前，最好先关闭计时器开关，起到跟中断互锁作用
                      uiRunTimeCnt=0;  //时间计数器清零，为接下来延时1秒钟做准备
            ucRunTimeFlag=1; //延时计数器开
            ucRunStep=4;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
        case 4:    //延时1秒
            if(uiRunTimeCnt>const_1s)  //延时1秒
            {
               ucRunStep=5;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
        case 5:    //机械手//从上边移动到下边
            up_to_down();
                    ucRunTimeFlag=0; //延时计数器关  在清零uiRunTimeCnt变量前，最好先关闭计时器开关，起到跟中断互锁作用
                      uiRunTimeCnt=0;  //时间计数器清零，为接下来延时1秒钟做准备
            ucRunTimeFlag=1; //延时计数器开
            ucRunStep=6;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
        case 6:    //延时1秒
            if(uiRunTimeCnt>const_1s)  //延时1秒
            {
               ucRunStep=7;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            }
            break;
        case 7:    //机械手从下边返回到上边
            down_to_up();
                    ucRunTimeFlag=0; //延时计数器关  在清零uiRunTimeCnt变量前，最好先关闭计时器开关，起到跟中断互锁作用
                      uiRunTimeCnt=0;  //时间计数器清零，为接下来延时1秒钟做准备
            ucRunTimeFlag=1; //延时计数器开  感谢郑文显捐助本节源代码
            ucRunStep=8;  //这就是鸿哥传说中的怎样灵活控制步骤变量
            break;
        case 8:    //延时1秒
            if(uiRunTimeCnt>const_1s)  //延时1秒
            {
               uiRunCnt++; //实际运行的计数值累加
                        if(uiRunCnt>9999)  //数码管最大显示4位9999，如果超过了，继续默认为9999
               {
                            uiRunCnt=9999;
                        }
               ucWd1Part2Update=1;  //右边4位数码管更新显示
               write_eeprom_int(2,uiRunCnt); //及时把数据存进EEPROM，避免掉电丢失数据
                        if(uiRunCnt>=uiSetData) //如果实际的计数大于或者等于设定上限，则停止
                        {
                            ucRunFlag=0;  //停止
                                ucRunStep=1;  //切换到第一步为下一次准备
                        }
                        else
```

```
                                {
                        ucRunStep=1;  //切换到第一步继续运行
                                }
                }
                break;
        }
    }
}
void display_service() //显示的窗口菜单服务程序
{
  //加了static关键字后，此局部变量不会每次进来函数都初始化一次，这样减少了一点指令消耗的时间。
            static unsigned char ucTemp4;   //中间过渡变量
            static unsigned char ucTemp3;   //中间过渡变量
            static unsigned char ucTemp2;   //中间过渡变量
            static unsigned char ucTemp1;   //中间过渡变量
            //左边4位数码管显示设置的计数上限
            if(ucWd1Part1Update==1)  //左边4位数码管要全部更新显示
            {
               ucWd1Part1Update=0;  //及时清零标志，避免一直进来扫描
              //先分解数据用来显示每一位
               ucTemp4=uiSetData/1000;
               ucTemp3=uiSetData%1000/100;
               ucTemp2=uiSetData%100/10;
               ucTemp1=uiSetData%10;

               if(uiSetData<1000)
               {
                  ucDigShow8=10;  //如果小于1000，千位显示无
               }
               else
               {
                  ucDigShow8=ucTemp4;  //第8位数码管要显示的内容
               }
               if(uiSetData<100)
               {
                  ucDigShow7=10;  //如果小于100，百位显示无
               }
               else
               {
                  ucDigShow7=ucTemp3;  //第7位数码管要显示的内容
               }
               if(uiSetData<10)
               {
                  ucDigShow6=10;  //如果小于10，十位显示无
               }
               else
               {
                  ucDigShow6=ucTemp2;  //第6位数码管要显示的内容
               }
                ucDigShow5=ucTemp1;  //第5位数码管要显示的内容
```

```
            }
            //右边4位数码管显示实际的计数
            if(ucWd1Part2Update==1)  //右边4位数码管要全部更新显示
            {
               ucWd1Part2Update=0;  //及时清零标志，避免一直进来扫描
              //先分解数据用来显示每一位
               ucTemp4=uiRunCnt/1000;
               ucTemp3=uiRunCnt%1000/100;
               ucTemp2=uiRunCnt%100/10;
               ucTemp1=uiRunCnt%10;

               if(uiRunCnt<1000)
               {
                  ucDigShow4=10;  //如果小于1000，千位显示无
               }
               else
               {
                  ucDigShow4=ucTemp4;  //第8位数码管要显示的内容
               }
               if(uiRunCnt<100)
               {
                  ucDigShow3=10;  //如果小于100，百位显示无
               }
               else
               {
                  ucDigShow3=ucTemp3;  //第7位数码管要显示的内容
               }
               if(uiRunCnt<10)
               {
                  ucDigShow2=10;  //如果小于10，十位显示无
               }
               else
               {
                  ucDigShow2=ucTemp2;  //第6位数码管要显示的内容
               }
                ucDigShow1=ucTemp1;  //第5位数码管要显示的内容
            }
}
void key_scan()//按键扫描函数 放在定时中断里
{
  //加了static关键字后，此局部变量不会每次进来函数都被初始化一次，这样可以记录保存上一次执行本函数后的数值
static unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
static unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
static unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
static unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
static unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
static unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
static unsigned int  uiKeyTimeCnt4=0; //按键去抖动延时计数器
static unsigned char ucKeyLock4=0; //按键触发后自锁的变量标志
```

```
static unsigned int  uiKeyTimeCnt12=0; //按键去抖动延时计数器
static unsigned char ucKeyLock12=0; //按键触发后自锁的变量标志
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //触发3号键
     }
  }
  if(key_sr4==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock4=0; //按键自锁标志清零
```

```
    uiKeyTimeCnt4=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock4==0)//有按键按下，且是第一次被按下
  {
    uiKeyTimeCnt4++; //累加定时中断次数
    if(uiKeyTimeCnt4>const_key_time4)
    {
      uiKeyTimeCnt4=0;
      ucKeyLock4=1;  //自锁按键置位,避免一直触发
      ucKeySec=4;    //触发4号键
    }
  }
//5号组合键
if(key_sr1==1||key_sr2==1)//IO是高电平，说明两个按键没有全部被按下，这时要及时清零一些标志位
  {
       ucKeyLock12=0; //按键自锁标志清零
       uiKeyTimeCnt12=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock12==0)//有按键按下，且是第一次被按下
  {
    uiKeyTimeCnt12++; //累加定时中断次数
    if(uiKeyTimeCnt12>const_key_time12)
    {
      uiKeyTimeCnt12=0;
      ucKeyLock12=1;  //自锁按键置位,避免一直触发
      ucKeySec=5;    //触发5号组合键

    }
  }
}
void key_service() //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          if(ucRunFlag==0)  //如果系统还没运行
                  {
             uiSetData++;    //被设置的计数上限
             if(uiSetData>9999) //最大值是9999
             {
                uiSetData=9999;
             }
             ucWd1Part1Update=1;  //左边4位数码管更新显示

                  write_eeprom_int(0,uiSetData); //及时保存数据进EEPROM，避免掉电丢失
             uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                  }
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
```

```
    case 2:// 减按键 对应朱兆祺学习板的S5键
          if(ucRunFlag==0)  //如果系统还没运行
                  {
             uiSetData--;
             if(uiSetData>9999)  //unsigned int 类型的0减去1会变成65535(0xffff)
             {
                uiSetData=0;  //最小值是0
             }
             ucWd1Part1Update=1;  //左边4位数码管更新显示
             write_eeprom_int(0,uiSetData); //及时保存数据进EEPROM，避免掉电丢失
             uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                  }
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 3://启动按键 对应朱兆祺学习板的S9键
          if(ucRunFlag==0&&uiRunCnt<uiSetData)  //如果系统还没运行，并且实际运行的次数小于设定的最大次数
，则启动
                  {
                      ucRunFlag=1;
              ucRunStep=1;
              uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                  }
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
    case 4://急停按键 对应朱兆祺学习板的S9键
          ucRunFlag=0; //急停
          ucRunStep=1;
          right_to_left(); //从右边返回到左边
          down_to_up();    //从下边返回到上边
          uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;

    case 5://清零的组合按键 对应朱兆祺学习板的(S1+S5)组合键
          if(ucRunFlag==0)  //如果系统还没运行
                  {
             uiRunCnt=0; //实际计数清零
             ucWd1Part2Update=1;  //右边4位数码管更新显示
             write_eeprom_int(2,uiRunCnt); //存入uiRunCnt，内部占用2个字节地址
             uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
                  }
          ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
          break;
  }
}
void led_update()  //LED更新函数
{
  //加了static关键字后，此局部变量不会每次进来函数都被初始化一次，这样可以记录保存上一次执行本函数后的数
值
static unsigned char ucLedStatus16_09=0;    //代表底层74HC595输出状态的中间变量
```

