51 单片机

及 C 语言入门教程

注：排成 16 开版式，是为了方便自己打印阅读。请不要用于非法用途。

2007.12.20

51 单片机及C语言入门教程

第一课　建立您的第一个 C 项目

使用C语言肯定要使用到C编译器，以便把写好的C程序编译为机器码，这样单片机才能执行编写好的程序。KEIL uVISION2是众多单片机应用开发软件中优秀的软件之一，它支持众多不同公司的MCS51架构的芯片，它集编辑，编译，仿真等于一体，同时还支持，PLM，汇编和C语言的程序设计，它的界面和常用的微软VC++的界面相似，界面友好，易学易用，在调试程序，软件仿真方面也有很强大的功能。因此很多开发51应用的工程师或普通的单片机爱好者，都对它十分喜欢。

以上简单介绍了KEIL51软件，要使用KEIL51软件，必需先要安装它。KEIL51是一个商业的软件，对于我们这些普通爱好者可以到KEIL中国代理周立功公司的网站上下载一份能编译2K的DEMO版软件，基本可以满足一般的个人学习和小型应用的开发。（安装的方法和普通软件相当这里就不做介绍了）

安装好后，你是不是迫不及待的想建立自己的第一个C程序项目呢？下面就让我们一起来建立一个小程序项目吧。或许你手中还没有一块实验板，甚至没有一块单片机，不过没有关系我们可以通过KEIL软件仿真看到程序运行的结果。

首先当然是运行KEIL51软件。怎么打开？噢，天！那你要从头学电脑了。呵呵，开个玩笑，这个问题我想读者们也不会提的了：P。运行几秒后，出现如图1－1的屏幕。

图1－1　启动时的屏幕

接着按下面的步骤建立您的第一个项目：

（1）点击Project菜单，选择弹出的下拉式菜单中的New Project，如图1－2。接着弹出一个标准Windows文件对话窗口，如图1－3，这个东东想必大家是见了N次的了，用法技巧也不是这里要说的，以后的章节中出现类似情况将不再说明。在"文件名"中输入您的第一个C程序项目名称，这里我们用"test"，这是笔者惯用的名称，大家不必照搬就是了，只要符合Windows文件规则的文件名都行。"保存"后的文件扩展名为uv2，这是KEIL uVision2项目文件扩展名，以后我们可以直接点击此文件以打开先前做的项目。

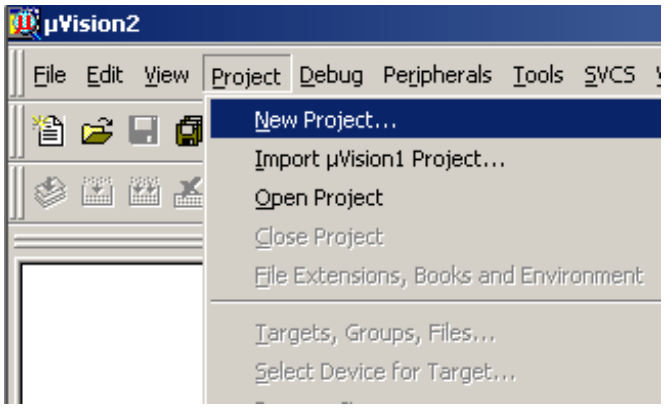


图1－2　New Project菜单

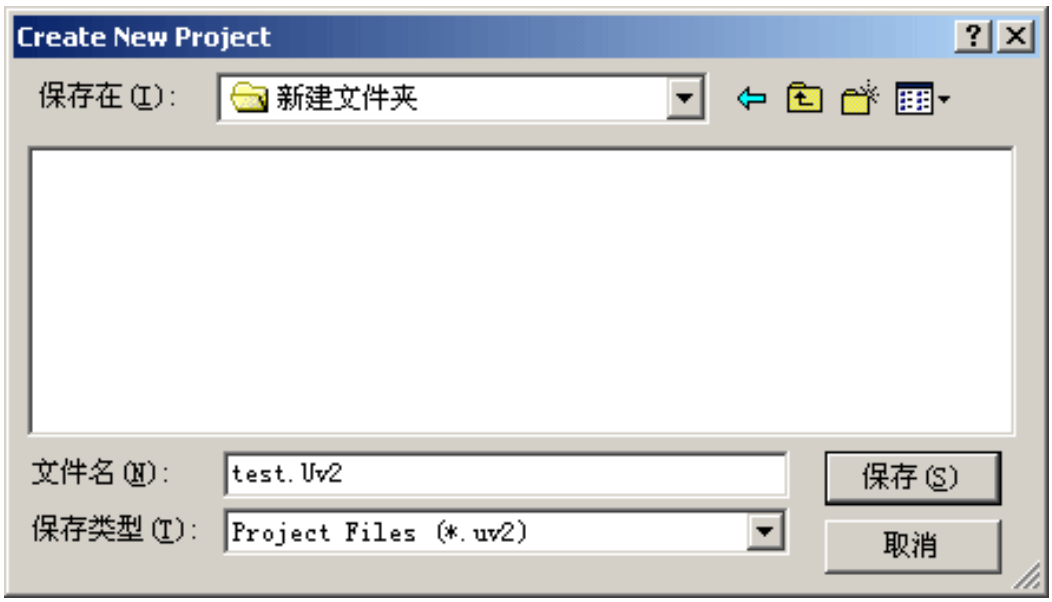


图1－3　文件窗口

（2）选择所要的单片机，这里我们选择常用的Ateml公司的AT89C51。此时屏幕如图1－4

所示。AT89C51有什么功能、特点呢？不用急，看图中右边有简单的介绍，稍后的章节会作较详细的介绍。完成上面步骤后，我们就可以进行程序的编写了。

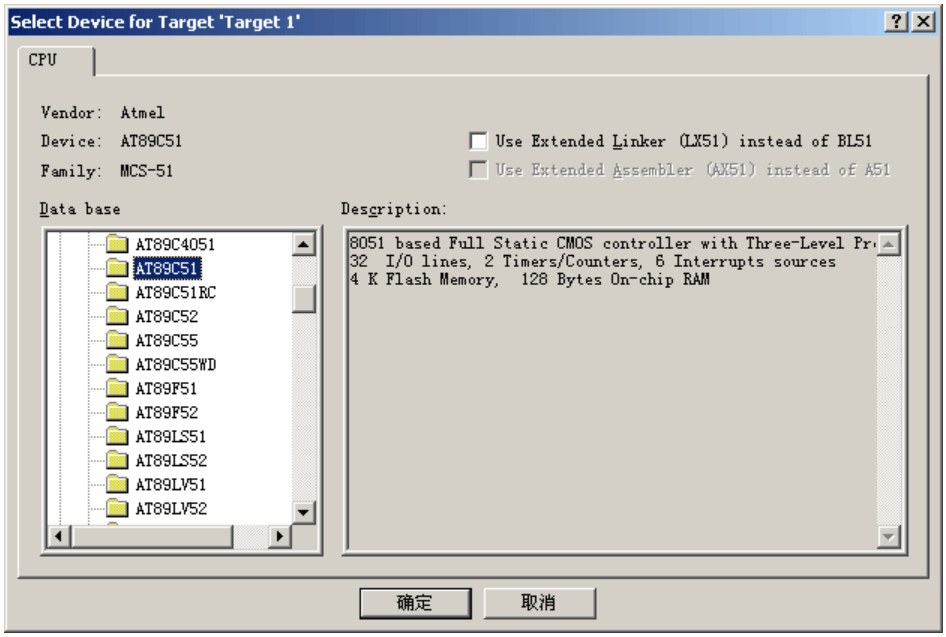


图1－4选取芯片

（3）首先我们要在项目中创建新的程序文件或加入旧程序文件。如果你没有现成的程序，那么就要新建一个程序文件。在KEIL中有一些程序的Demo，在这里我们还是以一个C程序为例介绍如何新建一个C程序和如何加到您的第一个项目中吧。点击图1－5中1的新建文件的快捷按钮，在2中出现一个新的文字编辑窗口，这个操作也可以通过菜单File－New或快捷键Ctrl+N来实现。好了，现在可以编写程序了，光标已出现在文本编辑窗口中，等待我们的输入了。第一程序嘛，写个简单明了的吧。下面是经典的一段程序，呵，如果你看过别的程序书也许也有类似的程序：

```
#include
#include void main(void)
{
  SCON = 0x50; //串口方式1,允许接收
  TMOD = 0x20; //定时器1定时方式2
  TCON = 0x40; //设定时器1开始计数
  TH1 = 0xE8; //11.0592MHz 1200波特率
  TL1 = 0xE8;
  TI = 1;
  TR1 = 1; //启动定时器   while(1)
    {
      printf ("Hello World!\n"); //显示Hello World
    }
}
```

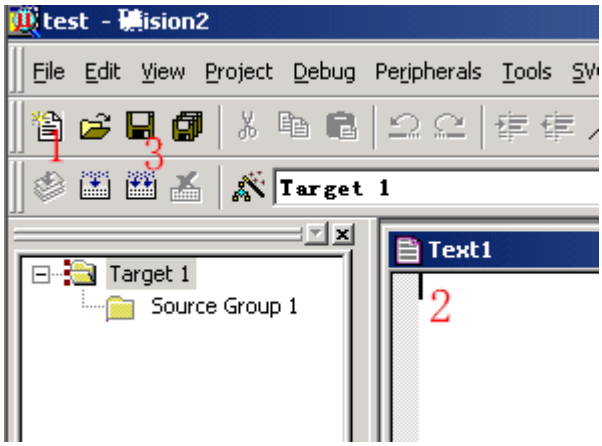


图1－5新建程序文件

这段程序的功能是不断从串口输出"Hello World!"字符，我们先不管程序的语法和意思吧，先看看如何把它加入到项目中和如何编译试运行。

（4）点击图1－5中的3保存新建的程序，也可以用菜单File－Save或快捷键Ctrl+S进行保存。因是新文件所以保存时会弹出类似图1－3的文件操作窗口，我们把第一个程序命名为test1.c，保存在项目所在的目录中，这时你会发现程序单词有了不同的颜色，说明KEIL的C语法检查生效了。如图1－6鼠标在屏幕左边的Source Group1文件夹图标上右击弹出菜单，在这里可以做在项目中增加减少文件等操作。我们?quot;Add File to Group 'Source Group 1'"弹出文件窗口，选择刚刚保存的文件，按ADD按钮，关闭文件窗，程序文件已加到项目中了。这时在Source Group1文件夹图标左边出现了一个小+号说明，文件组中有了文件，点击它可以展开查看。

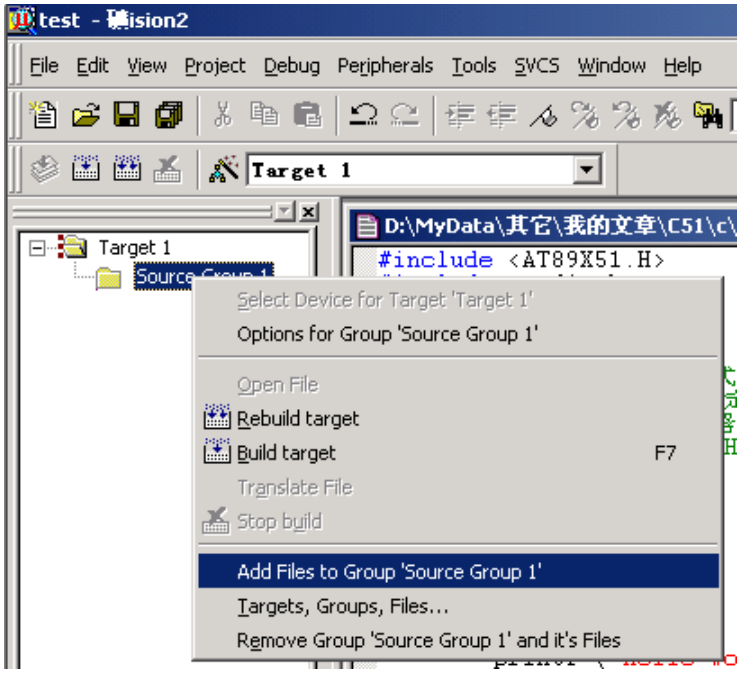


图1－6把文件加入到项目文件组中

（5）C程序文件已被我们加到了项目中了，下面就剩下编译运行了。这个项目我们只是用做学习新建程序项目和编译运行仿真的基本方法，所以使用软件默认的编译设置，它不会生成用于芯片烧写的HEX文件，如何设置生成HEX文件就请看下面的第三课。我们先来看图1－7吧，图中1、2、3都是编译按钮，不同是1是用于编译单个文件。2是编译当前项目，如果先前编译过一次之后文件没有做动编辑改动，这时再点击是不会再次重新编译的。3是重新编译，每点击一次均会再次编译链接一次，不管程序是否有改动。在3右边的是停止编译按钮，只有点击了前三个中的任一个，停止按钮才会生效。5是菜单中的它们，我个人就不习惯用它了。嘿嘿，这个项目只有一个文件，你按123中的一个都可以编译。按了？好快哦，呵呵。在4中可以看到编译的错误信息和使用的系统资源情况等，以后我们要查错就靠它了。6是有一个小放大镜的按钮，这就是开启\关闭调试模式的按钮，它也存在于菜单Debug－Start\Stop Debug Session，快捷键为Ctrl+F5。

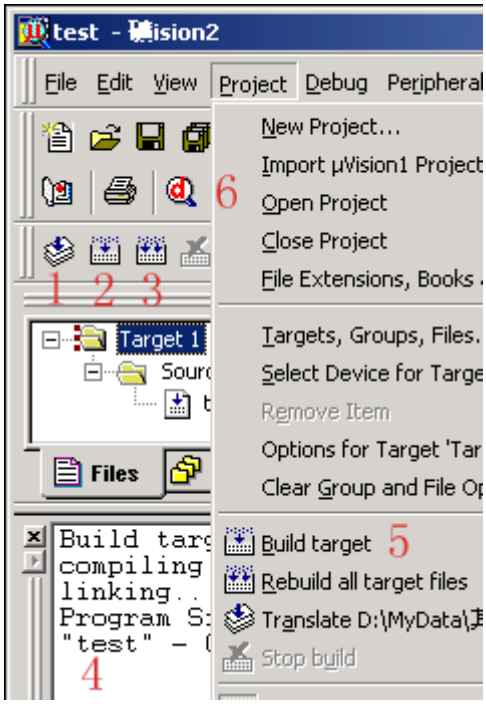


图1－7编译程序

(6)进入调试模式，软件窗口样式大致如图1－8所示。图中1为运行，当程序处于停止状态时才有效，2为停止，程序处于运行状态时才有效。3是复位，模拟芯片的复位，程序回到最开头处执行。按4我们可以打开5中的串行调试窗口，这个窗口我们可以看到从51芯片的串行口输入输出的字符，这里的第一个项目也正是在这里看运行结果。这些在菜单中也有，这里不再一一介绍大家不妨找找看，其它的功能也会在后面的课程中慢慢介绍。首先按4打开串行调试窗口，再按运行键，这时就可以看到串行调试窗口中不断的打?quot;Hello World！"。呵呵，是不是不难呀？这样就完成了您的第一个C项目。最后我们要停止程序运行回到文件编辑模式中，就要先按停止按钮再按开启\关闭调试模式按钮。然后我们就可以进行关闭KEIL等相关操作了。