```
static unsigned char ucLedStatus08_01=0;    //代表底层74HC595输出状态的中间变量
   if(ucLed_update==1)
   {
       ucLed_update=0;    //及时清零，让它产生只更新一次的效果，避免一直更新。
       if(ucLed_dr1==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x01;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfe;
           }
       if(ucLed_dr2==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x02;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfd;
           }
       if(ucLed_dr3==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x04;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xfb;
           }
       if(ucLed_dr4==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x08;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xf7;
           }
       if(ucLed_dr5==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x10;
           }
           else
           {
              ucLedStatus08_01=ucLedStatus08_01&0xef;
           }
       if(ucLed_dr6==1)
           {
              ucLedStatus08_01=ucLedStatus08_01|0x20;
           }
           else
           {
```

```
        ucLedStatus08_01=ucLedStatus08_01&0xdf;
    }
if(ucLed_dr7==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x40;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0xbf;
    }
if(ucLed_dr8==1)
    {
        ucLedStatus08_01=ucLedStatus08_01|0x80;
    }
    else
    {
        ucLedStatus08_01=ucLedStatus08_01&0x7f;
    }
if(ucLed_dr9==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x01;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xfe;
    }
if(ucLed_dr10==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x02;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xfd;
    }
if(ucLed_dr11==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x04;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xfb;
    }
if(ucLed_dr12==1)
    {
        ucLedStatus16_09=ucLedStatus16_09|0x08;
    }
    else
    {
        ucLedStatus16_09=ucLedStatus16_09&0xf7;
    }
```

```
        if(ucLed_dr13==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x10;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xef;
            }
        if(ucLed_dr14==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x20;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xdf;
            }
        if(ucLed_dr15==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x40;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0xbf;
            }
        if(ucLed_dr16==1)
            {
               ucLedStatus16_09=ucLedStatus16_09|0x80;
            }
            else
            {
               ucLedStatus16_09=ucLedStatus16_09&0x7f;
            }
        hc595_drive(ucLedStatus16_09,ucLedStatus08_01);  //74HC595底层驱动函数
    }
}
void display_drive()
{
    //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
    switch(ucDisplayDriveStep)
    {
       case 1:  //显示第1位
            ucDigShowTemp=dig_table[ucDigShow1];
                    if(ucDigDot1==1)
                    {
                       ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0xfe);
                break;
       case 2:  //显示第2位
            ucDigShowTemp=dig_table[ucDigShow2];
```

```
            if(ucDigDot2==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xfd);
         break;
case 3:  //显示第3位
     ucDigShowTemp=dig_table[ucDigShow3];
            if(ucDigDot3==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xfb);
         break;
case 4:  //显示第4位
     ucDigShowTemp=dig_table[ucDigShow4];
            if(ucDigDot4==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xf7);
         break;
case 5:  //显示第5位
     ucDigShowTemp=dig_table[ucDigShow5];
            if(ucDigDot5==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xef);
         break;
case 6:  //显示第6位
     ucDigShowTemp=dig_table[ucDigShow6];
            if(ucDigDot6==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
     dig_hc595_drive(ucDigShowTemp,0xdf);
         break;
case 7:  //显示第7位
     ucDigShowTemp=dig_table[ucDigShow7];
            if(ucDigDot7==1)
            {
               ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
     }
     dig_hc595_drive(ucDigShowTemp,0xbf);
         break;
case 8:  //显示第8位
     ucDigShowTemp=dig_table[ucDigShow8];
            if(ucDigDot8==1)
            {
```

```
                        ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                    }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;     //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
```

```
{
   unsigned char i;
   unsigned char ucTempData;
   hc595_sh_dr=0;
   hc595_st_dr=0;
   ucTempData=ucLedStatusTemp16_09;  //先送高8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   ucTempData=ucLedStatusTemp08_01;  //再先送低8位
   for(i=0;i<8;i++)
   {
         if(ucTempData>=0x80)hc595_ds_dr=1;
         else hc595_ds_dr=0;
         hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
         delay_short(1);
         hc595_sh_dr=1;
         delay_short(1);
         ucTempData=ucTempData<<1;
   }
   hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   hc595_st_dr=1;
   delay_short(1);
   hc595_sh_dr=0;    //拉低，抗干扰就增强
   hc595_st_dr=0;
   hc595_ds_dr=0;
}
//AT24C02驱动程序
void start24(void)  //开始位
{
    eeprom_sda_dr_sr=1;
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_sda_dr_sr=0;
        delay_short(15);
    eeprom_scl_dr=0;
}
void ack24(void)  //确认位时序
{
    eeprom_sda_dr_sr=1; //51单片机在读取数据之前要先置一，表示数据输入
    eeprom_scl_dr=1;
        delay_short(15);
```

```
    eeprom_scl_dr=0;
        delay_short(15);
//在本驱动程序中，我没有对ACK信号进行出错判断，因为我这么多年一直都是这样用也没出现过什么问题。
//有兴趣的朋友可以自己增加出错判断，不一定非要按我的方式去做。
}
void stop24(void)  //停止位
{
    eeprom_sda_dr_sr=0;
    eeprom_scl_dr=1;
        delay_short(15);
    eeprom_sda_dr_sr=1;
}
unsigned char read24(void)  //读取一个字节的时序
{
        unsigned char outdata,tempdata;
        outdata=0;
                eeprom_sda_dr_sr=1; //51单片机的IO口在读取数据之前要先置一，表示数据输入
        delay_short(2);
        for(tempdata=0;tempdata<8;tempdata++)
        {
            eeprom_scl_dr=0;
            delay_short(2);
            eeprom_scl_dr=1;
            delay_short(2);
            outdata<<=1;
            if(eeprom_sda_dr_sr==1)outdata++;
            eeprom_sda_dr_sr=1; //51单片机的IO口在读取数据之前要先置一，表示数据输入
            delay_short(2);
        }
    return(outdata);

}
void write24(unsigned char dd) //发送一个字节的时序
{
        unsigned char tempdata;
        for(tempdata=0;tempdata<8;tempdata++)
        {
                if(dd>=0x80)eeprom_sda_dr_sr=1;
                else eeprom_sda_dr_sr=0;
                dd<<=1;
                delay_short(2);
                eeprom_scl_dr=1;
                delay_short(4);
                eeprom_scl_dr=0;
        }
}
unsigned char read_eeprom(unsigned int address)   //从一个地址读取出一个字节数据
{
   unsigned char dd,cAddress;
   cAddress=address; //把低字节地址传递给一个字节变量。
```

```
    EA=0; //禁止中断
    start24(); //IIC通讯开始
    write24(0xA0); //此字节包含读写指令和芯片地址两方面的内容。
                   //指令为写指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
    ack24(); //发送应答信号
    write24(cAddress); //发送读取的存储地址(范围是0至255)
    ack24(); //发送应答信号
    start24(); //开始
    write24(0xA1); //此字节包含读写指令和芯片地址两方面的内容。
                   //指令为读指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
    ack24(); //发送应答信号
    dd=read24(); //读取一个字节
    ack24(); //发送应答信号
    stop24();  //停止
    EA=1; //允许中断
    delay_timer(2); //一气呵成的定时器延时方式，在延时的时候还可以动态扫描数码管
    return(dd);
}
void write_eeprom(unsigned int address,unsigned char dd) //往一个地址存入一个字节数据
{
    unsigned char cAddress;
    cAddress=address; //把低字节地址传递给一个字节变量。
    EA=0; //禁止中断
    start24(); //IIC通讯开始
    write24(0xA0); //此字节包含读写指令和芯片地址两方面的内容。
                   //指令为写指令。地址为"000"的信息，此信息由A0,A1,A2的引脚决定
    ack24(); //发送应答信号
    write24(cAddress);   //发送写入的存储地址(范围是0至255)
    ack24(); //发送应答信号
    write24(dd);  //写入存储的数据
    ack24(); //发送应答信号
    stop24();  //停止
    EA=1; //允许中断
    delay_timer(4); //一气呵成的定时器延时方式，在延时的时候还可以动态扫描数码管
}
unsigned int read_eeprom_int(unsigned int address)   //从一个地址读取出一个int类型的数据
{
    unsigned char ucReadDataH;
    unsigned char ucReadDataL;
    unsigned int  uiReadDate;
    ucReadDataH=read_eeprom(address);    //读取高字节
    ucReadDataL=read_eeprom(address+1);  //读取低字节
    uiReadDate=ucReadDataH;  //把两个字节合并成一个int类型数据
    uiReadDate=uiReadDate<<8;
    uiReadDate=uiReadDate+ucReadDataL;
    return uiReadDate;
}
void write_eeprom_int(unsigned int address,unsigned int uiWriteData) //往一个地址存入一个int类型的数据
{
    unsigned char ucWriteDataH;
```

```
   unsigned char ucWriteDataL;
   ucWriteDataH=uiWriteData>>8;
   ucWriteDataL=uiWriteData;
   write_eeprom(address,ucWriteDataH); //存入高字节
   write_eeprom(address+1,ucWriteDataL); //存入低字节
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  if(ucRunTimeFlag==1) //void run函数中的延时计数器开关
  {
     uiRunTimeCnt++; //延时计数器
  }
  if(ucDelayTimerFlag==1)//delay_timer函数中的延时计数器开关
  {
     if(uiDelayTimer>0)
         {
           uiDelayTimer--;   //一气呵成的定时器延时方式的计时器
         }

  }
  key_scan(); //按键扫描函数
  if(uiVoiceCnt!=0)
  {
     uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
     beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
//     beep_dr=1;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。
  }
  else
  {
     ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
     beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
//     beep_dr=0;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。
  }
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_timer(unsigned int uiDelayTimerTemp)
{
    ucDelayTimerFlag=0; //延时计时器关  在设置参数前，先关闭计时器
    uiDelayTimer=uiDelayTimerTemp;
    ucDelayTimerFlag=1; //延时计时器开
    while(uiDelayTimer!=0);  //一气呵成的定时器方式延时等待
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
```