到此为止，第一课已经完结了，初步学习了一些KEIL uVision2的项目文件创建、编译、运

行和软件仿真的基本操作方法。其中一直有提到一些功能的快捷键的使用，的确在实际的开发应用中快捷键的运用可以大大提高工作的效率，建议大家多多使用，还有就是对这里所讲的操作方法举一反三用于类似的操作中。

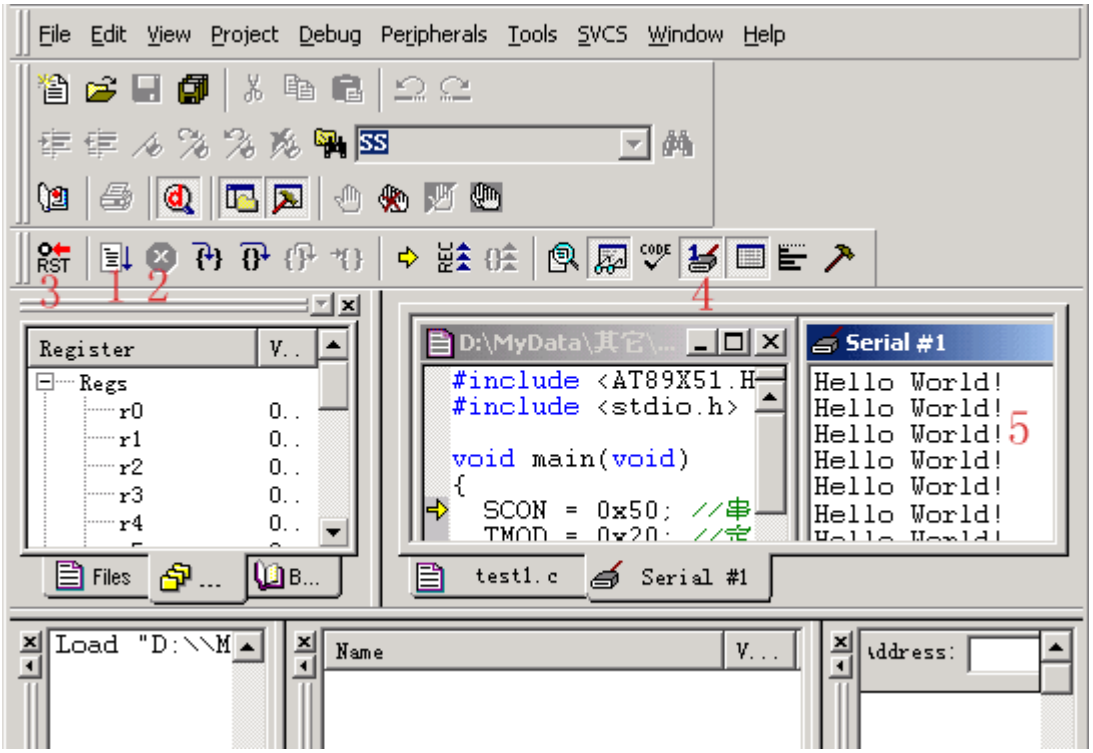


图1－8调试运行程序

第二课　初步认识51芯片

上一课我们的第一个项目完成了，可能有懂C语言的朋友会说，"这和PC机上的C语言没有多大的区别呀"。的确没有太大的区别，C语言只是一种程序语言的统称，针对不同的处理器相关的C语言都会有一些细节的改变。编写PC机的C程序时，如要对硬件编程你就必须对硬件要有一定的认识，51单片机编程就更是如此，因它的开发应用是不可与硬件脱节的，所以我们先要来初步认识一下51芯片的结构和引脚功能。MSC51架构的芯片种类很多，具体特点和功能不尽相同（在以后编写的附录中会加入常用的一些51芯片的资料列表），在此后的教程中就以Atmel公司的AT89C51和AT89C2051为中心对象来进行学习，两者是AT89系列的典型代表，在爱好者中使用相当的多，应用资料很多，价格便宜，是初学51的首选芯片。嘿嘿，口水多多有点卖广告之嫌了。:P

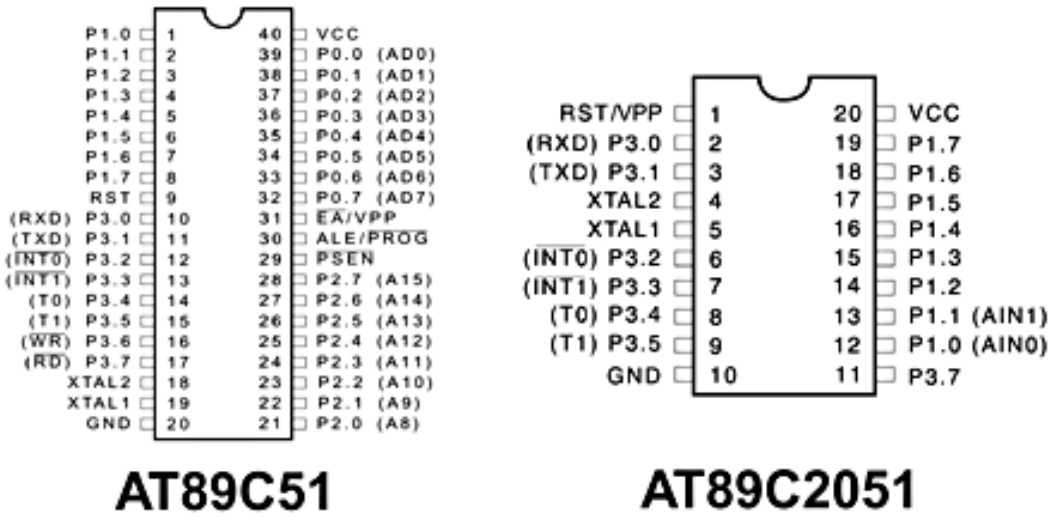


图2－1 AT89C51和AT89C2051引脚功能图

AT89C51	AT89C2051
4KB可编程Flash存储器（可擦写1000次）	2KB可编程Flash存储器（可擦写1000次）
三级程序存储器保密	两级程序存储器保密
静态工作频率:0Hz-24MHz	静态工作频率:0Hz-24MHz
128字节内部RAM	128字节内部RAM
2个16位定时/计数器	2个16位定时/计数器
一个串行通讯口	一个串行通讯口
6个中断源	6个中断源
32条I/O引线	15条I/O引线
片内时种振荡器	1个片内模拟比较器

表2－1 AT89C51和AT89C2051主要性能表

图2－1中是AT89C51和AT89C2051的引脚功能图。而表2－1中则是它们的主要性能表。以上可以看出它们是大体相同的,由于AT89C2051的IO线很少,导致它无法外加RAM和程序ROM,片内Flash存储器也少，但它的体积比AT89C51小很多，以后大家可根据实际需要来选用。它们各有其特点但其核心是一样的，下面就来看看AT89C51的引脚具体功能。

1.电源引脚

Vcc 40 电源端

GND 20 接地端

＊工作电压为5V，另有AT89LV51工作电压则是2.7-6V，引脚功能一样。

2.外接晶体引脚

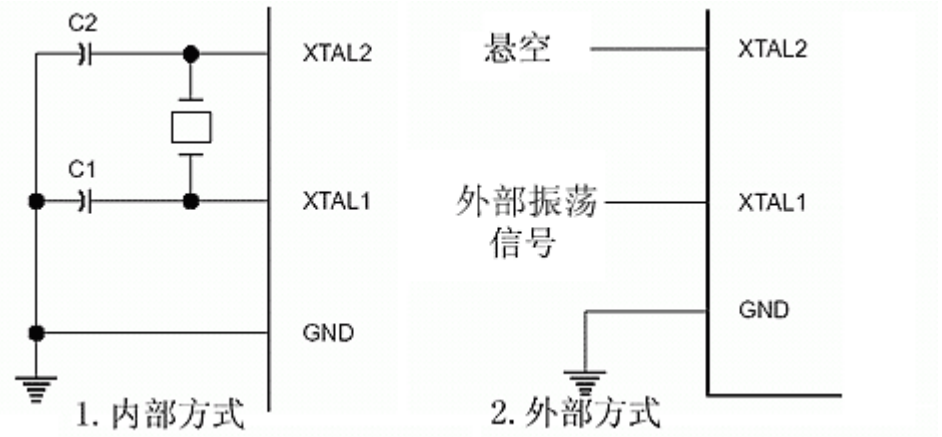


图2－2 外接晶体引脚

XTAL1 19

XTAL2 18

XTAL1是片内振荡器的反相放大器输入端，XTAL2则是输出端，使用外部振荡器时，外部振荡信号应直接加到XTAL1，而XTAL2悬空。内部方式时，时钟发生器对振荡脉冲二分频，如晶振为12MHz，时钟频率就为6MHz。晶振的频率可以在1MHz-24MHz内选择。电容取30PF左右。

＊型号同样为AT89C51的芯片，在其后面还有频率编号，有12,16,20,24MHz可选。大家在购买和选用时要注意了。如AT89C51 24PC就是最高振荡频率为24MHz,40P6封装的普通商用芯片。

3.复位 RST 9

在振荡器运行时，有两个机器周期（24个振荡周期）以上的高电平出现在此引腿时，将使单片机复位，只要这个脚保持高电平，51芯片便循环复位。复位后P0－P3口均置1引脚表现为高电平，程序计数器和特殊功能寄存器SFR全部清零。当复位脚由高电平变为低电平时，芯片为ROM的00H处开始运行程序。常用的复位电路如图2－3所示。

＊复位操作不会对内部RAM有所影响。

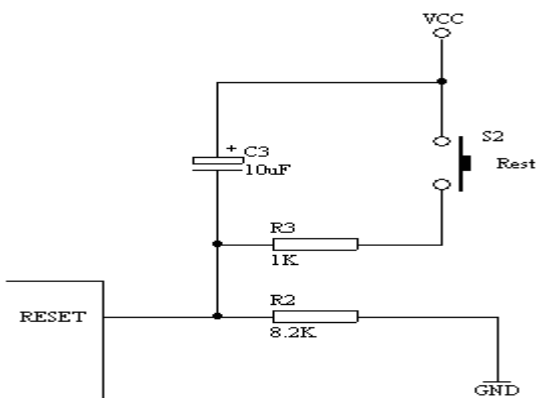


图2－3 常用复位电路

4.输入输出引脚

(1) P0端口[P0.0-P0.7] P0是一个8位漏极开路型双向I/O端口，端口置1（对端口写1）时作高阻抗输入端。作为输出口时能驱动8个TTL。

对内部Flash程序存储器编程时，接收指令字节;校验程序时输出指令字节，要求外接上拉电阻。

在访问外部程序和外部数据存储器时，P0口是分时转换的地址(低8位)/数据总线，访问期间内部的上拉电阻起作用。

(2) P1端口[P1.0－P1.7] P1是一个带有内部上拉电阻的8位双向I/0端口。输出时可驱动4个TTL。端口置1时，内部上拉电阻将端口拉到高电平，作输入用。

对内部Flash程序存储器编程时，接收低8位地址信息。

(3) P2端口[P2.0－P2.7] P2是一个带有内部上拉电阻的8位双向I/0端口。输出时可驱动4个TTL。端口置1时，内部上拉电阻将端口拉到高电平，作输入用。

对内部Flash程序存储器编程时，接收高8位地址和控制信息。

在访问外部程序和16位外部数据存储器时，P2口送出高8位地址。而在访问8位地址的外部数据存储器时其引脚上的内容在此期间不会改变。

(4) P3端口[P3.0－P3.7] P2是一个带有内部上拉电阻的8位双向I/0端口。输出时可驱动4个TTL。端口置1时，内部上拉电阻将端口拉到高电平，作输入用。

对内部Flash程序存储器编程时，接控制信息。除此之外P3端口还用于一些专门功能，具体请看 表2－2.。

＊P1－3端口在做输入使用时，因内部有上接电阻，被外部拉低的引脚会输出一定的电流。

P3引脚	兼用功能
P3.0	串行通讯输入（RXD）
P3.1	串行通讯输出（TXD）
P3.2	外部中断0（ INT0）
P3.3	外部中断1（INT1）
P3.4	定时器0输入(T0)
P3.5	定时器1输入(T1)
P3.6	外部数据存储器写选通WR
P3.7	外部数据存储器写选通RD

表2－2 P3端口引脚兼用功能表

呼！一口气说了那么多，停一下吧。嗯，什么？什么叫上拉电阻？上拉电阻简单来说就是把电平拉高，通常用4.7－10K的电阻接到Vcc电源，下拉电阻则是把电平拉低，电阻接到GND地线上。具体说明也不是这里要讨论的，接下来还是接着看其它的引脚功能吧。

5.其它的控制或复用引脚

(1) ALE/PROG 30 访问外部存储器时，ALE（地址锁存允许）的输出用于锁存地址的低位字节。即使不访问外部存储器，ALE端仍以不变的频率输出脉冲信号(此频率是振荡器频率的

1/6)。在访问外部数据存储器时，出现一个ALE脉冲。对Flash存储器编程时，这个引脚用于输入编程脉冲PROG

(2) PSEN 29 该引是外部程序存储器的选通信号输出端。当AT89C51由外部程序存储器取指令或常数时，每个机器周期输出2个脉冲即两次有效。但访问外部数据存储器时，将不会有脉冲输出。

(3) EA/Vpp 31 外部访问允许端。当该引脚访问外部程序存储器时，应输入低电平。要使AT89C51只访问外部程序存储器（地址为0000H-FFFFH）,这时该引脚必须保持低电平，而要使用片内的程序存储器时该引脚必须保持高电平。对Flash存储器编程时，该引脚用于施加Vpp编程电压。Vpp电压有两种，类似芯片最大频率值要根据附加的编号或芯片内的特征字决定。具体如表2－3所列。

	Vpp = 12V		Vpp = 5V	
印刷在芯片面上的型号	AT89C51 Xxxx YYWW	AT89LV51 Xxxx YYWW	AT89C51 xxxx-5 YYWW	AT89LV51 xxxx-5 YYWW
片内特征字	030H=1EH 031H=51H 032H=FFH	030H=1EH 031H=61H 032H=FFH	030H=1EH 031H=51H 032H=05H	030H=1EH 031H=61H 032H=05H

表2－3 Vpp与芯片型号和片内特征字的关系

看到这您对AT89C51引脚的功能应该有了一定的了解了，引脚在编程和校验时的时序我们在这里就不做详细的探讨，通常情况下我们也没有必要去撑握它，除非你想自己开发编程器。下来的课程我们要开始以一些简单的实例来讲述C程序的语法和编写方法技巧，中间穿插相关的硬件知识如串口，中断的用法等等。

第三课 生成HEX文件和最小化系统

在开始C语言的主要内容时，我们先来看看如何用KEIL uVISION2来编译生成用于烧写芯片的HEX文件。HEX文件格式是Intel公司提出的按地址排列的数据信息,数据宽度为字节,所有数据使用16进制数字表示, 常用来保存单片机或其他处理器的目标程序代码。它保存物理程序存储区中的目标代码映象。一般的编程器都支持这种格式。我们先来打开第一课做的第一项目，打开它的所在目录，找到test.Uv2的文件就可以打开先前的项目了。然后右击图3－1中的1项目文件夹，弹出项目功能菜单，选Options for Target'Target1',弹出项目选项设置窗口，同样先选中项目文件夹图标，这时在Project菜单中也有一样的菜单可选。打开项目选项窗口，转到Output选项页图3－2所示，图中1是选择编译输出的路径，2是设置编译输出生成的文件名，3则是决定是否要创建HEX文件，选中它就可以输出HEX文件到指定的路径中。选好了？好，我们再将它重新编译一次，很快在编译信息窗口中就显示HEX文件创建到指定的路径中了，如图3－3。这样我们就可用自己的编程器所附带的软件去读取并烧到芯片了，再用实验板看结果，至于编程器或仿真器品种繁多具体方法就看它的说明书了，这里也不做讨论。

（技巧：一、在图3－1中的1里的项目文件树形目录中，先选中对象，再单击它就可对它进行重

命名操作，双击文件图标便可打开文件。二、在Project下拉菜单的最下方有最近编辑过的项目路径保存，这里可以快速打开最近在编辑的项目。）

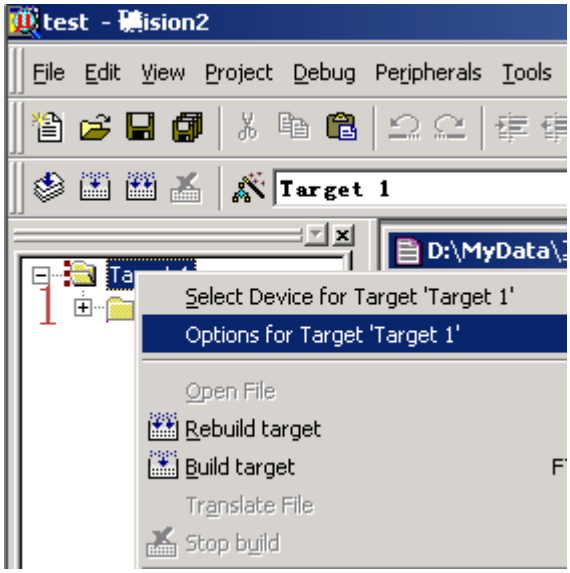


图3－1项目功能菜单

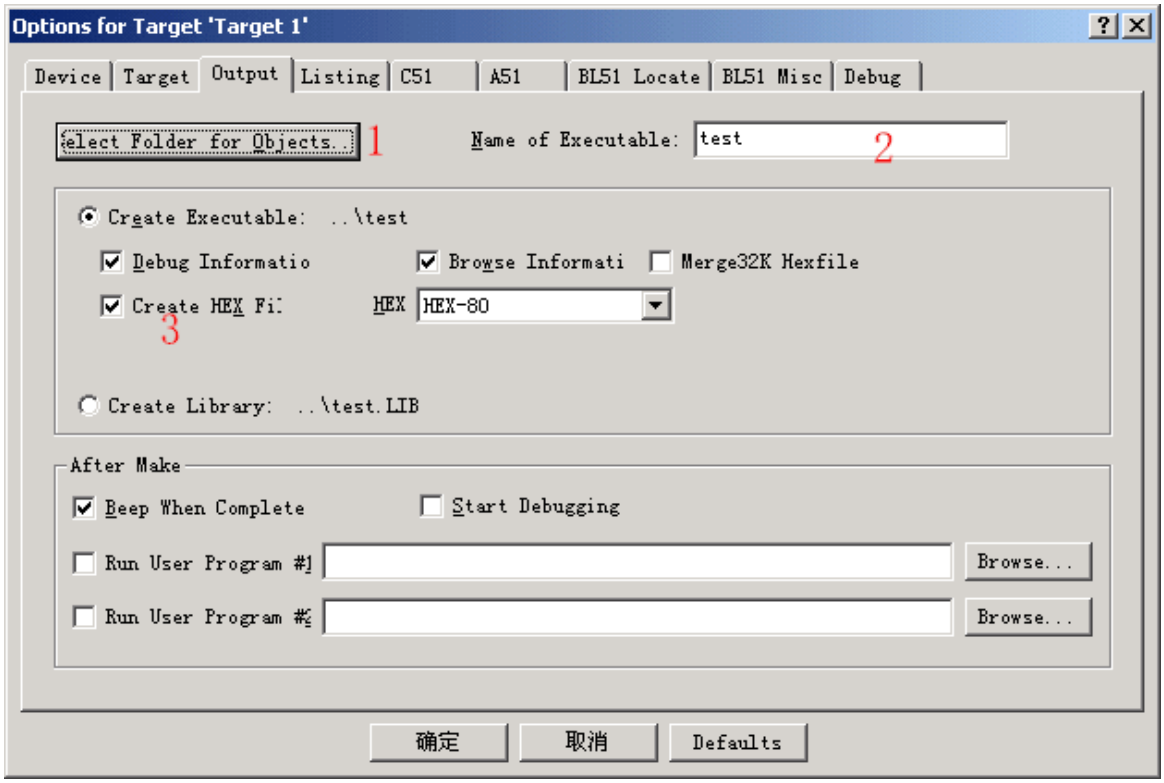


图3－2　项目选项窗口

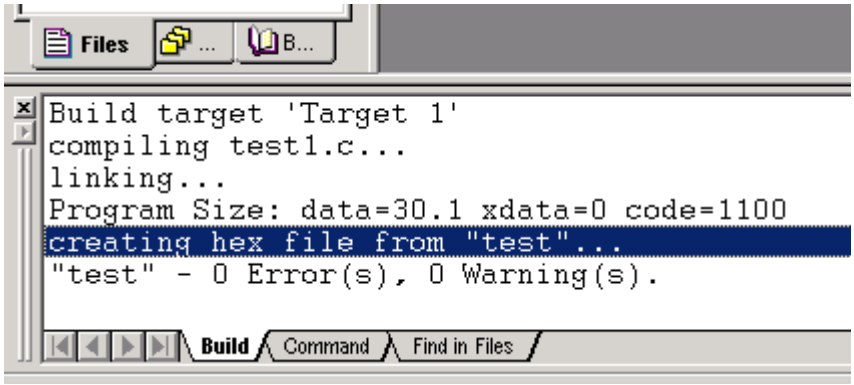


图3－3 编译信息窗口

或许您已把编译好的文件烧到了芯片上，如果您购买或自制了带串口输出元件的学习实验板，那您就可以把串口和PC机串口相联用串口调试软件或Windows的超级终端,将其波特率设为1200，就可以看到不停输出的"Hello World!"字样。也许您还没有实验板，那这里先说说AT89C51的最小化系统，再以一实例程序验证最小化系统是否在运行，这个最小化系统也易于自制用于实验。图3－4便是AT89C51的最小化系统,不过为了让我们可以看出它是在运行的，我加了一个电阻和一个LED，用以显示它的状态，晶振可以根据自己的情况使用，一般实验板上是用11.0592MHz或12MHz,使用前者的好外是可以产生标准的串口波特率，后者则一个机器周期为1微秒，便于做精确定时。在自己做实验里，注意的是VCC是+5V的，不能高于此值，否则将损坏单片机，太低则不能正常工作。在31(EA)脚要接高电平，这样我们才能执行片内的程序，如接低电平则使用片外的程序存储器。下面，我们建一个新的项目名为OneLED来验证最小化系统是否可以工作（所有的例程都可在我的主页下面下载到，网址：http://cdle.yeah.net 或 http://www.cdle.net ）。程序如下：

```
#include <AT89X51.h> //预处理命令

void main(void) //主函数名

{
//这是第一种注释方式
unsigned int a; //定义变量a为int类型
/*这是第二种注释方式*/
do{//do while组成循环
for (a=0; a<50000; a++); //这是一个循环
P1_0 = 0; //设P1.0口为低电平，点亮LED
for (a=0; a<50000; a++); //这是一个循环
P1_0 = 1; //设P1.0口为高电平，熄灭LED
}
while(1);

}
```

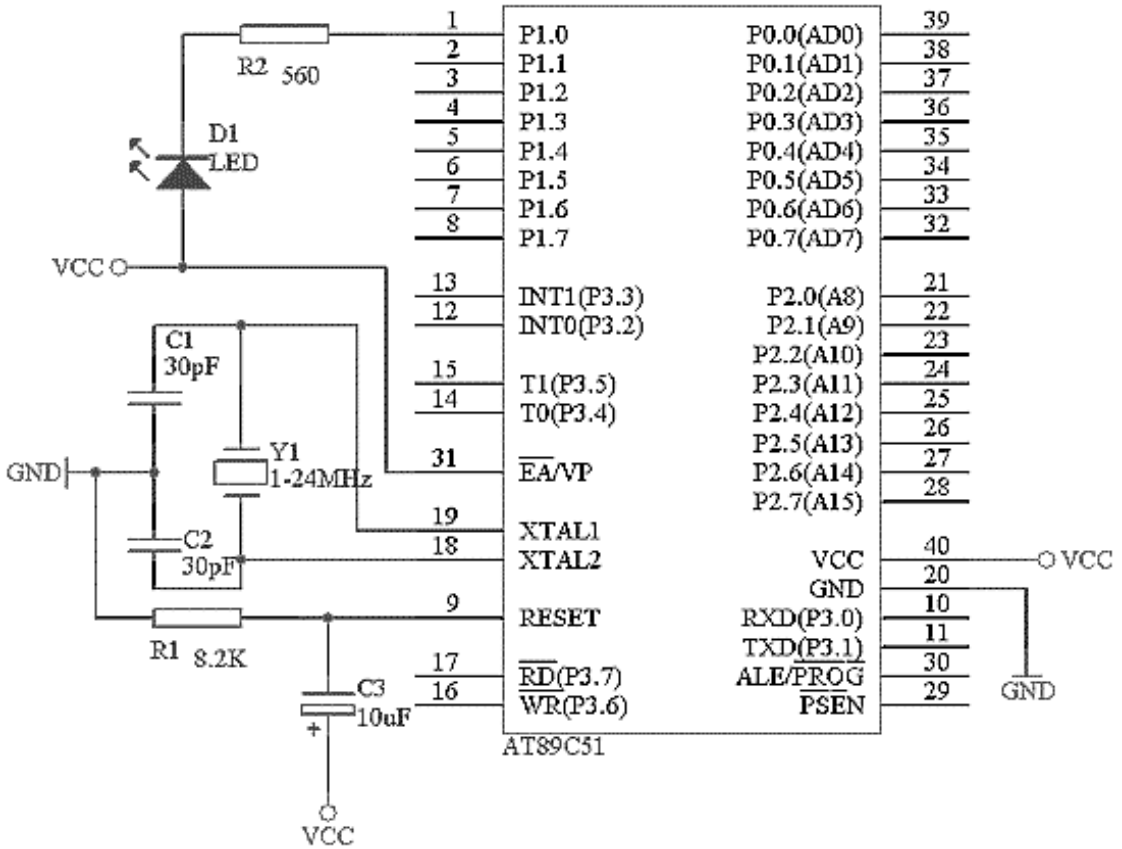


图3－4　AT89C51最小化系统

这里先讲讲 KEIL C 编译器所支持的注释语句。一种是以"//"符号开始的语句，符号之后的语句都被视为注释，直到有回车换行。另一种是在"/*"和"*/"符号之内的为注释。注释不会被 C 编译器所编译。一个 C 应用程序中应有一个 main 主函数，main 函数可以调用别的功能函数，但其它功能函数不允许调用 main 函数。不论 main 函数放在程序中的那个位置，总是先被执行。用上面学到的知识编译写好的 OneLED 程序，并把它烧到刚做好的最小化系统中。上电，刚开始时 LED 是不亮的（因为上电复位后所有的 IO 口都置 1 引脚为高电平），然后延时一段时间（for (a=0; a<50000; a++)这句在运行），LED 亮，再延时，LED 熄灭，然后交替亮、灭。第一个真正的小应用就做完，呵呵，先不要管它是否实用哦。如果没有这样的效果那么您就要认真检查一下电路或编译烧写的步骤了。

第四课　数据类型

先来简单说说C语言的标识符和关键字。标识符是用来标识源程序中某个对象的名字的，这些对象可以是语句、数据类型、函数、变量、数组等等。C语言是大小字敏感的一种高级语言，如果我们要定义一个定时器1，可以写做"Timer1"，如果程序中有"TIMER1"，那么这两个是完全不同定义的标识符。标识符由字符串，数字和下划线等组成，注意的是第一个字符必须是字母或下划线，如"1Timer"是错误的，编译时便会有错误提示。有些编译系统专用的标识符是以下划线开头，所以一般不要以下划线开头命名标识符。标识符在命名时应当简单，含义清晰，这样有助

于阅读理解程序。在C51编译器中，只支持标识符的前32位为有效标识，一般情况下也足够用了，除非你要写天书：P。

关键字则是编程语言保留的特殊标识符，它们具有固定名称和含义，在程序编写中不允许标识符与关键字亦同。在KEIL uVision2中的关键字除了有ANSI C标准的32个关键字外还根据51单片机的特点扩展了相关的关键字。其实在KEIL uVision2的文本编辑器中编写C程序，系统可以把保留字以不同颜色显示，缺省颜色为天蓝色。(标准和扩展关键字请看附录一中的附表1-1和附表1-2)

先看表4－1，表中列出了KEIL uVision2 C51编译器所支持的数据类型。在标准C语言中基本的数据类型为char,int,short,long,float和double，而在C51编译器中int和short相同，float和double相同，这里就不列出说明了。下面来看看它们的具体定义：

数据类型	长　度	值　域
unsigned char	单字节	0～255
signed char	单字节	-128～+127
unsigned int	双字节	0～65535
signed int	双字节	-32768～+32767
unsigned long	四字节	0～4294967295
signed long	四字节	-2147483648～+2147483647
float	四字节	±1.175494E-38～±3.402823E+38
*	1～3字节	对象的地址
bit	位	0或1
sfr	单字节	0～255
sfr16	双字节	0～65535
sbit	位	0或1