```
    for(i=0;i<uiDelayShort;i++)
    {
      ;   //一个分号相当于执行一条空语句
    }
}
void delay_long(unsigned int uiDelayLong)
{
    unsigned int i;
    unsigned int j;
    for(i=0;i<uiDelayLong;i++)
    {
       for(j=0;j<500;j++)  //内嵌循环的空指令数量
           {
              ; //一个分号相当于执行一条空语句
           }
    }
}
void initial_myself()  //第一区 初始化单片机
{
/* 注释一:
* 矩阵键盘也可以做独立按键，前提是把某一根公共输出线输出低电平，
* 模拟独立按键的触发地，本程序中，把key_gnd_dr输出低电平。
* 朱兆祺51学习板的S1就是本程序中用到的一个独立按键。
*/
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  led_dr=0;  //关闭独立LED灯 感谢郑文显捐助本节源代码
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral() //第二区 初始化外围
{
    ucDigDot8=0;   //小数点全部不显示
    ucDigDot7=0;
    ucDigDot6=0;
    ucDigDot5=0;
    ucDigDot4=0;
    ucDigDot3=0;
    ucDigDot2=0;
    ucDigDot1=0;
    EA=1;     //开总中断
    ET0=1;    //允许定时中断
    TR0=1;    //启动定时中断
    uiSetData=read_eeprom_int(0);  //读取uiSetData，内部占用2个字节地址
    if(uiSetData>9999)   //不在范围内
    {
        uiSetData=0;   //填入一个初始化数据
        write_eeprom_int(0,uiSetData); //存入uiSetData，内部占用2个字节地址
```

```
    }
    uiRunCnt=read_eeprom_int(2);  //读取uiRunCnt，内部占用2个字节地址
    if(uiRunCnt>9999)//不在范围内
    {
        uiRunCnt=0;  //填入一个初始化数据
        write_eeprom_int(2,uiRunCnt); //存入uiRunCnt，内部占用2个字节地址
    }
}
```

总结陈词:

再次感谢郑文显的无私奉献。前面第38节到第45节是讲串口的，我的串口程序大部分都是通过靠时间来识别每一串数据是否接收完毕，只要第41节内容不是靠时间来判断，而是根据特定关键字来快速识别数据串是否接收完毕，下一节我打算结合我最新的一个项目经验，继续讲一个这方面的例子。欲知详情，请听下回分解----当主机连续不断地发送一串串数据给从机时，从机串口如何快速截取有效数据串。

（未完待续，下节更精彩，不要走开哦）

第八十八节：电子称连续不断从串口对外发送数据，单片机靠关键字快速截取有效数据串。

开场白:

我前面串口程序大部分都是通过靠时间来识别每一串数据是否接收完毕，有一些串口项目的协议是固定不变的，而且也不需要从机反馈任何应答信号，这类项目只需根据特定关键字来快速识别数据串是否接收完毕即可。比如现在有一种电子称，它的测量范围是0.00克到500.00克，他是靠串口不断对外发送当前重量数据的，每串数据固定长度26个字节，最后两个字节是回车换行符0x0d 0x0a，倒数第9,10,11,12,13,14为有效的ASCII码数字，其中倒数第11位为固定的小数点，其它的数据可以忽略不计。这类串口框架的思路是：根据数据尾是否有0x0d 0x0a来判断数据串是否有效的，一旦发现有此关键字，再判断总的数据长度是否等于或者大于一串数据的固定长度，如果满足，则把相关标志位置位，通知主函数中的串口服务程序进行处理。同时也及时关闭串口中断，避免在处理串口数据期间受到串口数据的中断干扰，等串口服务程序处理完毕再打开。

具体内容，请看源代码讲解。

（1）　　硬件平台:

基于朱兆祺51单片机学习板。

（2）　　实现功能:

波特率是：9600。把当前电子称的重量数据显示在数码管上，在电脑上用串口助手软件来模拟电子称发送以下格式协议的3串数据，它的协议很简单，每串数据固定长度26个字节，最后两个字节是回车换行符0x0d 0x0a，倒数第9,10,11,12,13,14为有效的ASCII码数字，其中倒数第11位为固定的小数点，其它的数据可以忽略不计。

（a）字符是:

```
ST,GS,+       0.77     g
```

转换成16进制是:

```
20 53 54 2C 47 53 2C 2B 20 20 20 20 20 20 30 2E 37 37 20 2020 20 20 67 0D 0A
```

数码管显示：0.77

(b)

字符是:

```
ST,GS,+     136.39     g
```

转换成16进制是:

```
20 53 54 2C 47 53 2C 2B 20 20 20 20 31 33 36 2E 33 39 20 2020 20 20 67 0D 0A
```

数码管显示：136.39

(c)

字符是:

```
ST,GS,+       0.00     g
```

转换成16进制是:

```
20 53 54 2C 47 53 2C 2B 20 20 20 20 20 20 30 2E 30 30 20 2020 20 20 67 0D 0A
```

数码管显示：0.00

（3）源代码讲解如下:

```
#include "REG52.H"
#define const_rc_size  36  //接收串口中断数据的缓冲区数组大小
#define const_least_size 26   //一串标准数据的大小
void initial_myself();
void initial_peripheral();
void delay_short(unsigned int uiDelayShort);
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(); //显示数码管字模的驱动函数
void display_service(); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void usart_service(void);  //串口接收服务程序,在main函数里
void usart_receive(void); //串口接收中断函数
void T0_time();  //定时中断函数
sbit dig_hc595_sh_dr=P2^0;     //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;    //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit led_dr=P3^5;  //独立LED灯
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
unsigned int  uiRcregTotal=0;  //代表当前缓冲区已经接收了多少个数据
unsigned int  uiRcregTotalTemp=0;  //代表当前缓冲区已经接收了多少个数据的中间变量
unsigned char ucRcregBuf[const_rc_size]; //接收串口中断数据的缓冲区数组
unsigned char ucReceiveFlag=0; //接收成功标志
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
```

```
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd1Part1Update=1; //8位数码管更新显示标志
unsigned long ulWeightCurrent=12345; //显示当前实际的重量
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      usart_service();  //串口接收服务程序
      display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释一:
* 本节内容处理串口数据是根据数据尾是否有0x0d 0x0a来判断数据串是否有效的，一旦发现有此关键字，
* 再判断总的数据长度是否等于或者大于一串数据的固定长度，如果满足，则把相关标志位置位，通知主函数中
* 的串口服务程序进行处理。同时也及时关闭串口中断，避免在处理串口数据期间受到串口数据的中断干扰，
* 等串口服务程序处理完毕再打开。
*/
void usart_receive(void) interrupt 4   //串口接收数据中断函数
{
   if(RI==1)
   {
        RI = 0;
        ++uiRcregTotal;
        ucRcregBuf[uiRcregTotal-1]=SBUF;    //将串口接收到的数据缓存到接收缓冲区里
        if(uiRcregTotal>=2&&ucRcregBuf[uiRcregTotal-2]==0x0d&&ucRcregBuf[uiRcregTotal-1]==0x0a)  //一旦
发现后缀是0x0d 0x0a关键字的就进去处理判断
        {
           if(uiRcregTotal<const_least_size)  //如果发现总的接收数据小于一串数据的固定长度。则无效，清零
重新开始。
           {
               uiRcregTotal=0;
           }
           else
           {
```

```
                uiRcregTotalTemp=uiRcregTotal; //把接收到的总数据传递给一个中间变量，在主函数那边处理这个中间变量
                ucReceiveFlag=1;              //通知主程序接收成功
                            ES=0;       // 禁止接收中断,等主函数处理完接收的数据后再打开串口中断，避免在处理串口数据期间受到串口数据的中断干扰。
              }
           }
           else if(uiRcregTotal>=const_rc_size)  //超过缓冲区
           {
              uiRcregTotal=0;
           }


     }
     else    //如果不是串口接收中断，那么必然是串口发送中断，及时清除发送中断的标志，否则一直发送中断
     {
          TI = 0;
     }

}
void usart_service(void)  //串口接收服务程序,在main函数里
{
  //加了static关键字后，此局部变量不会每次进来函数都初始化一次，这样有可能减少了一点指令消耗的时间。
    static unsigned long ulReceiveData10000; //定义成long类型，是为了方便后面换算的乘法运算，让它不会溢出而出错。
    static unsigned long ulReceiveData1000;
    static unsigned long ulReceiveData100;
    static unsigned long ulReceiveData10;
    static unsigned long ulReceiveData1;
    if(ucReceiveFlag==1)  //说明有数据接收成功，进入数据处理分析
    {
       ulReceiveData10000=0;
       ulReceiveData1000=0;
       ulReceiveData100=0;
       ulReceiveData10=0;
       ulReceiveData1=0;
/* 注释二:
* 根据协议，倒数第9,10,11,12,13,14为有效的ASCII码数字，其中倒数第11位为固定的小数点，因此省略不写。
*/
       if(ucRcregBuf[uiRcregTotalTemp-9]>=0x30)
       {
          ulReceiveData1=ucRcregBuf[uiRcregTotalTemp-9]-0x30; //接收到的ASCII码数字减去0x30变成实际数值.
       }
       if(ucRcregBuf[uiRcregTotalTemp-10]>=0x30)
       {
          ulReceiveData10=ucRcregBuf[uiRcregTotalTemp-10]-0x30;
          ulReceiveData10=ulReceiveData10*10;
       }
       if(ucRcregBuf[uiRcregTotalTemp-12]>=0x30)
       {
```

```
            ulReceiveData100=ucRcregBuf[uiRcregTotalTemp-12]-0x30;
            ulReceiveData100=ulReceiveData100*100;
        }
        if(ucRcregBuf[uiRcregTotalTemp-13]>=0x30)
        {
            ulReceiveData1000=ucRcregBuf[uiRcregTotalTemp-13]-0x30;
            ulReceiveData1000=ulReceiveData1000*1000;
        }
        if(ucRcregBuf[uiRcregTotalTemp-14]>=0x30)
        {
            ulReceiveData10000=ucRcregBuf[uiRcregTotalTemp-14]-0x30;
            ulReceiveData10000=ulReceiveData10000*10000;
        }
ulWeightCurrent=ulReceiveData10000+ulReceiveData1000+ulReceiveData100+ulReceiveData10+ulReceiveData1;
        ucWd1Part1Update=1; //更新显示
        uiRcregTotalTemp=0;  //清零实际接收到的字节数的中间变量
        uiRcregTotal=0;  //清零实际接收到的字节数
        ucReceiveFlag=0;  //清零完成标志
            ES = 1;             // 允许接收中断
    }
}
void display_service() //显示的窗口菜单服务程序
{
  //加了static关键字后，此局部变量不会每次进来函数都初始化一次，这样有可能减少了一点指令消耗的时间。
            static unsigned char ucTemp5;   //中间过渡变量
            static unsigned char ucTemp4;   //中间过渡变量
            static unsigned char ucTemp3;   //中间过渡变量
            static unsigned char ucTemp2;   //中间过渡变量
            static unsigned char ucTemp1;   //中间过渡变量
            if(ucWd1Part1Update==1)  //更新显示
            {
               ucWd1Part1Update=0;  //及时清零标志，避免一直进来扫描
              //先分解数据用来显示每一位
               ucTemp5=ulWeightCurrent%100000/10000;
               ucTemp4=ulWeightCurrent%10000/1000;
               ucTemp3=ulWeightCurrent%1000/100;
               ucTemp2=ulWeightCurrent%100/10;
               ucTemp1=ulWeightCurrent%10;
               ucDigDot3=1;  //显示第3位数码管的小数点，实际数据带2位小数点。
               ucDigShow8=10;  //没有用到第8位数码管，因此显示无。10代表显示空。
               ucDigShow7=10;  //没有用到第7位数码管，因此显示无。10代表显示空。
               ucDigShow6=10;  //没有用到第6位数码管，因此显示无。10代表显示空。
               if(ulWeightCurrent<10000)
               {
                  ucDigShow5=10;  //如果小于1000，千位显示无
               }
               else
               {
                  ucDigShow5=ucTemp5;  //第5位数码管要显示的内容
               }
```

```
                if(ulWeightCurrent<1000)
                {
                   ucDigShow4=10;  //如果小于1000，千位显示无
                }
                else
                {
                   ucDigShow4=ucTemp4;  //第4位数码管要显示的内容
                }
                 //因为带2位小数点，因此最前面3位数据都是有效数，必然要显示，不要判断去0的空显示处理。
                 ucDigShow3=ucTemp3;  //第3位数码管要显示的内容
                 ucDigShow2=ucTemp2;  //第2位数码管要显示的内容
                 ucDigShow1=ucTemp1;  //第1位数码管要显示的内容
            }

}
void display_drive()
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
           ucDigShowTemp=dig_table[ucDigShow3];
                   if(ucDigDot3==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfb);
               break;
      case 4:  //显示第4位
           ucDigShowTemp=dig_table[ucDigShow4];
                   if(ucDigDot4==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
```

```
                }
            dig_hc595_drive(ucDigShowTemp,0xf7);
                break;
        case 5:  //显示第5位
            ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xef);
                break;
        case 6:  //显示第6位
            ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0xdf);
                break;
        case 7:  //显示第7位
            ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
            }
            dig_hc595_drive(ucDigShowTemp,0xbf);
                break;
        case 8:  //显示第8位
            ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
            dig_hc595_drive(ucDigShowTemp,0x7f);
                break;
    }
    ucDisplayDriveStep++;
    if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
```

```
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
```

```
        if(ucTempData>=0x80)hc595_ds_dr=1;
        else hc595_ds_dr=0;
        hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
        delay_short(1);
        hc595_sh_dr=1;
        delay_short(1);
        ucTempData=ucTempData<<1;
   }
   hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
   delay_short(1);
   hc595_st_dr=1;
   delay_short(1);
   hc595_sh_dr=0;    //拉低，抗干扰就增强
   hc595_st_dr=0;
   hc595_ds_dr=0;
}
void T0_time() interrupt 1
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself()  //第一区 初始化单片机
{
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  led_dr=0;  //关闭独立LED灯
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
```

```
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
  //配置串口
  SCON=0x50;
  TMOD=0X21;
/* 注释三:
* 为了保证串口中断接收的数据不丢失，必须设置IP = 0x10，相当于把串口中断设置为最高优先级，
* 这个时候，串口中断可以打断任何其他的中断服务函数实现嵌套，
*/
  IP =0x10;  //把串口中断设置为最高优先级，必须的。
  TH1=TL1=-(11059200L/12/32/9600);  //串口波特率为9600。
  TR1=1;
}
void initial_peripheral() //第二区 初始化外围
{
   ucDigDot8=0;   //初始化小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;      //开总中断
   ES=1;      //允许串口中断
   ET0=1;     //允许定时中断
   TR0=1;     //启动定时中断
}
```

总结陈词:

前面我在第48节里讲过用ds1302做的时钟程序，但是后来很多网友建议，为了方便初学者学习编程思路，我应该用单片机定时器做一个时钟程序。因此，我决定下一节讲这方面的内容。欲知详情，请听下回分解----用单片机内部定时器做一个时钟。

（未完待续，下节更精彩，不要走开哦）

第八十九节: 用单片机内部定时器做一个时钟。

开场白:

很多网友建议，为了方便初学者学习编程思路，我应该用单片机定时器做一个时钟程序供大家参考学习。其实我前面第48节就已经用ds1302做了一个可以显示和更高时间的时钟，这一节只要在第48节的源代码基础上，大的框架不用动，只需要把ds1302产生的时间改成用定时中断产生的时间就可以了，改动的地方非常小。但是为了让时间的精度更高，最后必须跟标准时间进行校验，来修正系统中一秒钟需要多个定时中断的误差，这个误差决定了系统的时间精度，其实这个校验方法我在前面很多章节上跟大家介绍过了:

第一步: 在程序代码上先写入1秒钟大概需要200个定时中断。

第二步: 把程序烧录进单片机后，上电开始测试，手上同步打开手机里的秒表，当手机的标准时间跑了780秒(这个标准时间跑得越长校验精度越高)，而此时单片机仅仅跑了1632秒。那么最终得出1秒钟需要的定时中断次数是:const_time_1s=(200*1632)/780=418。

第三步: 如果发现时钟还是不太准，可以继续返回第一步根据最新1秒钟的时间是418次，多校验几次，来不断调整const_time_1s的数值，直到找到相对精度的时间为止。

本系统仅供学习，精度不可能做得很好，因为影响时间精度的因素还有定时中断的重装值，定时中断里面的代码尽量少，以及晶振等不好控制的因素。所以鸿哥一直不推荐在实际项目中用单片机的内部定时器做实时时钟，因为精度有限。

真正想要准确的时钟时间，还是强烈建议大家用外部专用的时钟芯片或者用CPLD/FPGA来做。

具体内容，请看源代码讲解。

（1 硬件平台.

基于朱兆祺51单片机学习板。

（2）实现功能:

本程序有2两个窗口。

第1个窗口显示日期。显示格式“年-月-日”。注意中间有“-”分开。

第2个窗口显示时间。显示格式“时 分 秒”。注意中间没“-”，只有空格分开。

系统上电后，默认显示第2个窗口，实时显示动态的“时 分 秒”时间。此时按下S13按键不松手就会切换到显示日期的第1个窗口。松手后自动切换回第2个显示动态时间的窗口。

需要更改时间的时候，长按S9按键不松手超过3秒后，系统将进入修改时间的状态，切换到第1个日期窗口，并且显示“年”的两位数码管会闪烁，此时可以按S1或者S5加减按键修改年的参数，修改完年后，继续短按S9按键，会切换到“月”的参数闪烁状态，只要依次不断按下S9按键，就会依次切换年，月，日，时，分，秒的参数闪烁状态，最后修改完秒的参数后，系统会自动把我们修改设置的日期时间一次性更改到定时中断函数内部的时间变量，达到修改日期时间的目的。

S13是电平变化按键，用来切换窗口的，专门用来查看当前日期。按下S13按键时显示日期窗口，松手后返回到显示实时时间的窗口。

[size=10.5000pt]（3）源代码讲解如下:

```
#include "REG52.H"
#define const_dpy_time_half  200  //数码管闪烁时间的半值
#define const_dpy_time_all   400  //数码管闪烁时间的全值 一定要比const_dpy_time_half 大
#define const_voice_short  40   //蜂鸣器短叫的持续时间
#define const_key_time1  20    //按键去抖动延时的时间
#define const_key_time2  20    //按键去抖动延时的时间
#define const_key_time3  20    //按键去抖动延时的时间
#define const_key_time4  20    //按键去抖动延时的时间
#define const_key_time17  1200  //长按超过3秒的时间
/* 注释一:
  * const_timer_1s这个是产生多少次定时中断才算1秒钟的标准。这个标准决定了时钟的精度。这个标准最后是需要校验的。
  * 那么是如何检验的呢？根据我们前面介绍的校验时间方法:
  * 步骤:
  * 第一步: 在程序代码上先写入1秒钟大概需要200个定时中断。
  * 第二步: 把程序烧录进单片机后，上电开始测试，手上同步打开手机里的秒表，当手机的标准时间跑了780秒(这个标准时间跑得越长校验精度越高),
  *          而此时单片机仅仅跑了1632秒。那么最终得出1秒钟需要的定时中断次数是
:const_time_1s=(200*1632)/780=418。
  * 第三步: 如果发现时钟还是不太准，可以继续返回第一步根据最新1秒钟的时间是418次，多校验几次。本系统仅供学习，精度不可能做得很好，因为
  *          影响时间精度的因素还有定时中断的重装值，定时中断里面的代码尽量少，以及晶振等不好控制的因素。所以鸿哥一直不推荐在实际项目中
  *          用单片机的内部定时器做实时时钟，因为精度有限。真正想要准确的时钟时间，还是强烈建议大家用外部专用的时钟芯片或者用CPLD/FPGA来做。
  */
//#define const_timer_1s  200   //第一次假设大概1秒的时间需要200个定时中断
#define const_timer_1s  418  //第二次校验后，最终选定大概1秒的时间需要418个定时中断。如果发现时间还是不准，可以在此基础上继续校验来调整此数据。
void initial_myself(void);
void initial_peripheral(void);
void delay_short(unsigned int uiDelayShort);
```

```
void delay_long(unsigned int uiDelaylong);
//驱动数码管的74HC595
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01);
void display_drive(void); //显示数码管字模的驱动函数
void display_service(void); //显示的窗口菜单服务程序
//驱动LED的74HC595
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01);
void T0_time(void);  //定时中断函数
void key_service(void); //按键服务的应用程序
void key_scan(void);//按键扫描函数 放在定时中断里
void timer_sampling(void); //定时器采样程序，内部每秒钟采集更新一次
unsigned char get_date(unsigned char ucYearTemp,unsigned char ucMonthTemp);  //获取当前月份的最大天数
//日调整 每个月份的日最大取值不同，有的最大28日，有的最大29日，有的最大30，有的最大31
unsigned char date_adjust(unsigned char ucYearTemp,unsigned char ucMonthTemp,unsigned char ucDateTemp);
//日调整
sbit key_sr1=P0^0; //对应朱兆祺学习板的S1键
sbit key_sr2=P0^1; //对应朱兆祺学习板的S5键
sbit key_sr3=P0^2; //对应朱兆祺学习板的S9键
sbit key_sr4=P0^3; //对应朱兆祺学习板的S13键
sbit key_gnd_dr=P0^4; //模拟独立按键的地GND，因此必须一直输出低电平
sbit beep_dr=P2^7; //蜂鸣器的驱动IO口
sbit dig_hc595_sh_dr=P2^0;      //数码管的74HC595程序
sbit dig_hc595_st_dr=P2^1;
sbit dig_hc595_ds_dr=P2^2;
sbit hc595_sh_dr=P2^3;     //LED灯的74HC595程序
sbit hc595_st_dr=P2^4;
sbit hc595_ds_dr=P2^5;
unsigned char ucKeySec=0;    //被触发的按键编号
unsigned int  uiKeyTimeCnt1=0; //按键去抖动延时计数器
unsigned char ucKeyLock1=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt2=0; //按键去抖动延时计数器
unsigned char ucKeyLock2=0; //按键触发后自锁的变量标志
unsigned int  uiKeyTimeCnt3=0; //按键去抖动延时计数器
unsigned char ucKeyLock3=0; //按键触发后自锁的变量标志
unsigned int uiKey4Cnt1=0;  //在软件滤波中，用到的变量
unsigned int uiKey4Cnt2=0;
unsigned char ucKey4Sr=1;  //实时反映按键的电平状态
unsigned char ucKey4SrRecord=0; //记录上一次按键的电平状态
unsigned int  uiVoiceCnt=0;  //蜂鸣器鸣叫的持续时间计数器
unsigned char  ucVoiceLock=0;  //蜂鸣器鸣叫的原子锁
unsigned char ucDigShow8;  //第8位数码管要显示的内容
unsigned char ucDigShow7;  //第7位数码管要显示的内容
unsigned char ucDigShow6;  //第6位数码管要显示的内容
unsigned char ucDigShow5;  //第5位数码管要显示的内容
unsigned char ucDigShow4;  //第4位数码管要显示的内容
unsigned char ucDigShow3;  //第3位数码管要显示的内容
unsigned char ucDigShow2;  //第2位数码管要显示的内容
unsigned char ucDigShow1;  //第1位数码管要显示的内容
unsigned char ucDigDot8;  //数码管8的小数点是否显示的标志
unsigned char ucDigDot7;  //数码管7的小数点是否显示的标志
```

```
unsigned char ucDigDot6;  //数码管6的小数点是否显示的标志
unsigned char ucDigDot5;  //数码管5的小数点是否显示的标志
unsigned char ucDigDot4;  //数码管4的小数点是否显示的标志
unsigned char ucDigDot3;  //数码管3的小数点是否显示的标志
unsigned char ucDigDot2;  //数码管2的小数点是否显示的标志
unsigned char ucDigDot1;  //数码管1的小数点是否显示的标志
unsigned char ucDigShowTemp=0; //临时中间变量
unsigned char ucDisplayDriveStep=1;  //动态扫描数码管的步骤变量
unsigned char ucWd=2;  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
unsigned char ucPart=0;//本程序的核心变量，局部显示变量。类似于二级菜单的变量。代表显示不同的局部。
unsigned char ucWd1Update=0; //窗口1更新显示标志
unsigned char ucWd2Update=1; //窗口2更新显示标志
unsigned char ucWd1Part1Update=0;  //在窗口1中，局部1的更新显示标志
unsigned char ucWd1Part2Update=0; //在窗口1中，局部2的更新显示标志
unsigned char ucWd1Part3Update=0; //在窗口1中，局部3的更新显示标志
unsigned char ucWd2Part1Update=0;  //在窗口2中，局部1的更新显示标志
unsigned char ucWd2Part2Update=0; //在窗口2中，局部2的更新显示标志
unsigned char ucWd2Part3Update=0; //在窗口2中，局部3的更新显示标志
unsigned char  ucYear=15;    //用来显示和设置的时间变量
unsigned char  ucMonth=1;
unsigned char  ucDate=1;
unsigned char  ucHour=12;
unsigned char  ucMinute=0;
unsigned char  ucSecond=0;
unsigned int uiTimerCnt=0;   //计时器的时基
unsigned char  ucTimerYear=15;   //在定时器内部时基产生的时间变量
unsigned char  ucTimerMonth=1;
unsigned char  ucTimerDate=1;
unsigned char  ucTimerHour=12;
unsigned char  ucTimerMinute=0;
unsigned char  ucTimerSecond=0;
unsigned char  ucTimerDateMax=31; //当前月份的最大天数
unsigned char  ucTimerUpdate=0; //定时器每1秒钟所产生的标志
unsigned char  ucTimerStart=1;  //是否打开定时器内部时间的标志，在本程序相当于原子锁的作用。
unsigned char ucTemp1=0;  //中间过渡变量
unsigned char ucTemp2=0;  //中间过渡变量
unsigned char ucTemp4=0;  //中间过渡变量
unsigned char ucTemp5=0;  //中间过渡变量
unsigned char ucTemp7=0;  //中间过渡变量
unsigned char ucTemp8=0;  //中间过渡变量
unsigned char ucDelayTimerLock=0; //原子锁
unsigned int  uiDelayTimer=0;
unsigned char ucDpyTimeLock=0; //原子锁
unsigned int  uiDpyTimeCnt=0;  //数码管的闪烁计时器,放在定时中断里不断累加
//根据原理图得出的共阴数码管字模表
code unsigned char dig_table[]=
{
0x3f,  //0       序号0
0x06,  //1       序号1
0x5b,  //2       序号2
```