表4－1　KEIL uVision2 C51编译器所支持的数据类型

1. char字符类型

char类型的长度是一个字节(8位)，通常用于定义处理字符数据的变量或常量。分无符号字符类型unsigned char和有符号字符类型signed char，默认值为signed char类型。unsigned char类型用字节中所有的位来表示数值，所可以表达的数值范围是0～255。signed char类型用字节中最高位字节表示数据的符号，"0"表示正数，"1"表示负数，负数用补码表示。所能表示的数值范围是-128～+127。unsigned char常用于处理ASCII字符或用于处理小于或等于255的整型数。

＊正数的补码与原码相同，负二进制数的补码等于它的绝对值按位取反后加1。

2. int整型

int整型长度为两个字节(16位)，用于存放一个双字节数据。分有符号int整型数signed int和无符号整型数unsigned int，默认值为signed int类型。signed int表示的数值范围是-32768～+32767，字节中最高位表示数据的符号，"0"表示正数，"1"表示负数。unsigned int表示的数值范围是0～65535。好了，先停一下吧，我们来写个小程序看看unsigned char和unsigned int用于延时的不同效果，说明它们的长度是不同的，呵，尽管它并没有实际的应用意义，这里我们学习它们的用

法就行。依旧用我们上一课的最小化系统做实验，不过要加多一个电阻和LED，如图4－1。实验中用D1的点亮表明正在用unsigned int数值延时，用D2点亮表明正在用unsigned char数值延时。

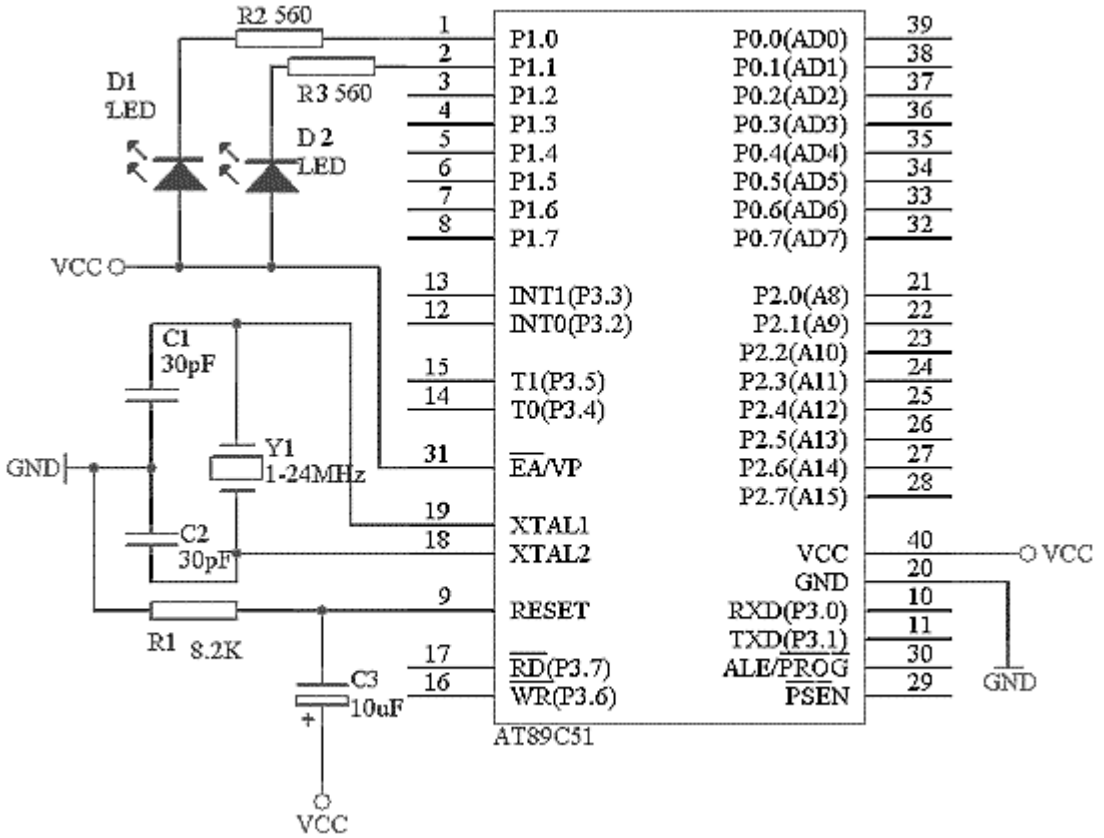


图4－1　第4课实验用电路

我们把这个项目称为TwoLED,实验程序如下：

```
#include //预处理命令

void main(void) //主函数名

{
unsigned int a; //定义变量a为unsigned int类型
unsigned char b; //定义变量b为unsigned char类型do
{ //do while组成循环
for (a=0; a<65535; a++)
P1_0 = 0; //65535次设P1.0口为低电平，点亮LED
P1_0 = 1; //设P1.0口为高电平，熄灭LED
for (a=0; a<30000; a++); //空循环
for (b=0; b<255; b++)
P1_1 = 0; //255次设P1.1口为低电平，点亮LED
P1_1 = 1; //设P1.1口为高电平，熄灭LED
```

```
for (a=0; a<30000; a++); //空循环
}
while(1);
}
```

同样编译烧写，上电运行您就可以看到结果了。很明显D1点亮的时间长于D2点亮的时间。程序中的循环延时时间并不是很好确定，并不太适合要求精确延时的场合，关于这方面我们以后也会做讨论。这里必须要讲的是，当定义一个变量为特定的数据类型时，在程序使用该变量不应使它的值超过数据类型的值域。如本例中的变量b不能赋超出0～255的值，如for (b=0; b<255; b++)改为for (b=0; b<256; b++),编译是可以通过的，但运行时就会有问题出现，就是说b的值永远都是小于256的，所以无法跳出循环执行下一句P1_1 = 1，从而造成死循环。同理a的值不应超出0～65535。大家可以烧片看看实验的运行结果，同样软件仿真也是可以看到结果的。

3．long长整型

long长整型长度为四个字节(32位)，用于存放一个四字节数据。分有符号long长整型signed long和无符号长整型unsigned long，默认值为signed long类型。signed int表示的数值范围是-2147483648～+2147483647，字节中最高位表示数据的符号，"0"表示正数，"1"表示负数。unsigned long表示的数值范围是0～4294967295。

4．float浮点型

float浮点型在十进制中具有7位有效数字，是符合IEEE－754标准的单精度浮点型数据，占用四个字节。因浮点数的结构较复杂在以后的章节中再做详细的讨论。

5．* 指针型

指针型本身就是一个变量，在这个变量中存放的指向另一个数据的地址。这个指针变量要占据一定的内存单元，对不同的处理器长度也不尽相同，在C51中它的长度一般为1～3个字节。指针变量也具有类型，在以后的课程中有专门一课做探讨，这里就不多说了。

6．bit位标量

bit位标量是C51编译器的一种扩充数据类型，利用它可定义一个位标量，但不能定义位指针，也不能定义位数组。它的值是一个二进制位，不是0就是1，类似一些高级语言中的Boolean类型中的True和False。

7．sfr特殊功能寄存器

sfr也是一种扩充数据类型，占用一个内存单元（8位），值域为0～255。利用它可以访问51单片机内部的所有特殊功能寄存器。如用sfr P1 = 0x90这一句定P1（工作寄存器）为P1端口在片内的寄存器，在后面的语句中我们可以用P1 = 255（对P1端口的所有引脚置高电平）之类的语句来操作特殊功能寄存器。

＊AT89C51的特殊功能寄存器表请看附录二

8．sfr16 16位特殊功能寄存器

sfr16占用两个内存单元（16位），值域为0～65535。sfr16和sfr一样用于操作特殊功能寄存器，所不同的是它用于操作占两个字节的寄存器，好定时器T0和T1。

9． sbit可寻址位

sbit同位是C51中的一种扩充数据类型，利用它可以访问芯片内部的RAM中的可寻址位或特殊功能寄存器中的可寻址位。如先前我们定义了

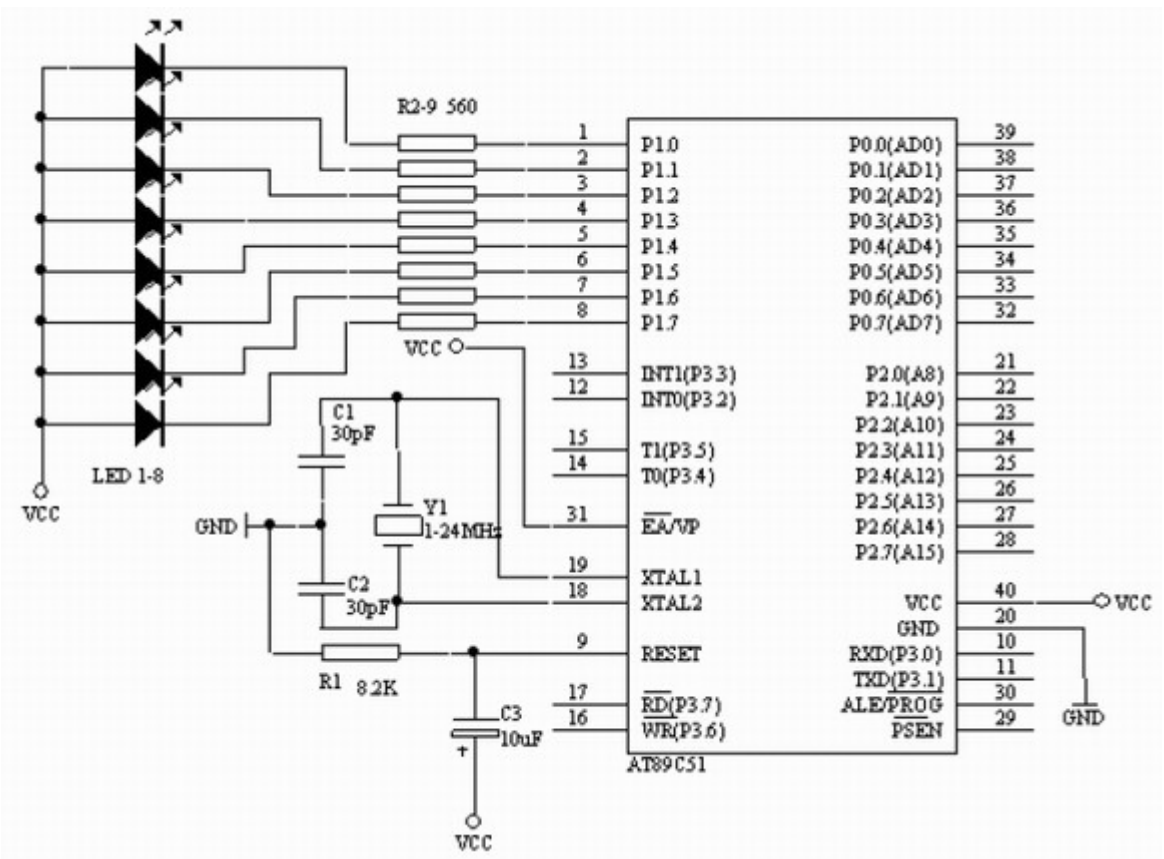
```
sfr P1 = 0x90; //因P1端口的寄存器是可位寻址的，所以我们可以定义
```

sbit P1_1 = P1 ^ 1; //P1_1为P1中的P1.1引脚
//同样我们可以用P1.1的地址去写,如sbit P1_1 = 0x91;
这样我们在以后的程序语句中就可以用 P1_1 来对 P1.1 引脚进行读写操作了。通常这些可以直接使用系统提供的预处理文件，里面已定义好各特殊功能寄存器的简单名字，直接引用可以省去一点时间，我自己是一直用的。当然您也可以自己写自己的定义文件，用您认为好记的名字。关于数据类型转换等相关操作在后面的课程或程序实例中将有所提及。大家可以用所讲到的数据类型改写一下这课的实例程序，加深对各类型的认识。

以上就是Keil 51中常用的数据类型，下面我们来看一个跑马灯的程序，加深了解一下C51的程序结构。

```
#include <AT89X51.H> //预处理文件里面定义了特殊寄存器的名称，如P1口定义为P1
void main(void)
{
//定义花样数据
const unsigned char design[32]={0xFF, 0xFE, 0xFD, 0xFB, 0xF7, 0xEF, 0xDF, 0xBF, 0x7F, 0x7F, 0xBF, 0xDF, 0xEF, 0xF7, 0xFB, 0xFD, 0xFE, 0xFF, 0xFF, 0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x0, 0xE7, 0xDB, 0xBD, 0x7E, 0xFF };
unsigned int a; //定义循环用的变量
unsigned char b; //在C51编程中因内存有限尽可能注意变量类型的使用尽可能使用少字节的类型，在大型的程序中很受用
do{
for (b=0; b<32; b++)
{
for(a=0; a<30000; a++); //延时一段时间
P1 = design[b]; //读已定义的花样数据并写花样数据到P1口
}
}while(1);
}
```

对应硬件电路图如下：



程序中的花样数据可以自己去定义，因这里我们的LED要AT89C51的P1引脚为低电平才会点亮，所以我们要向P1口的各引脚写数据0，对应连接的LED才会被点亮，P1口的八个引脚刚好对应P1口特殊寄存器的八个二进位，如向P1口定数据0xFE，转成二进制就是11111110，最低位D0为0，这里P1.0引脚输出低电平，LED1被点亮。如此类推，大家不难算出自己想要做的效果了。大家编译烧写看看，效果就出来，显示的速度您可以根据需要调整延时a的值，不要超过变量类型的值域就行了。如果你还没有开发板，或者连最小系统板也没自己焊一块，也没关系，还记得Keil的I/O口仿真功能吗？看看这里就知道该怎么办了。

回到程序中来，第一句的#include跟C语言里面的引用是一样的，这个头文件包含了程序中没有声明的变量P1，所以P1可以直接使用不会出错啦。接下来程序直接跳转到main函数执行，do-while循环保证单片机一直循环工作。

下面我们把程序换一种方式写，以加深对寄存器的理解。 、

```
sfr P1 = 0x90; //这里没有使用预定义文件，而是自己定义特殊寄存器，之前我们使用的预定义文件其实就是这个作用

sbit P1_0 = P1^0;
sbit P1_7 = 0x90^7;
sbit P1_1 = 0x91; //这里分别定义P1端口和P1.0，P1.1，P1.7引脚
```

```
void main(void)
{
unsigned int a;
unsigned char b;
do{
for (a=0;a<50000;a++)
P1_0 = 0; //点亮P1_0
for (a=0;a<50000;a++)
P1_7 = 0; //点亮P1_7
for (b=0;b<255;b++)
{
for (a=0;a<10000;a++)
P1 = b; //用b的值来做跑马灯的花样
}
P1 = 255; //熄灭P1上的LED
for (b=0;b<255;b++)
{
for (a=0;a<10000;a++) //P1_1闪烁
P1_1 = 0;
for (a=0;a<10000;a++)
P1_1 = 1;
}
}while(1);
}
```

到这里，你应该对单片机编程有了一个基本的概念，其实单片机C程序跟PC机上面没有什么大的区别，只要弄清楚单片机特有的寄存器功能，编写单片机程序将是一件很轻松的事情。

第五课　常量

上一节我们学习了KEIL C51编译器所支持的数据类型。而这些数据类型又是怎么用在常量和变量的定义中的呢？又有什么要注意的吗？下面就来看看吧。晕！你还区分不清楚什么是常量，什么是变量。常量是在程序运行过程中不能改变值的量，而变量是可以在程序运行过程中不断变化的量。变量的定义可以使用所有C51编译器支持的数据类型，而常量的数据类型只有整型、浮点型、字符型、字符串型和位标量。这一节我们学习常量定义和用法，而下一节则学习变量。常量的数据类型说明是这样的

1． 整型常量可以表示为十进制如123,0，－89等。十六进制则以0x开头如0x34,-0x3B等。长整型就在数字后面加字母L，如104L，034L，0xF340等。

2． 浮点型常量可分为十进制和指数表示形式。十进制由数字和小数点组成，如0.888,3345.345,0.0等，整数或小数部分为0，可以省略但必须有小数点。指数表示形式为[±]数字[.数字]e[±]数字，[ ]中的内容为可选项，其中内容根据具体情况可有可无，但其余部分必须有,如125e3,7e9,-3.0e-3。

3． 字符型常量是单引号内的字符，如'a','d'等，不可以显示的控制字符，可以在该字符前面加一个反斜杠"\"组成专用转义字符。常用转义字符表请看表5－1。

4． 字符串型常量由双引号内的字符组成，如"test","OK"等。当引号内的没有字符时，为空字符串。在使用特殊字符时同样要使用转义字符如双引号。在C中字符串常量是做为字符类型数组来处理的，在存储字符串时系统会在字符串尾部加上\o转义字符以作为该字符串的结束符。字符串常量"A"和字符常量'A'是不同的，前者在存储时多占用一个字节的字间。

5． 位标量，它的值是一个二进制。

转义字符	含义	ASCII码（16/10进制）
\o	空字符(NULL)	00H/0
\n	换行符(LF)	0AH/10
\r	回车符(CR)	0DH/13
\t	水平制表符(HT)	09H/9
\b	退格符(BS)	08H/8
\f	换页符(FF)	0CH/12
\'	单引号	27H/39
\"	双引号	22H/34
\\	反斜杠	5CH/92

表5－1 常用转义字符表

常量可用在不必改变值的场合，如固定的数据表，字库等。常量的定义方式有几种,下面来加以说明。

```
#difine False 0x0; //用预定义语句可以定义常量
#difine True 0x1; //这里定义False为0,True为1
                  //在程序中用到False编译时自动用0替换，同理True替换为1
unsigned int code a=100; //这一句用code把a定义在程序存储器中并赋值
const unsigned int c=100; //用const定义c为无符号int常量并赋值
```

以上两句它们的值都保存在程序存储器中，而程序存储器在运行中是不允许被修改的，所以如果在这两句后面用了类似a=110，a++这样的赋值语句，编译时将会出错。

说了一通还不如写个程序来实验一下吧。写什么程序呢？跑马灯！对，就写这个简单易懂的吧，这个也好说明典型的常量用法。先来看看电路图吧。它是在我们上一课的实验电路的基础上增加6个LED组成的，也就是用P1口的全部引脚分别驱动一个LED，电路如图5－1所示。

新建一个RunLED的项目，主程序如下：

```
#include <AT89X51.H> //预处理文件里面定义了特殊寄存器的名称如P1口定义为P1
void main(void)
{
//定义花样数据
```

```
const unsigned char design[32]={0xFF,0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,
0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,0xFF,
0xFF,0xFE,0xFC,0xF8,0xF0,0xE0,0xC0,0x80,0x0,
0xE7,0xDB,0xBD,0x7E,0xFF};
unsigned int a; //定义循环用的变量
unsigned char b; //在C51编程中因内存有限尽可能注意变量类型的使用
//尽可能使用少字节的类型，在大型的程序中很受用
do{
for (b=0; b<32; b++)
{
for(a=0; a<30000; a++); //延时一段时间
P1 = design[b]; //读已定义的花样数据并写花样数据到P1口
}
}while(1);
}
```

程序中的花样数据可以自以去定义,因这里我们的LED要AT89C51的P1引脚为低电平才会点亮，所以我们要向P1口的各引脚写数据O对应连接的LED才会被点亮，P1口的八个引脚刚好对应P1口特殊寄存器的八个二进位，如向P1口定数据0xFE,转成二进制就是11111110,最低位D0为0这里P1.0引脚输出低电平，LED1被点亮。如此类推，大家不难算出自己想要做的效果了。大家编译烧写看看，效果就出来，显示的速度您可以根据需要调整延时a的值，不要超过变量类型的值域就很行了。哦，您还没有实验板？那如何可以知道程序运行的结果呢？呵，不用急，这就来说说用KEIL uVision2的软件仿真来调试IO口输出输入程序。

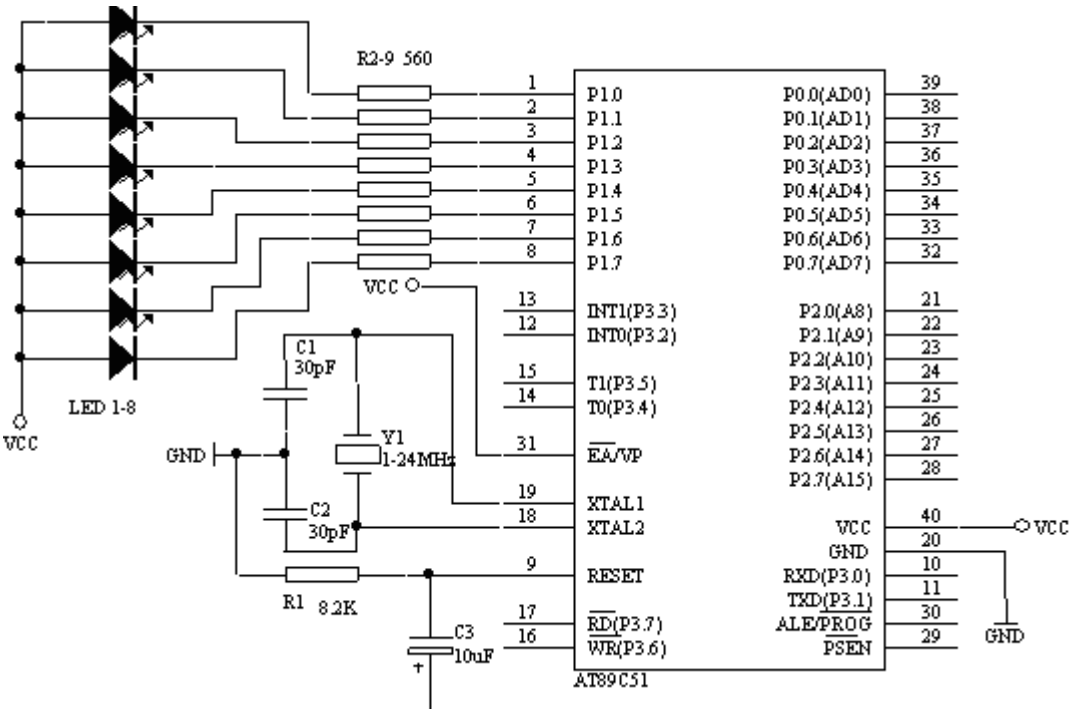


图5－1　八路跑马灯电路

编译运行上面的程序，然后按外部设备菜单Peripherals－I/O Ports－Port1就打开Port1的调试窗口了，如图5－3中的2。这时程序运行了，但我们并不能在Port1调试窗口上看到有会什么效果，这时我们可以用鼠标左击图5－3中1旁边绿色的方条，点一下就有一个小红方格在点一下又没有了，哪一句语句前有小方格程序运行到那一句时就停止了，就是设置调试断点，同样图5－2中的1也是同样功能，分别是增加/移除断点、移除所有断点、允许/禁止断点、禁止所有断点，菜单也有一样的功能，另外菜单中还有Breakpoints可打开断点设置窗口它的功能更强大，不过我们这里先不用它。我们?quot;P1 = design[b];"这一句设置一个断点这时程序运行到这里就停住了，再留意一下Port1调试窗口，再按图5-2中的2的运行键，程序又运行到设置断点的地方停住了，这时Port1调试窗口的状态又不同了。也就是说Port1调试窗口模拟了P1口的电平状态，打勾为高电平，不打勾则为低电平，窗口中P1为P1寄存器的状态，Pins为引脚的状态，注意的是如果是读引脚值必须把引脚对应的寄存器置1才能正确读取。图5－2中2旁边的{ }样的按钮分别为单步入，步越，步出和执行到当前行。图中3为显示下一句将要执行的语句。图5－3中的3是Watches窗口可查看各变量的当前值，数组和字串是显示其头一个地址，如本例中的design数组是保存在RAM存储区的首地址为D:0x08,可以在图中4 Memory存储器查看窗口中的Address地址中打入D:0x08就可以查看到design各数据和存放地址了。如果你的uVision2没有显示这些窗口，可以在View菜单中打开在图5－2中3后面一栏的查看窗口快捷栏中打开。

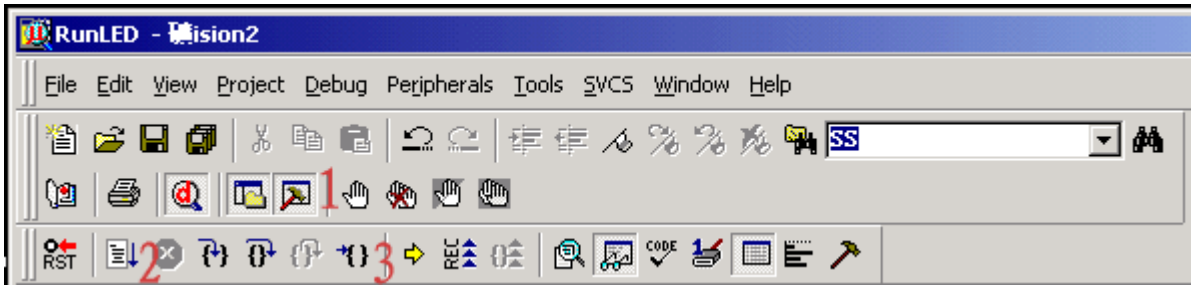


图5－2 调试用快捷菜单栏

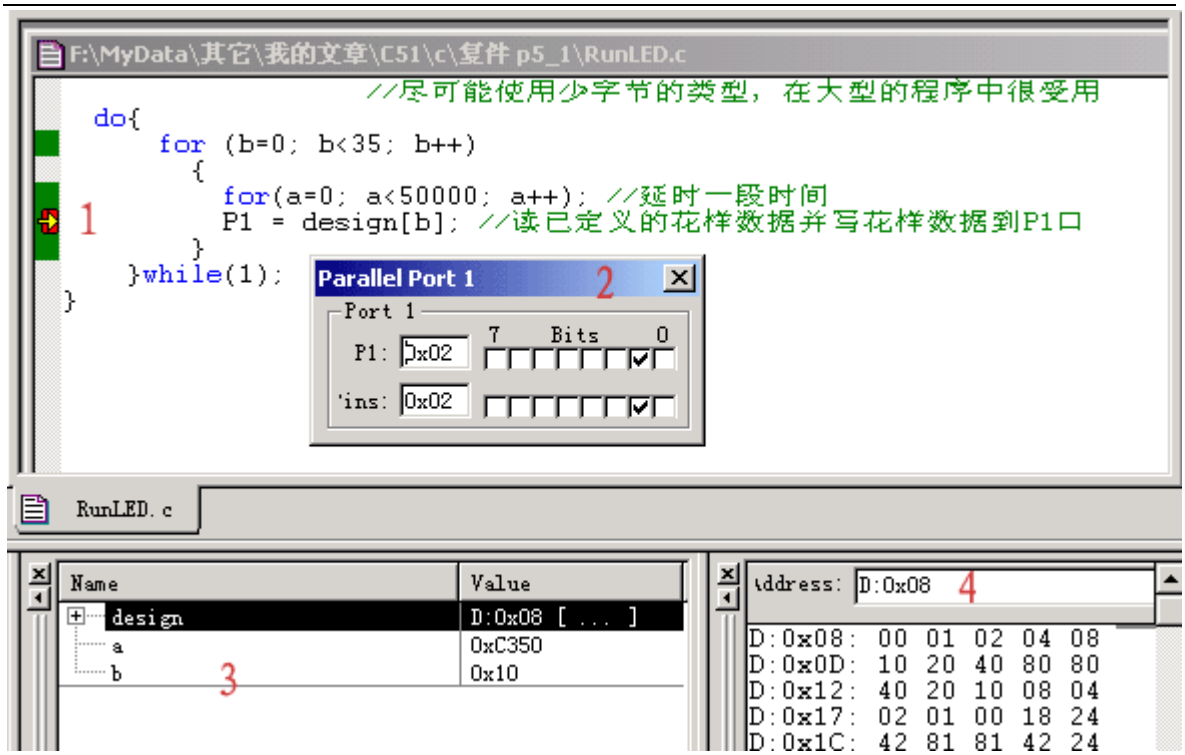


图 5－3 各调试窗口

第六课 变量

上课所提到变量就是一种在程序执行过程中其值能不断变化的量。要在程序中使用变量必须先用标识符作为变量名，并指出所用的数据类型和存储模式，这样编译系统才能为变量分配相应的存储空间。定义一个变量的格式如下：

[存储种类] 数据类型 [存储器类型] 变量名表

在定义格式中除了数据类型和变量名表是必要的，其它都是可选项。存储种类有四种：自动（auto）,外部（extern）,静态（static）和寄存器（register），缺省类型为自动(auto)。

而这里的数据类型则是和我们在第四课中学习到的名种数据类型的定义是一样的。说明了一个变量的数据类型后，还可选择说明该变量的存储器类型。存储器类型的说明就是指定该变量在C51硬件系统中所使用的存储区域，并在编译时准确的定位。表6－1中是KEIL uVision2所能认别的存储器类型。注意的是在AT89C51芯片中RAM只有低128位，位于80H到FFH的高128位则在52芯片中才有用，并和特殊寄存器地址重叠。特殊寄存器（SFR）的地址表请看附录二 AT89C51特殊功能寄存器列表