```
0x4f,  //3       序号3
0x66,  //4       序号4
0x6d,  //5       序号5
0x7d,  //6       序号6
0x07,  //7       序号7
0x7f,  //8       序号8
0x6f,  //9       序号9
0x00,  //无      序号10
0x40,  //-       序号11
0x73,  //P       序号12
};
void main()
  {
   initial_myself();
   delay_long(100);
   initial_peripheral();
   while(1)
   {
      key_service(); //按键服务的应用程序
      timer_sampling(); //定时器采样程序，内部每秒钟采集更新一次
      display_service(); //显示的窗口菜单服务程序
   }
}
/* 注释二:
  * 系统不用时时刻刻采集定时器的内部数据，每隔1秒钟的时间更新采集一次就可以了。
  * 这个1秒钟的时间是根据定时器内部ucTimerUpdate变量来判断。
  */
void timer_sampling(void) //采样定时器的程序，内部每秒钟采集更新一次
{
   if(ucPart==0)  //当系统不是处于设置日期和时间的情况下
   {
      if(ucTimerUpdate==1)  //每隔1秒钟时间就更新采集一次定时器的时间数据
      {
          ucTimerUpdate=0;  //及时清零，避免一直更新。
          ucYear=ucTimerYear; //读取定时器内部的年
          ucMonth=ucTimerMonth; //读取定时器内部的月
          ucDate=ucTimerDate;  //读取定时器内部的日
          ucHour=ucTimerHour;   //读取定时器内部的时
          ucMinute=ucTimerMinute;  //读取定时器内部的分
          ucSecond=ucTimerSecond;  //读取定时器内部的秒
          ucWd2Update=1; //窗口2更新显示时间
      }
   }
}
/* 注释三:
  * 根据年份和月份来获取当前这个月的最大天数。每个月份的天数最大取值不同，有的最大28日，
  * 有的最大29日，有的最大30，有的最大31。
  */
unsigned char get_date(unsigned char ucYearTemp,unsigned char ucMonthTemp)
{
```

```
   unsigned char ucDayResult;
   unsigned int uiYearTemp;
   unsigned int uiYearYu;
   ucDayResult=31; //默认最大是31天，以下根据不同的年份和月份来决定是否需要修正这个值
   switch(ucMonthTemp)  //根据不同的月份来获取当前月份天数的最大值
   {
      case 2:  //二月份要计算是否是闰年
           uiYearTemp=2000+ucYearTemp;
           uiYearYu=uiYearTemp%4;
           if(uiYearYu==0) //闰年
           {
              ucDayResult=29;
           }
           else
           {
              ucDayResult=28;
           }
           break;
      case 4:
      case 6:
      case 9:
      case 11:
              ucDayResult=30;
           break;
   }
   return ucDayResult;
}
//日调整 每个月份的日最大取值不同，有的最大28日，有的最大29日，有的最大30，有的最大31
unsigned char date_adjust(unsigned char ucYearTemp,unsigned char ucMonthTemp,unsigned char ucDateTemp)
//日调整
{
   unsigned char ucDayResult;
   unsigned int uiYearTemp;
   unsigned int uiYearYu;

   ucDayResult=ucDateTemp;
   switch(ucMonthTemp)  //根据不同的月份来修正不同的日最大值
   {
      case 2:  //二月份要计算是否是闰年
           uiYearTemp=2000+ucYearTemp;
           uiYearYu=uiYearTemp%4;
           if(uiYearYu==0) //闰年
           {
               if(ucDayResult>29)
               {
                  ucDayResult=29;
               }
           }
           else
           {
```

```
                if(ucDayResult>28)
                {
                   ucDayResult=28;
                }
            }
            break;
       case 4:
       case 6:
       case 9:
       case 11:
            if(ucDayResult>30)
            {
               ucDayResult=30;
            }
            break;
   }
   return ucDayResult;
}
void display_service(void) //显示的窗口菜单服务程序
{
   switch(ucWd)  //本程序的核心变量，窗口显示变量。类似于一级菜单的变量。代表显示不同的窗口。
   {
       case 1:   //显示日期窗口的数据  数据格式 NN-YY-RR 年-月-日
            if(ucWd1Update==1)  //窗口1要全部更新显示
            {
               ucWd1Update=0;  //及时清零标志，避免一直进来扫描
               ucDigShow6=11;  //显示一杠"-"
               ucDigShow3=11;  //显示一杠"-"
               ucWd1Part1Update=1;  //局部年更新显示
               ucWd1Part2Update=1;  //局部月更新显示
               ucWd1Part3Update=1;  //局部日更新显示
            }
            if(ucWd1Part1Update==1)//局部年更新显示
            {
               ucWd1Part1Update=0;
               ucTemp8=ucYear/10;  //年
               ucTemp7=ucYear%10;
               ucDigShow8=ucTemp8; //数码管显示实际内容
               ucDigShow7=ucTemp7;
            }
            if(ucWd1Part2Update==1)//局部月更新显示
            {
               ucWd1Part2Update=0;
               ucTemp5=ucMonth/10;  //月
               ucTemp4=ucMonth%10;
               ucDigShow5=ucTemp5; //数码管显示实际内容
               ucDigShow4=ucTemp4;
            }
            if(ucWd1Part3Update==1) //局部日更新显示
            {
```

```
            ucWd1Part3Update=0;
            ucTemp2=ucDate/10;  //日
            ucTemp1=ucDate%10;

            ucDigShow2=ucTemp2; //数码管显示实际内容
            ucDigShow1=ucTemp1;
        }
          //数码管闪烁
        switch(ucPart)  //相当于二级菜单，根据局部变量的值，使对应的参数产生闪烁的动态效果。
        {
            case 0:  //都不闪烁
                 break;
            case 1:  //年参数闪烁
                 if(uiDpyTimeCnt==const_dpy_time_half)
                 {
                       ucDigShow8=ucTemp8; //数码管显示实际内容
                       ucDigShow7=ucTemp7;
                  }
                 else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                 {
                       ucDpyTimeLock=1; //原子锁加锁
                       uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                       ucDpyTimeLock=0;  //原子锁解锁
                       ucDigShow8=10;   //数码管显示空，什么都不显示
                       ucDigShow7=10;
                 }
                 break;
            case 2:   //月参数闪烁
                 if(uiDpyTimeCnt==const_dpy_time_half)
                 {
                       ucDigShow5=ucTemp5; //数码管显示实际内容
                       ucDigShow4=ucTemp4;
                  }
                 else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                 {
                       ucDpyTimeLock=1; //原子锁加锁
                       uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                       ucDpyTimeLock=0;  //原子锁解锁
                       ucDigShow5=10;   //数码管显示空，什么都不显示
                       ucDigShow4=10;
                 }
                break;
            case 3:   //日参数闪烁
                 if(uiDpyTimeCnt==const_dpy_time_half)
                 {
                       ucDigShow2=ucTemp2; //数码管显示实际内容
                       ucDigShow1=ucTemp1;
                  }
```

```
                    else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                    {
                          ucDpyTimeLock=1; //原子锁加锁
                          uiDpyTimeCnt=0;    //及时把闪烁记时器清零
                          ucDpyTimeLock=0;  //原子锁解锁
                          ucDigShow2=10;    //数码管显示空，什么都不显示
                          ucDigShow1=10;
                    }
                   break;
             }
             break;
       case 2:   //显示时间窗口的数据  数据格式 SS FF MM 时 分 秒
             if(ucWd2Update==1)  //窗口2要全部更新显示
             {
                ucWd2Update=0;  //及时清零标志，避免一直进来扫描
                ucDigShow6=10;  //显示空
                ucDigShow3=10;  //显示空
                ucWd2Part3Update=1;  //局部时更新显示
                ucWd2Part2Update=1;  //局部分更新显示
                ucWd2Part1Update=1;  //局部秒更新显示
             }
             if(ucWd2Part1Update==1)//局部时更新显示
             {
                ucWd2Part1Update=0;
                ucTemp8=ucHour/10;  //时
                ucTemp7=ucHour%10;
                ucDigShow8=ucTemp8; //数码管显示实际内容
                ucDigShow7=ucTemp7;
             }
             if(ucWd2Part2Update==1)//局部分更新显示
             {
                ucWd2Part2Update=0;
                ucTemp5=ucMinute/10;  //分
                ucTemp4=ucMinute%10;
                ucDigShow5=ucTemp5; //数码管显示实际内容
                ucDigShow4=ucTemp4;
             }
             if(ucWd2Part3Update==1) //局部秒更新显示
             {
                ucWd2Part3Update=0;
                ucTemp2=ucSecond/10;  //秒
                ucTemp1=ucSecond%10;

                ucDigShow2=ucTemp2; //数码管显示实际内容
                ucDigShow1=ucTemp1;
             }
               //数码管闪烁
             switch(ucPart)  //相当于二级菜单，根据局部变量的值，使对应的参数产生闪烁的动态效果。
             {
```

```
                case 0:  //都不闪烁
                     break;
                case 1:  //时参数闪烁
                     if(uiDpyTimeCnt==const_dpy_time_half)
                     {
                           ucDigShow8=ucTemp8; //数码管显示实际内容
                           ucDigShow7=ucTemp7;
                      }
                     else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                     {
                           ucDpyTimeLock=1; //原子锁加锁
                           uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                           ucDpyTimeLock=0;  //原子锁解锁
                           ucDigShow8=10;    //数码管显示空，什么都不显示
                           ucDigShow7=10;
                     }
                     break;
                case 2:   //分参数闪烁
                     if(uiDpyTimeCnt==const_dpy_time_half)
                     {
                           ucDigShow5=ucTemp5; //数码管显示实际内容
                           ucDigShow4=ucTemp4;
                     }
                     else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                     {
                           ucDpyTimeLock=1; //原子锁加锁
                           uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                           ucDpyTimeLock=0;  //原子锁解锁
                           ucDigShow5=10;    //数码管显示空，什么都不显示
                           ucDigShow4=10;
                     }
                    break;
                case 3:   //秒参数闪烁
                     if(uiDpyTimeCnt==const_dpy_time_half)
                     {
                           ucDigShow2=ucTemp2; //数码管显示实际内容
                           ucDigShow1=ucTemp1;
                     }
                     else if(uiDpyTimeCnt>const_dpy_time_all) //const_dpy_time_all一定要比
const_dpy_time_half 大
                     {
                           ucDpyTimeLock=1; //原子锁加锁
                           uiDpyTimeCnt=0;   //及时把闪烁记时器清零
                           ucDpyTimeLock=0;  //原子锁解锁
                           ucDigShow2=10;    //数码管显示空，什么都不显示
                           ucDigShow1=10;
                     }
                    break;
```