存储器类型	说 明
data	直接访问内部数据存储器（128字节),访问速度最快
bdata	可位寻址内部数据存储器（16字节），允许位与字节混合访问
idata	间接访问内部数据存储器（256字节），允许访问全部内部地址
pdata	分页访问外部数据存储器（256字节），用MOVX @Ri指令访问
xdata	外部数据存储器(64KB)，用MOVX @DPTR指令访问
code	程序存储器（64KB）,用MOVC @A+DPTR指令访问

表6－1 存储器类型

如果省略存储器类型，系统则会按编译模式SMALL,COMPACT或LARGE所规定的默认存储器类型去指定变量的存储区域。无论什么存储模式都可以声明变量在任何的8051存储区范围，然而把最常用的命令如循环计数器和队列索引放在内部数据区可以显著的提高系统性能。还有要指出的就是变量的存储种类与存储器类型是完全无关的。

SMALL存储模式把所有函数变量和局部数据段放在8051系统的内部数据存储区这使访问数据非常快，但SMALL存储模式的地址空间受限。在写小型的应用程序时，变量和数据放在data内部数据存储器中是很好的因为访问速度快，但在较大的应用程序中data区最好只存放小的变量、数据或常用的变量（如循环计数、数据索引），而大的数据则放置在别的存储区域。

COMPACT存储模式中所有的函数和程序变量和局部数据段定位在8051系统的外部数据存储区。外部数据存储区可有最多256字节(一页),在本模式中外部数据存储区的短地址用@R0/R1。

LARGE存储模式所有函数和过程的变量和局部数据段都定位在8051系统的外部数据区外部数据区最多可有64KB，这要求用DPTR数据指针访问数据。

之前提到简单提到sfr,sfr16,sbit定义变量的方法，下面我们再来仔细看看。

sfr和sfr16可以直接对51单片机的特殊寄存器进行定义，定义方法如下：

```
sfr  特殊功能寄存器名=  特殊功能寄存器地址常数;
sfr16  特殊功能寄存器名=  特殊功能寄存器地址常数;
```

我们可以这样定义AT89C51的P1口

```
sfr P1 = 0x90; //定义P1 I/O口，其地址90H
```

sfr关键定后面是一个要定义的名字，可任意选取，但要符合标识符的命名规则，名字最好有一定的含义如P1口可以用P1为名，这样程序会变的好读好多。等号后面必须是常数，不允许有带运算符的表达式，而且该常数必须在特殊功能寄存器的地址范围之内（80H－FFH），具体可查看附录中的相关表。sfr是定义8位的特殊功能寄存器而sfr16则是用来定义16位特殊功能寄存器，如8052的T2定时器，可以定义为：

```
sfr16 T2 = 0xCC; //这里定义8052定时器2，地址为T2L=CCH,T2H=CDH
```

用sfr16定义16位特殊功能寄存器时，等号后面是它的低位地址，高位地址一定要位于物理低位地址之上。注意的是不能用于定时器0和1的定义。

sbit可定义可位寻址对象。如访问特殊功能寄存器中的某位。其实这样应用是经常要用的如要访问P1口中的第2个引脚P1.1。我们可以照以下的方法去定义:

(1)sbit 位变量名＝位地址

```
sbit P1_1 = Ox91;
```

这样是把位的绝对地址赋给位变量。同sfr一样sbit的位地址必须位于80H-FFH之间。

(2)Sbit 位变量名＝特殊功能寄存器名^位位置

```
sft P1 = 0x90;
sbit P1_1 = P1 ^ 1; //先定义一个特殊功能寄存器名再指定位变量名所在的位置
```

当可寻址位位于特殊功能寄存器中时可采用这种方法

(3)sbit 位变量名＝字节地址^位位置

```
sbit P1_1 = 0x90 ^ 1;
```

这种方法其实和2是一样的，只是把特殊功能寄存器的位址直接用常数表示。

在C51存储器类型中提供有一个bdata的存储器类型，这个是指可位寻址的数据存储器，位于单片机的可位寻址区中，可以将要求可位录址的数据定义为bdata,如:

unsigned char bdata ib; //在可位录址区定义ucsigned char类型的变量ib
int bdata ab[2]; //在可位寻址区定义数组ab[2]，这些也称为可寻址位对象
sbit ib7=ib^7 //用关键字sbit定义位变量来独立访问可寻址位对象的其中一位
sbit ab12=ab[1]^12;

操作符" ^ "后面的位位置的最大值取决于指定的基址类型，char0-7,int0-15,long0-31。

下面我们用上一课的电路来实践一下这一课的知识。同样是做一下简单的跑马灯实验，项目名为RunLED2。程序如下:

```
sfr P1 = 0x90; //这里没有使用预定义文件，
sbit P1_0 = P1 ^ 0; //而是自己定义特殊寄存器
sbit P1_7 = 0x90 ^ 7; //之前我们使用的预定义文件其实就是这个作用
sbit P1_1 = 0x91; //这里分别定义P1端口和P10,P11,P17引脚void main(void)
{
unsigned int a;
unsigned char b;
do{
for (a=0;a<50000;a++)
P1_0 = 0; //点亮P1_0
for (a=0;a<50000;a++)
P1_7 = 0; //点亮P1_7
for (b=0;b<255;b++)
{
for (a=0;a<10000;a++)
P1 = b; //用b的值来做跑马灯的花样
}
P1 = 255; //熄灭P1上的LED
for (b=0;b<255;b++)
{
for (a=0;a<10000;a++) //P1_1闪烁
P1_1 = 0;
for (a=0;a<10000;a++)
P1_1 = 1;
}
}while(1);
}
```

第七课 运算符和表达式（1）

上课到这一课相隔了好长一段时间，这些日子里收到不少网友的来信支持和鼓励，要求尽快完成余下的部分。出门在外的人不得不先为吃饭而努力，似乎这也成为我的借口，以后每晚抽空打一些吧这样大家也就可以不用隔太久就能看到一些新东西。或许我的笔记并不是很正确，但我尽量的保证每课的实验都会亲自做一次，包括硬件的部分,已求不会误人子弟。
随着访问量不断的增加，网站已启用了www.cdle.net的国际域名,在这里我感谢各位一直支持磁动力工作室的朋友，更要感激身在远方一直默默支持我的女友。 明浩 2003-7-14 晚

呵，费话少说了。上两课说了常量和变量,先来补充一个用以重新定义数据类型的的语句吧。这个语句就是typedef，这是个很好用的语句，但我自己却不常用它，通常我定义变量的数据类型时都是使用标准的关键字，这样别人可以很方便的研读你的程序。如果你是个DELPHI编程爱好者或是程序员，你对变量的定义也许习惯了DELPHI的关键字，如int类型常会用关键字Integer来定义，在用C51时你还想用回这个的话，你可以这样写：

```
typedef int integer;
integer a,b;
```

这两句在编译时，其实是先把integer定义为int，在以后的语句中遇到integer就用int置换,integer就等于int,所以a,b也就被定义为int。typedef不能直接用来定义变量，它只是对已有的数据类型作一个名字上的置换，并不是产生一个新的数据类型。下面两句就是一个错误的例子：

```
typedef int integer;
integer = 100;
```

使用typedef可以有方便程序的移植和简化较长的数据类型定义。用typedef还可以定义结构类型，这一点在后面详细解说结构类型时再一并说明。typedef的语法是

```
typedef  已有的数据类型  新的数据类型名
```

运算符就是完成某种特定运算的符号。运算符按其表达式中与运算符的关系可分为单目运算符，双目运算符和三目运算符。单目就是指需要有一个运算对象，双目就要求有两个运算对象，三目则要三个运算对象。表达式则是由运算及运算对象所组成的具有特定含义的式子。C是一种表达式语言，表达式后面加"；"号就构成了一个表达式语句。

赋值运算符

对于"="这个符号大家不会陌生的，在C中它的功能是给变量赋值，称之为赋值运算符。它的作用不用多说大家也明白，就是但数据赋给变量。如，x=10;由此可见利用赋值运算符将一个变量与一个表达式连接起来的式子为赋值表达式，在表达式后面加"；"便构成了赋值语句。使用"="的赋值语句格式如下：

```
变量 = 表达式;
```

示例如下

```
a = 0xFF; //将常数十六进制数FF赋于变量a
b = c = 33; //同时赋值给变量b,c
```

```
d = e; //将变量e的值赋于变量d
f = a+b; //将变量a+b的值赋于变量f
```

由上面的例子可以知道赋值语句的意义就是先计算出"="右边的表达式的值，然后将得到的值赋给左边的变量。而且右边的表达式可以是一个赋值表达式。

在一些朋友的来信中会出现"=="与"="这两个符号混淆的错误原码，问为何编译报错，往往就是错在if (a=x)之类的语句中，错将"="用为"=="。"=="符号是用来进行相等关系运算。**算术，增减量运算符**

对于a+b,a/b这样的表达式大家都很熟悉，用在C语言中，+，/，就是算术运算符。C51中的算术运算符有如下几个,其中只有取正值和取负值运算符是单目运算符,其它则都是双目运算符：

+ 加或取正值运算符
- 减或取负值运算符
* 乘运算符
/ 除运算符
% 取余运算符

算术表达式的形式：

表达式1 算术运算符 表达式2

如：a+b*(10-a), (x+9)/(y-a)

除法运算符和一般的算术运算规则有所不同，如是两浮点数相除，其结果为浮点数，如10.0/20.0所得值为0.5，而两个整数相除时，所得值就是整数，如7/3，值为2。像别的语言一样C的运算符与有优先级和结合性，同样可用用括号"（）"来改变优先级。这些和我们小时候学的数学几乎是一样的，我也不必过多的说明了。

:(还有这么多运算符呀！暂时停一停吧，我们先来做一个实验吧。学习运算符和另外一些知识时,我们还是给我们的实验板加个串行接口吧。借助电脑转件直观的看单片机的输出结果，以后我还会用一些简单的实例讲解单片机和PC串口通讯的简单应用和编程。如果你用的是成品实验板或仿真器，那你就可以跳过这一段了。

在制作电路前我们先来看看要用的MAX232，这里我们不去具体讨论它，只要知道它是TTL和RS232电平相互转换的芯片和基本的引脚接线功能就行了。通常我会用两个小功率晶体管加少量的电路去替换MAX232，可以省一点，效果也不错（如有兴趣可以查看http://www.cdle.net网站中的相关资料）。下图就是MAX232的基本接线图。

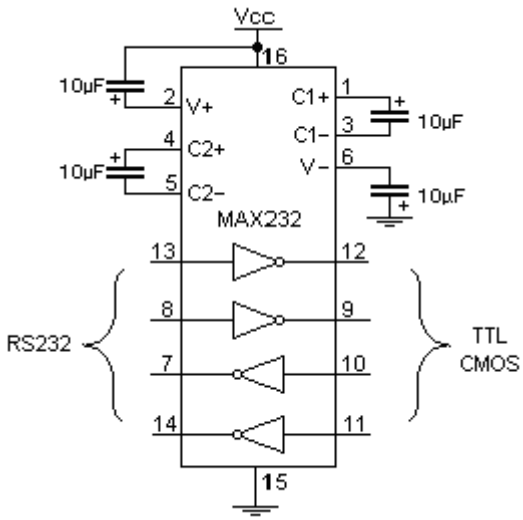


图7－1　MAX232

在上两课的电路的基础上按图7－3加上MAX232就可以了。这大热天的拿烙铁焊焊，还真的是热气迫人来呀：P串口座用DB9的母头，这样就可以用买来的PC串口延长线进行和电脑相连接，也可以直接接到电脑com口上。

图7－2　DB9接头

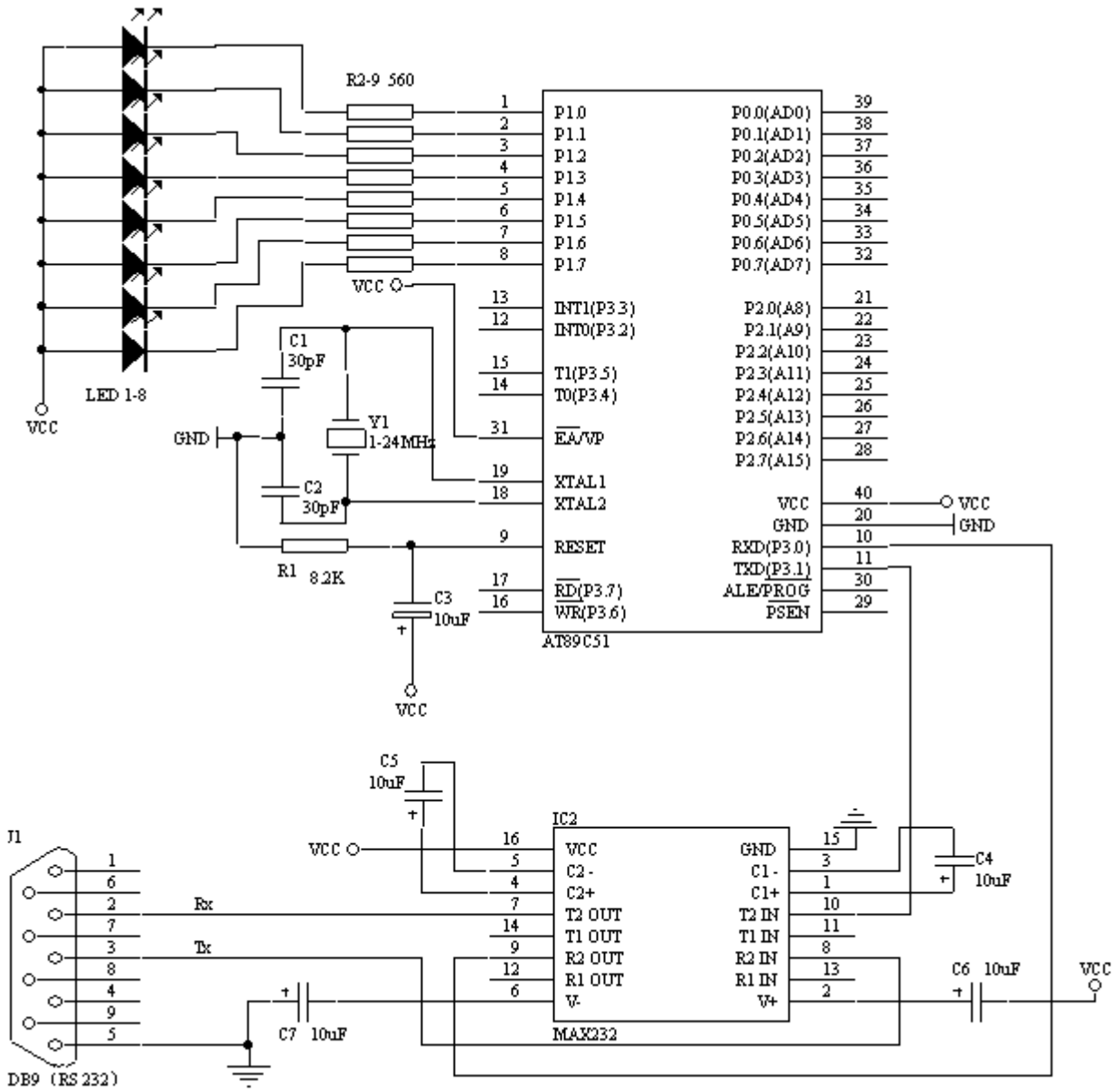


图7－3 加上了MAX232的实验电路

做好后我们就先用回第一课的"Hello World！"程序，用它来和你的电脑说声Hello!把程序烧到芯片上，把串口连接好。嘿嘿，这时要打开你的串口调试软件，没有就赶快到网上DOWN一个了。你会用Windows的超级中端也行，不过我从不用它。我用http://emouze.com的comdebug，它是个不错的软件，我喜欢它是因为它功能好而且还有"线路状态"功能，这对我制作小玩意时很有用。串口号，波特率调好，打开串口，单片机上电，就可以在接收区看到不断出现的"Hello World!"。一定要先打开软件的串口，再把单片机上电，否则可能因字符不对齐而看到乱码哦。

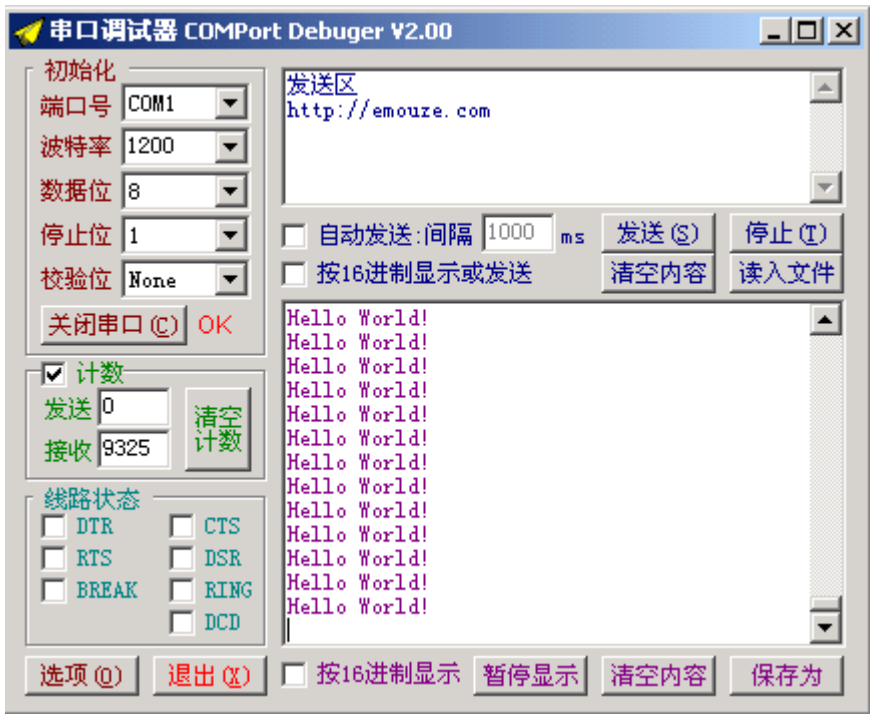


图 7-4 调试结果

第七课 运算符和表达式（2）

关系运算符

对于关系运算符，同样我们也并不陌生。C中有六种关系运算符，这些家伙同样是在小时候学算术时学习过的：

> 大于
< 小于
>= 大于等于
<= 小于等于
== 等于
! = 等于

或者你是个非C程序员，那么对前四个一定是再熟悉不过的了。而"=="在VB或PASCAL等中是用"=","!="则是用"not "。由于工作关系我自己要使用好几种的程序语言，所以有时也会头晕搞错。老了咯 ：P

小学时的数学课就教授过运算符是有优先级别的，计算机的语言也不过是人类语言的一种扩展，这里的运算符同样有着优先级别。前四个具有相同的优先级，后两个也具有相同的优先级，但是前四个的优先级要高于后2个的。

当两个表达式用关系运算符连接起来时，这时就是关系表达式。关系表达式通常是用来判别某个条件是否满足。要注意的是用关系运算符的运算结果只有0和1两种，也就是逻辑的真与假，当指定的条件满足时结果为1，不满足时结果为0。

表达式1 关系运算符 表达式2

如：I＜J，I==J,(I=4)>(J=3),J+I>J

借助我们在上一课做好的电路和学习了的相关操作。我们来做一个关系运算符相关的实例程序。为了增加学习的趣味性和生动性，不妨我们来假设在做一个会做算术的机器人，当然真正会思考对话的机器，我想我是做不出来的了，这里的程序只是用来学习关系运算符的基本应用。

```
#include <AT89X51.H>
#include <stdio.h>
void main(void)
{
int x,y;
SCON = 0x50; //串口方式1,允许接收
TMOD = 0x20; //定时器1定时方式2
TH1 = 0xE8; //11.0592MHz 1200波特率
TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器

while(1)
{
printf("您好!我叫Robot!我是一个会做算术的机器人!\n"); //显示
printf("请您输入两个int,X 和 Y\n"); //显示
scanf("%d%d",&x,&y); //输入
if (x < y)
printf("X<Y\n"); //当X小于Y时
else //当X不小于Y时再作判断
{
if (x == y)
printf("X=Y\n"); //当X等于Y时
else
printf("X>Y\n"); //当X大于Y时
}
}
}
```

要注意的是，在连接PC串口调试时。发送数字时，发送完一个数字后还要发送一个回车符，以使scanf函数确认有数据输入。Printf,scanf函数的具体用法，将和其它相关函数集中出现在www.cdle.net的C51函数详解中，敬请大家留意。

逻辑运算符

关系运算符所能反映的是两个表达式之间的大小等于关系，那逻辑运算符则是用于求条件式的逻辑值，用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式了。也许你会对为什?quot;逻辑运算符将关系表达式连接起来就是逻辑表达式了"这一个描述有疑惑的地方。其实之前说过"要注意的是用关系运算符的运算结果只有0和1两种，也就是逻辑的真与假"，换句话说也就是逻辑量，而逻辑运算符就用于对逻辑量运算的表达。至于复杂的逻辑量的运算法则我也知之甚少，如要了解的朋友可以参看数字电路的教科书、逻辑学或数学书，而之里只能说说简单常用的几种。逻辑表达式的一般形式为：

逻辑与：条件式1 && 条件式2

逻辑或：条件式1 || 条件式2

逻辑非:！条件式2

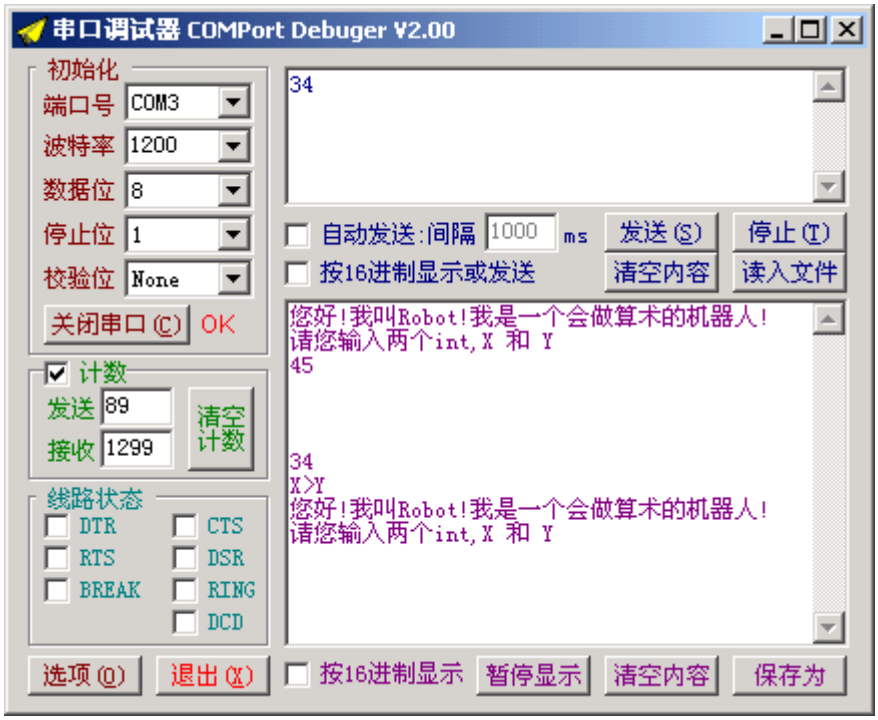


图7－5 演示结果

逻辑与，说白了就是当条件式1"与"条件式2都为真时结果为真（非0值），否则为假（0值）。也就是说运算会先对条件式1进行判断，如果为真（非0值），则继续对条件式2进行判断，当结果为真时，逻辑运算的结果为真（值为1），如果结果不为真时，逻辑运算的结果为假（0值）。如果在判断条件式1时就不为真的话，就不用再判断条件式2了，而直接给出运算结果为假。

逻辑或，是指只要二个运算条件中有一个为真时，运算结果就为真，只有当条件式都不为真时，逻辑运算结果才为假。

逻辑非则是把逻辑运算结果值取反，也就是说如果两个条件式的运算值为真，进行逻辑非运算后则结果变为假，条件式运算值为假时最后逻辑结果为真。

同样逻辑运算符也有优先级别，！（逻辑非）→&&（逻辑与）→||（逻辑或）,逻辑非的优先值最高。

如有 !True || False && True
按逻辑运算的优先级别来分析则得到（True代表真，False代表假）
!True || False && True
False || False && True //!Ture先运算得False
False || False //False && True运算得False
False //最终False || False得False
下面我们来用程序语言去有表达，如下：

```
#include <AT89X51.H>
#include <stdio.h>

void main(void)

{
unsigned char True = 1; //定义
unsigned char False = 0;

SCON = 0x50; //串口方式1,允许接收

TMOD = 0x20; //定时器1定时方式2
TH1 = 0xE8; //11.0592MHz 1200波特率
TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器


if (!True || False && True)
printf("True\n"); //当结果为真时
else
printf("False\n"); //结果为假时
}
```

大家可以使用以往学习的方法用keil或烧到片子上用串口调试。可以更改"!True || False && True"这个条件式，以实验不同算法组合来掌握逻辑运算符的使用方法。

第七课 运算符和表达式（3）

位运算符

学过汇编的朋友都知道汇编对位的处理能力是很强的，但是C语言也能对运算对象进行按位操作，从而使C语言也能具有一定的对硬件直接进行操作的能力。位运算符的作用是按位对变量进行运算，但是并不改变参与运算的变量的值。如果要求按位改变变量的值，则要利用相应的赋值运算。还有就是位运算符是不能用来对浮点型数据进行操作的。C51中共有6种位运算符。

位运算一般的表达形式如下：

变量1 位运算符 变量2

位运算符也有优先级，从高到低依次是:"~"(按位取反)→"<<"(左移) →">>"(右移) →"&"(按位与)→"^"(按位异或)→"|"(按位或)

表7－1是位逻辑运算符的真值表，X表示变量1，Y表示变量2

X	Y	~X	~Y	X&Y	X|Y	X^Y
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

表 7－1　按位取反，与，或和异或的逻辑真值表

利用以前建立起来的实验板，我们来做个实验验证一下位运算是否真是不改变参与变量的值，同时学习位运算的表达形式。程序很简单，用P1口做运算变量，P1.0-P1.7对应P1变量的最低位到最高位，通过连接在P1口上的LED我们便可以直观看到每个位运算后变量是否有改变或如何改变。程序如下：

```
#include
void main(void)
{
unsigned int a;
unsigned int b;
unsigned char temp; //临时变量
P1 = 0xAA; //点亮D1,D3,D5,D7 P1口的二进制为10101010,为0时点亮LED
for (a=0;a<1000;a++)
for (b=0;b<1000;b++); //延时
temp = P1 & 0x7; //单纯的写P1|0x7是没有意义的，因为没有变量被影响，不会被编译
//执行P1丨0x7后结果存入temp，这时改变的是temp，但P1不会被影响。
//这时LED没有变化，仍然是D1,D3,D5,D7亮
for (a=0;a<1000;a++)
for (b=0;b<1000;b++); //延时
P1 = 0xFF; //熄灭LED
for (a=0;a<1000;a++)
for (b=0;b<1000;b++); //延时
P1 = 0xAA; //点亮D1,D3,D5,D7 P1口的二进制为10101010,为0时点亮LED
for (a=0;a<1000;a++)
for (b=0;b<1000;b++); //延时
P1 = P1 & 0x7; //这时LED会变得只有D2灭
//因为之前P1=0xAA=10101010
//与0x7位与  0x7=00000111
//结果存入P1 P1=00000010 //位为O时点亮LED，电路看第三课
```

```
for (a=0;a<1000;a++)
for (b=0;b<1000;b++); //延时
P1 = 0xFF; //熄灭LED
while(1);
//大家可以根据上面的程序去做位或，左移，取反等等。
}
```

复合赋值运算符

复合赋值运算符就是在赋值运算符"="的前面加上其他运算符。以下是C语言中的复合赋值运算符：

+= 加法赋值	>>= 右移位赋值
-= 减法赋值	&= 逻辑与赋值
*= 乘法赋值	|= 逻辑或赋值
/= 除法赋值	^= 逻辑异或赋值
%= 取模赋值	-= 逻辑非赋值
<<= 左移位赋值	

复合运算的一般形式为：

变量 复合赋值运算符 表达式

其含义就是变量与表达式先进行运算符所要求的运算，再把运算结果赋值给参与运算的变量。其实这是C语言中一种简化程序的一种方法，凡是二目运算都可以用复合赋值运算符去简化表达。例如：

a+=56等价于a=a+56

y/=x+9 等价于 y=y/(x+9)

很明显采用复合赋值运算符会降低程序的可读性，但这样却可以使程序代码简单化，并能提高编译的效率。对于初学C语言的朋友在编程时最好还是根据自己的理解力和习惯去使用程序表达的方式，不要一味追求程序代码的短小。

逗号运算符

如果你有编程的经验，那么对逗号的作用也不会陌生了。如在VB中"Dim a,b,c"的逗号就是把多个变量定义为同一类型的变量，在C也一样，如"int a,b,c",这些例子说明逗号用于分隔表达式用。但在C语言中逗号还是一种特殊的运算符，也就是逗号运算符，可以用它将两个或多个表达式连接起来，形成逗号表达式。逗号表达式的一般形式为：

表达式1，表达式2，表达式3……表达式n

这样用逗号运算符组成的表达式在程序运行时，是从左到右计算出各个表达式的值，而整个用逗号运算符组成的表达式的值等于最右边表达式的值，就是"表达式n"的值。在实际的应用中，大部分情况下，使用逗号表达式的目的只是为了分别得到名个表达式的值，而并不一定要得到和使用整个逗号表达式的值。要注意的还有，并不是在程序的任何位置出现的逗号，都可以认为是逗号运算符。如函数中的参数，同类型变量的定义中的逗号只是用来间隔之用而不是逗号运算符。