```
            }
            break;
        }

}
void key_scan(void)//按键扫描函数 放在定时中断里
{
  if(key_sr1==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock1=0; //按键自锁标志清零
     uiKeyTimeCnt1=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock1==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt1++; //累加定时中断次数
     if(uiKeyTimeCnt1>const_key_time1)
     {
        uiKeyTimeCnt1=0;
        ucKeyLock1=1;  //自锁按键置位,避免一直触发
        ucKeySec=1;    //触发1号键
     }
  }
  if(key_sr2==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock2=0; //按键自锁标志清零
     uiKeyTimeCnt2=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock2==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt2++; //累加定时中断次数
     if(uiKeyTimeCnt2>const_key_time2)
     {
        uiKeyTimeCnt2=0;
        ucKeyLock2=1;  //自锁按键置位,避免一直触发
        ucKeySec=2;    //触发2号键
     }
  }
/* 注释四:
  * 注意，此处把一个按键的短按和长按的功能都实现了。
  */
  if(key_sr3==1)//IO是高电平，说明按键没有被按下，这时要及时清零一些标志位
  {
     ucKeyLock3=0; //按键自锁标志清零
     uiKeyTimeCnt3=0;//按键去抖动延时计数器清零，此行非常巧妙，是我实战中摸索出来的。
  }
  else if(ucKeyLock3==0)//有按键按下，且是第一次被按下
  {
     uiKeyTimeCnt3++; //累加定时中断次数
     if(uiKeyTimeCnt3>const_key_time3)
     {
```

```
        uiKeyTimeCnt3=0;
        ucKeyLock3=1;  //自锁按键置位,避免一直触发
        ucKeySec=3;    //短按触发3号键
      }
  }
  else if(uiKeyTimeCnt3<const_key_time17)   //长按3秒
  {
     uiKeyTimeCnt3++; //累加定时中断次数
         if(uiKeyTimeCnt3==const_key_time17)  //等于3秒钟，触发17号长按按键
         {
            ucKeySec=17;    //长按3秒触发17号键
         }
  }
/* 注释五:
  * 注意，此处是电平按键的滤波抗干扰处理
  */
   if(key_sr4==1)  //对应朱兆祺学习板的S13键
   {
       uiKey4Cnt1=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiKey4Cnt2++; //类似独立按键去抖动的软件抗干扰处理
       if(uiKey4Cnt2>const_key_time4)
       {
           uiKey4Cnt2=0;
           ucKey4Sr=1;  //实时反映按键松手时的电平状态
       }
   }
   else
   {
       uiKey4Cnt2=0; //在软件滤波中，非常关键的语句！！！类似按键去抖动程序的及时清零
       uiKey4Cnt1++;
       if(uiKey4Cnt1>const_key_time4)
       {
          uiKey4Cnt1=0;
          ucKey4Sr=0;  //实时反映按键按下时的电平状态
       }
   }
}
void key_service(void) //按键服务的应用程序
{
  switch(ucKeySec) //按键服务状态切换
  {
    case 1:// 加按键 对应朱兆祺学习板的S1键
          switch(ucWd)  //在不同的窗口下，设置不同的参数
          {
               case 1:
                    switch(ucPart) //在不同的局部变量下，相当于二级菜单
                    {
                        case 1:  //年
                             ucYear++;
                             if(ucYear>99)
```

```
                {
                   ucYear=99;
                }
                ucWd1Part1Update=1;  //更新显示
                break;
            case 2: //月
                ucMonth++;
                if(ucMonth>12)
                {
                    ucMonth=12;
                }
                ucWd1Part2Update=1;  //更新显示

                break;
            case 3: //日
                ucDate++;
                if(ucDate>31)
                {
                   ucDate=31;
                }
                ucWd1Part3Update=1;  //更新显示
                break;
        }
        break;
case 2:
        switch(ucPart) //在不同的局部变量下，相当于二级菜单
        {
            case 1:  //时
                ucHour++;
                if(ucHour>23)
                {
                   ucHour=23;
                }
                ucWd2Part1Update=1;  //更新显示

                break;
            case 2: //分
                ucMinute++;
                if(ucMinute>59)
                {
                   ucMinute=59;
                }
                ucWd2Part2Update=1;  //更新显示

                break;
            case 3: //秒
                ucSecond++;
                if(ucSecond>59)
                {
                    ucSecond=59;
```

```
                    }
                    ucWd2Part3Update=1;  //更新显示
                    break;
                }
                break;

        }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;

case 2:// 减按键 对应朱兆祺学习板的S5键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                switch(ucPart) //在不同的局部变量下，相当于二级菜单
                {
                    case 1:  //年
                        ucYear--;
                        if(ucYear>99)
                        {
                           ucYear=0;
                        }
                        ucWd1Part1Update=1;  //更新显示
                        break;
                    case 2: //月
                        ucMonth--;
                        if(ucMonth<1)
                        {
                           ucMonth=1;
                        }
                        ucWd1Part2Update=1;  //更新显示

                        break;
                    case 3: //日
                        ucDate--;
                        if(ucDate<1)
                        {
                           ucDate=1;
                        }
                        ucWd1Part3Update=1;  //更新显示
                        break;
                }
                break;
            case 2:
                switch(ucPart) //在不同的局部变量下，相当于二级菜单
                {
                     case 1:  //时
```

```
                    ucHour--;
                    if(ucHour>23)
                    {
                       ucHour=0;
                    }
                    ucWd2Part1Update=1;  //更新显示

                    break;
                case 2: //分
                    ucMinute--;
                    if(ucMinute>59)
                    {
                       ucMinute=0;
                    }
                    ucWd2Part2Update=1;  //更新显示

                    break;
                case 3: //秒
                    ucSecond--;
                    if(ucSecond>59)
                    {
                       ucSecond=0;
                    }
                    ucWd2Part3Update=1;  //更新显示
                    break;
            }
            break;

        }
        ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
        uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
        ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
        ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
        break;
    case 3://短按设置按键 对应朱兆祺学习板的S9键
        switch(ucWd)  //在不同的窗口下，设置不同的参数
        {
            case 1:
                ucPart++;
                if(ucPart>3)
                {
                    ucPart=1;
                    ucWd=2; //切换到第二个窗口，设置时分秒
                    ucWd2Update=1;  //窗口2更新显示
                }
                ucWd1Update=1;  //窗口1更新显示
                break;
            case 2:
                if(ucPart>0) //在窗口2的时候，要第一次激活设置时间，必须是长按3秒才可以，这里短按激
活不了第一次
```

```
                    {
                        ucPart++;
                        if(ucPart>3)  //设置时间结束
                        {
                            ucPart=0;
/* 注释六:
  * 每个月份的天数最大值是不一样的，在写入ds1302时钟芯片内部数据前，应该做一次调整。
  * 有的月份最大28天，有的月份最大29天，有的月份最大30天，有的月份最大31天，
  */
                            ucDate=date_adjust(ucYear,ucMonth,ucDate); //日调整 避免日的数值在某个月份超
范围
                            ucTimerStart=0;  //关闭定时器的时间。在更改定时器内部时间数据时，先关闭它，相
当于原子锁的加锁作用。
                            ucTimerYear=ucYear;  //把设置和显示的数据更改到定时器内部的时间变量
                            ucTimerMonth=ucMonth;
                            ucTimerDate=ucDate;
                            ucTimerHour=ucHour;
                            ucTimerMinute=ucMinute;
                            ucTimerSecond=ucSecond;
                            ucTimerStart=1;  //打开定时器的时间。在更改定时器内部时间数据后，再打开它，相
当于原子锁的解锁作用。
                        }
                        ucWd2Update=1;  //窗口2更新显示
                    }
                    break;

            }

            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
            break;
        case 17://长按3秒设置按键 对应朱兆祺学习板的S9键
            switch(ucWd)  //在不同的窗口下，设置不同的参数
            {
                case 2:
                    if(ucPart==0) //处于非设置时间的状态下，要第一次激活设置时间，必须是长按3秒才可以
                    {
                        ucWd=1;
                        ucPart=1;  //进入到设置日期的状态下
                        ucWd1Update=1;  //窗口1更新显示
                    }
                    break;

            }
            ucVoiceLock=1;  //原子锁加锁，保护主函数与中断函数的共享变量uiVoiceCnt
            uiVoiceCnt=const_voice_short; //按键声音触发，滴一声就停。
            ucVoiceLock=0;  //原子锁解锁，保护主函数与中断函数的共享变量uiVoiceCnt
            ucKeySec=0;  //响应按键服务处理程序后，按键编号清零，避免一致触发
```

```
            break;

    }

/* 注释七:
    * 注意，此处就是第一次出现的电平按键程序，跟以往的下降沿按键不一样。
    * ucKey4Sr是经过软件滤波处理后，直接反应IO口电平状态的变量.当电平发生
    * 变化时，就会切换到不同的显示界面，这里多用了一个ucKey4SrRecord变量
    * 记录上一次的电平状态，是为了避免一直刷新显示。
    */
    if(ucKey4Sr!=ucKey4SrRecord)  //说明S13的切换按键电平状态发生变化
    {
       ucKey4SrRecord=ucKey4Sr;//及时记录当前最新的按键电平状态  避免一直进来触发
           if(ucKey4Sr==1) //松手后切换到显示时间的窗口
           {
              ucWd=2;     //显示时分秒的窗口
              ucPart=0;  //进入到非设置时间的状态下
              ucWd2Update=1;  //窗口2更新显示
           }
           else  //按下去切换到显示日期的窗口
           {
              ucWd=1;    //显示年月日的窗口
              ucPart=0;  //进入到非设置时间的状态下
              ucWd1Update=1;  //窗口1更新显示
           }

    }
}
void display_drive(void)
{
   //以下程序，如果加一些数组和移位的元素，还可以压缩容量。但是鸿哥追求的不是容量，而是清晰的讲解思路
   switch(ucDisplayDriveStep)
   {
      case 1:  //显示第1位
           ucDigShowTemp=dig_table[ucDigShow1];
                   if(ucDigDot1==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfe);
               break;
      case 2:  //显示第2位
           ucDigShowTemp=dig_table[ucDigShow2];
                   if(ucDigDot2==1)
                   {
                      ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                   }
           dig_hc595_drive(ucDigShowTemp,0xfd);
               break;
      case 3:  //显示第3位
```

```
        ucDigShowTemp=dig_table[ucDigShow3];
                if(ucDigDot3==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xfb);
            break;
    case 4:  //显示第4位
        ucDigShowTemp=dig_table[ucDigShow4];
                if(ucDigDot4==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xf7);
            break;
    case 5:  //显示第5位
        ucDigShowTemp=dig_table[ucDigShow5];
                if(ucDigDot5==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xef);
            break;
    case 6:  //显示第6位
        ucDigShowTemp=dig_table[ucDigShow6];
                if(ucDigDot6==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0xdf);
            break;
    case 7:  //显示第7位
        ucDigShowTemp=dig_table[ucDigShow7];
                if(ucDigDot7==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
        }
        dig_hc595_drive(ucDigShowTemp,0xbf);
            break;
    case 8:  //显示第8位
        ucDigShowTemp=dig_table[ucDigShow8];
                if(ucDigDot8==1)
                {
                   ucDigShowTemp=ucDigShowTemp|0x80;  //显示小数点
                }
        dig_hc595_drive(ucDigShowTemp,0x7f);
            break;
}
ucDisplayDriveStep++;
if(ucDisplayDriveStep>8)  //扫描完8个数码管后，重新从第一个开始扫描
```

```
    {
      ucDisplayDriveStep=1;
    }
}
//数码管的74HC595驱动函数
void dig_hc595_drive(unsigned char ucDigStatusTemp16_09,unsigned char ucDigStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    dig_hc595_sh_dr=0;
    dig_hc595_st_dr=0;
    ucTempData=ucDigStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucDigStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)dig_hc595_ds_dr=1;
          else dig_hc595_ds_dr=0;
          dig_hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          dig_hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    dig_hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    dig_hc595_st_dr=1;
    delay_short(1);
    dig_hc595_sh_dr=0;    //拉低，抗干扰就增强
    dig_hc595_st_dr=0;
    dig_hc595_ds_dr=0;
}
//LED灯的74HC595驱动函数
void hc595_drive(unsigned char ucLedStatusTemp16_09,unsigned char ucLedStatusTemp08_01)
{
    unsigned char i;
    unsigned char ucTempData;
    hc595_sh_dr=0;
    hc595_st_dr=0;
    ucTempData=ucLedStatusTemp16_09;  //先送高8位
    for(i=0;i<8;i++)
```

```
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    ucTempData=ucLedStatusTemp08_01;  //再先送低8位
    for(i=0;i<8;i++)
    {
          if(ucTempData>=0x80)hc595_ds_dr=1;
          else hc595_ds_dr=0;
          hc595_sh_dr=0;      //SH引脚的上升沿把数据送入寄存器
          delay_short(1);
          hc595_sh_dr=1;
          delay_short(1);
          ucTempData=ucTempData<<1;
    }
    hc595_st_dr=0;  //ST引脚把两个寄存器的数据更新输出到74HC595的输出引脚上并且锁存起来
    delay_short(1);
    hc595_st_dr=1;
    delay_short(1);
    hc595_sh_dr=0;    //拉低，抗干扰就增强
    hc595_st_dr=0;
    hc595_ds_dr=0;
}
void T0_time(void) interrupt 1   //定时中断
{
  TF0=0;  //清除中断标志
  TR0=0; //关中断
/* 注释八:
  * 以下是本节内容的核心程序，是定时器内部产生的时间。const_timer_1s这个是产生多少次定时中断才
  * 算1秒钟的标准。这个标准决定了时钟的精度。这个标准最后是需要校验的。
  */
  if(ucTimerStart==1)  //定时器的时间已经打开
  {
     uiTimerCnt++;  //产生1秒钟的时基
     if(uiTimerCnt>=const_timer_1s) //一秒钟的时间到。这个const_timer_1s具体数值最后需要校验得出。
     {
        uiTimerCnt=0; //清零为产生下一个1秒钟准备
        ucTimerUpdate=1; //定时器每1秒钟所产生的标志，通知主函数及时更新采集时间数据
        ucTimerSecond++; //秒时间累加1
        if(ucTimerSecond>=60)
        {
           ucTimerSecond=0;
           ucTimerMinute++; //分时间累加1
           if(ucTimerMinute>=60)
           {
```

```
                ucTimerMinute=0;
                ucTimerHour++;  //小时的时间累加1，为了避免if的嵌套过多，把小时的判断放到外面两层的if来继续判断
            }
        }
        if(ucTimerHour>=24)
        {
            ucTimerHour=0;
            ucTimerDate++; //天时间累加1
            ucTimerDateMax=get_date(ucTimerYear,ucTimerMonth);  //根据年和月获取当前月份的最大天数
            if(ucTimerDate>ucTimerDateMax)  //
            {
                ucTimerDate=1; //每个月都是从1号开始
                ucTimerMonth++;  //月时间累加1
                if(ucTimerMonth>12)
                {
                    ucTimerMonth=1; //每年从1月份开始
                    ucTimerYear++; //年时间累加1
                    if(ucTimerYear>99) //本系统的最高有效年份是2099年
                    {
                       ucTimerYear=99;
                    }
                }
            }
        }
    }

  }
  if(ucVoiceLock==0) //原子锁判断
  {
     if(uiVoiceCnt!=0)
     {
        uiVoiceCnt--; //每次进入定时中断都自减1，直到等于零为止。才停止鸣叫
        beep_dr=0;  //蜂鸣器是PNP三极管控制，低电平就开始鸣叫。

     }
     else
     {
        ; //此处多加一个空指令，想维持跟if括号语句的数量对称，都是两条指令。不加也可以。
        beep_dr=1;  //蜂鸣器是PNP三极管控制，高电平就停止鸣叫。

     }
  }
  if(ucDpyTimeLock==0) //原子锁判断
  {
     uiDpyTimeCnt++;  //数码管的闪烁计时器
  }
  key_scan(); //按键扫描函数
  display_drive();  //数码管字模的驱动函数
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
```

```
  TL0=0x0b;
  TR0=1;  //开中断
}
void delay_short(unsigned int uiDelayShort)
{
   unsigned int i;
   for(i=0;i<uiDelayShort;i++)
   {
     ;   //一个分号相当于执行一条空语句
   }
}
void delay_long(unsigned int uiDelayLong)
{
   unsigned int i;
   unsigned int j;
   for(i=0;i<uiDelayLong;i++)
   {
      for(j=0;j<500;j++)  //内嵌循环的空指令数量
          {
             ; //一个分号相当于执行一条空语句
          }
   }
}
void initial_myself(void)  //第一区 初始化单片机
{
  key_gnd_dr=0; //模拟独立按键的地GND，因此必须一直输出低电平
  beep_dr=1; //用PNP三极管控制蜂鸣器，输出高电平时不叫。
  hc595_drive(0x00,0x00);  //关闭所有经过另外两个74HC595驱动的LED灯
  TMOD=0x01;  //设置定时器0为工作方式1
  TH0=0xfe;   //重装初始值(65535-500)=65035=0xfe0b
  TL0=0x0b;
}
void initial_peripheral(void) //第二区 初始化外围
{
   ucDigDot8=0;   //小数点全部不显示
   ucDigDot7=0;
   ucDigDot6=0;
   ucDigDot5=0;
   ucDigDot4=0;
   ucDigDot3=0;
   ucDigDot2=0;
   ucDigDot1=0;
   EA=1;     //开总中断
   ET0=1;    //允许定时中断
   TR0=1;    //启动定时中断
}
```

复制代码

总结陈词:

任何一个电子产品在投入生产的时候都要考虑到生产的测试，朱兆祺51单片机学习板在生产加工后也一样要进行测试。那么这个测试的程序如何能够做到快速，全面，易用这三个要求呢？欲知详情，请听下回分解-----生产朱兆祺51学习

板的从机自检测试程序源代码.。
（未完待续，下节更精彩，不要走开哦）