条件运算符

上面我们说过C语言中有一个三目运算符，它就是"?:"条件运算符，它要求有三个运算对象。它可以把三个表达式连接构成一个条件表达式。条件表达式的一般形式如下:

逻辑表达式? 表达式1 : 表达式2

条件运算符的作用简单来说就是根据逻辑表达式的值选择使用表达式的值。当逻辑表达式的值为真时（非0值）时，整个表达式的值为表达式1的值；当逻辑表达式的值为假（值为0）时，整个表达式的值为表达式2的值。要注意的是条件表达式中逻辑表达式的类型可以与表达式1和表达式2的类型不一样。下面是一个逻辑表达式的例子。

如有a=1,b=2这时我们要求是取ab两数中的较小的值放入min变量中，也许你会这样写：

```
if (a min = a;
else
min = b; //这一段的意思是当a
```

用条件运算符去构成条件表达式就变得简单明了了：

```
min = (a
```

很明显它的结果和含意都和上面的一段程序是一样的，但是代码却比上一段程序少很多，编译的效率也相对要高，但有着和复合赋值表达式一样的缺点就是可读性相对效差。在实际应用时根据自己要习惯使用，就我自己来说我喜欢使用较为好读的方式和加上适当的注解，这样可以有助于程序的调试和编写，也便于日后的修改读写。

指针和地址运算符

在第四课我们学习数据类型时，学习过指针类型，知道它是一种存放指向另一个数据的地址的变量类型。指针是C语言中一个十分重要的概念，也是学习C语言中的一个难点。对于指针将会在第九课中做详细的讲解。在这里我们先来了解一下C语言中提供的两个专门用于指针和地址的运算符：

* 取内容

& 取地址

取内容和地址的一般形式分别为：

变量 =* 指针变量

指针变量 =& 目标变量

取内容运算是将指针变量所指向的目标变量的值赋给左边的变量；取地址运算是将目标变量的地址赋给左边的变量。要注意的是：指针变量中只能存放地址（也就是指针型数据），一般情况下不要将非指针类型的数据赋值给一个指针变量。

下面来看一个例子，并用一个图表和实例去简单理解指针的用法和含义。

设有两个unsigned int 变量 ABC处CBA 存放在0x0028,0x002A中

另有一个指针变量 portA 存放在0x002C中

那么我们写这样一段程序去看看*,&的运算结果

```
unsigned int data ABC _at_ 0x0028;
unsigned int data CBA _at_ 0x002A;
unsigned int data *Port _at_ 0x002C;

#include
#include

void main(void)
{
    SCON = 0x50; //串口方式1,允许接收
    TMOD = 0x20; //定时器1定时方式2
    TH1 = 0xE8; //11.0592MHz 1200波特率
    TL1 = 0xE8;
    TI = 1;
    TR1 = 1; //启动定时器
    ABC = 10; //设初值
    CBA = 20;
    Port = &CBA; //取CBA的地址放到指针变量Port
    *Port = 100; //更改指针变量Port所指向的地址的内容
    printf("1: CBA=%d\n",CBA); //显示此时CBA的值
    Port = &ABC; //取ABC的地址放到指针变量Port
    CBA = *Port; //把当前Port所指的地址的内容赋给变量CBA
    printf("2: CBA=%d\n",CBA); //显示此时CBA的值
    printf("   ABC=%d\n",ABC); //显示ABC的值
}
```

程序初始时

值	地址	说明
0x00	0x002DH	
0x00	0x002CH	
0x00	0x002BH	
0x00	0x002AH	
0x0A	0x0029H	
0x00	0x0028H	

执行 ABC = 10;向 ABC 所指的地址 0x28H 写入 10(0xA)，因 ABC 是 int 类型要占用 0x28H 和 0x29H 两个字节的内存空间，低位字节会放入高地址中，所以 0x28H 中放入 0x00，0x29H 中放入 0x0A

值	地址	说明
0x00	0x002DH	
0x00	0x002CH	
0x00	0x002BH	
0x00	0x002AH	
0x0A	0x0029H	ABC为int类型占用两字节
0x00	0x0028H	

执行 CBA = 20;原理和上一句一样

值	地址	说明
0x00	0x002DH	
0x00	0x002CH	
0x14	0x002BH	CBA为int类型占用两字节
0x00	0x002AH	
0x0A	0x0029H	ABC为int类型占用两字节
0x00	0x0028H	

执行 Port = &CBA; 取 CBA 的首地址放到指针变量 Port

值	地址	说明
0x00	0x002DH	
0x2A	0x002CH	CBA的首地址存入Port
0x14	0x002BH	
0x00	0x002AH	
0x0A	0x0029H	
0x00	0x0028H	

*Port = 100; 更改指针变量 Port 所指向的地址的内容

值	地址	说明
0x00	0x002DH	
0x2A	0x002CH	
0x64	0x002BH	Port指向了CBA所在地址2AH
0x00	0x002AH	并存入100
0x0A	0x0029H	
0x00	0x0028H	

其它的语句也是一样的道理，大家可以用 Keil 的单步执行和打开存储器查看器一看，这样就更容易理解了。

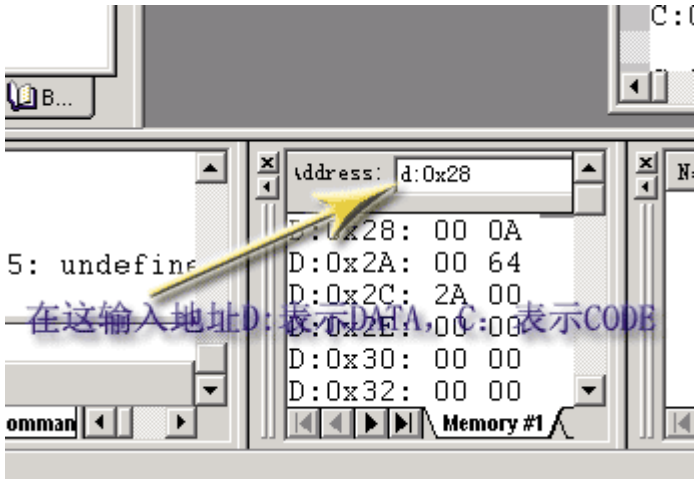


图 7－6　存储器查看窗

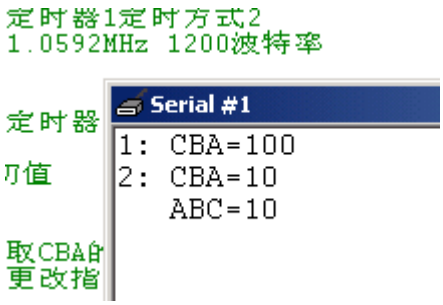


图7－7　在串行调试窗口的最终结果

sizeof运算符

看上去这确实是个奇怪的运算符，有点像函数，却又不是。大家看到size应该就猜到是和大小有关的吧？是的，sizeof是用来求数据类型、变量或是表达式的字节数的一个运算符，但它并不像"＝"之类运算符那样在程序执行后才能计算出结果，它是直接在编译时产生结果的。它的语

法如下：

sizeof (数据类型)

sizeof (表达式)

下面是两句应用例句，程序大家可以试着编写一下。

printf("char是多少个字节? %bd 字节\n",sizeof(char));

printf("long是多少个字节? %bd 字节\n",sizeof(long));

结果是:

char是多少个字节? 1字节
long是多少个字节? 4字节

强制类型转换运算符

不知你们是否有自己去试着编一些程序，从中是否有遇到一些问题？初学时我就遇到过这样一个问题：两个不同数据类型的数在相互赋值时会出现不对的值。如下面的一段小程序：

```
void main(void)
{
unsigned char a;
unsigned int b;

b=100*4;

a=b;
while(1);
}
```

这段小程序并没有什么实际的应用意义，如果你是细心的朋友定会发现a的值是不会等于100*4的。是的a和b一个是char类型一个是int类型，从以前的学习可知char只占一个字节值最大只能是255。但编译时为何不出错呢？先来看看这程序的运行情况：

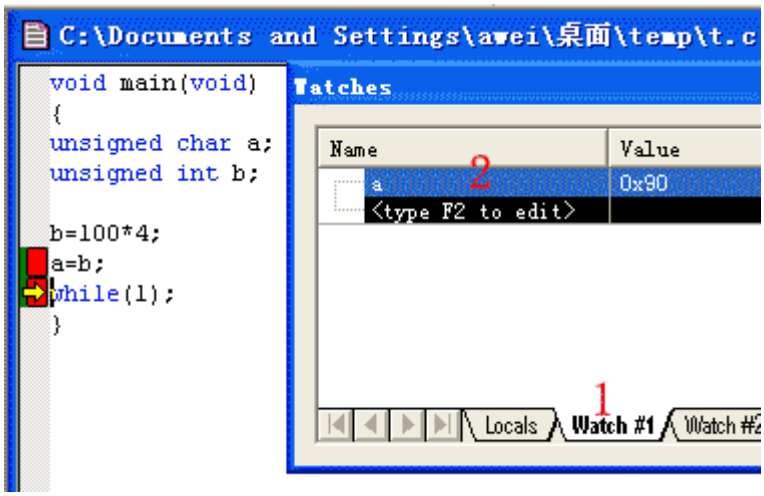


图 7－8 小程序的运行情况

b=100*4就可以得知b=0x190,这时我们可以在Watches查看a的值，对于watches窗口我们在第5课时简单学习过，在这个窗口Locals页里可以查看程序运行中的变量的值，也可以在watch页中输入所要查看的变量名对它的值进行查看。做法是按图中1的watch#1(或watch#2),然后光标移到图中的2按F2键，这样就可以输入变量名了。在这里我们可以查看到a的值为0x90，也就是b的低8位。这是因为执行了数据类型的隐式转换。隐式转换是在程序进行编译时由编译器自动去处理完成的。所以有必要了解隐式转换的规则：

1．变量赋值时发生的隐式转换，"＝"号右边的表达式的数据类型转换成左边变量的数据类型。就如上面例子中的把INT赋值给CHAR字符型变量，得到的CHAR将会是INT的低8位。如把浮点数赋值给整形变量，小数部分将丢失。

2．所有char型的操作数转换成int型。

3．两个具有不同数据类型的操作数用运算符连接时，隐式转换会按以下次序进行：如有一操作数是float类型，则另一个操作数也会转换成float类型；如果一个操作数为long类型，另一个也转换成long；如果一个操作数是unsigned类型，则另一个操作会被转换成unsigned类型。

从上面的规则可以大概知道有那几种数据类型是可以进行隐式转换的。是的，在C51中只有char,int,long及float这几种基本的数据类型可以被隐式转换。而其它的数据类型就只能用到显示转换。要使用强制转换运算符应遵循以下的表达形式：

(类型) 表达式

用显示类型转换来处理不同类型的数据间运算和赋值是十分方便和方便的，特别对指针变量赋值是很有用的。看一面一段小程序：

```
#include
#include
void main(void)
{
char xdata * XROM;
char a;
int Aa = 0xFB1C;
long Ba = 0x893B7832;
float Ca = 3.4534;
SCON = 0x50; //串口方式1,允许接收
TMOD = 0x20; //定时器1定时方式2
TH1 = 0xE8; //11.0592MHz 1200波特率
TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器
XROM=(char xdata *) 0xB012; //给指针变量赋XROM初值
*XROM = 'R'; //给XROM指向的绝对地址赋值
```

```
a = *((char xdata *) 0xB012); //等同于a = *XROM
printf ("%bx %x %d %c \n",(char) Aa, (int) Ba,(int)Ca, a);//转换类型并输出
while(1);
}
```

程序运行结果：1c 7832 3 R

在上面这段程序中，可以很清楚到到各种类型进行强制类型转换的基本用法，程序中先在外部数据存储器XDATA中定义了一个字符型指针变量XROM，当用XROM=(char xdata *) 0xB012这一语句时，便把0xB012这个地址指针赋于了XROM，如你用XROM则会是非法的，这种方法特别适合于用标识符来存取绝对地址，如在程序前用#define ROM 0xB012这样的语句，在程序中就可以用上面的方法用ROM对绝对地址0xB012进行存取操作了。

在附录三中运算符的优先级说明。

在这课的完结后，C语言中一些数据类型和运算规律已基本学习完了，下一课会开始讲述语法，函数等。

第八课 语 句(1)-表达式语句

从第四课到第七课，学习了大部分的基本语法，这一课所要学习的各种基本语句的语法可以说是组成程序的灵魂。在前面的课程中的例子里，也简单理解过一些语句的用法，可以看出C语言是一种结构化的程序设计语言。C语言提供了相当丰富的程序控制语句。学习掌握这些语句的用法也是C语言学习中的重点。

表达式语句是最基本的一种语句。不同的程序设计语言都会有不一样的表达式语句，如VB就是在表达式后面加入回车就构成了VB的表达式语句，而在51单片机的C语言中则是加入分号";"构成表达式语句。举例如下：

```
b = b * 10;
Count++;
X = A;Y = B;
Page = (a+b)/a-1;
```

以上的都是合法的表达式语句。在我收到的一些网友的Email中，发现很多初学的朋友往往在编写调试程序时忽略了分号";"，造成程序不法被正常的编译。我个人的经验是在遇到编译错误时先语法是否有误，这在初学时往往会因在程序中加入了全角符号、运算符打错漏掉或没有在后面加";"。

在C语言中有一个特殊的表达式语句，称为空语句，它仅仅是由一个分号";"组成。有时候为了使语法正确，那么就要求有一个语句，但这个语句又没有实际的运行效果那么这时就要有一个空语句。说起来就像大家在晚自修的时候用书包占位一样，呵呵。

空语句通常用会以下两种用法。

(1)while,for构成的循环语句后面加一个分号，形成一个不执行其它操作的空循环体。我会会常常用它来写等待事件发生的程序。大家要注意的是";"号作为空语句使用时，要与语句中有效组成部分的分号相区分，如 for (;a<50000;a++);第一个分号也应该算是空语句，它会使a赋值为0(但要注意的是如程序前有a值，则a的初值为a的当前值)，最后一个分号则使整个语句行成一个空循环。那么for (;a<50000;a++);就相当于for (a=0;a<50000;a++);我个人习惯是写后面的写法，这样能使人更容易读明白。

(2)在程序中为有关语句提供标号，标记程序执行的位置，使相关语句能跳转到要执行的位置。这会用在goto语句中。

下面的示例程序是简单说明while空语句的用法。硬件的功能很简单，就是在P3.7上接一个开关，当开关按下时P1上的灯会全亮起来。当然实际应用中按键的功能实现并没有这么的简单，往往还要进行防抖动处理等。

先在我们的实验板上加一个按键。电路图如图8－1。

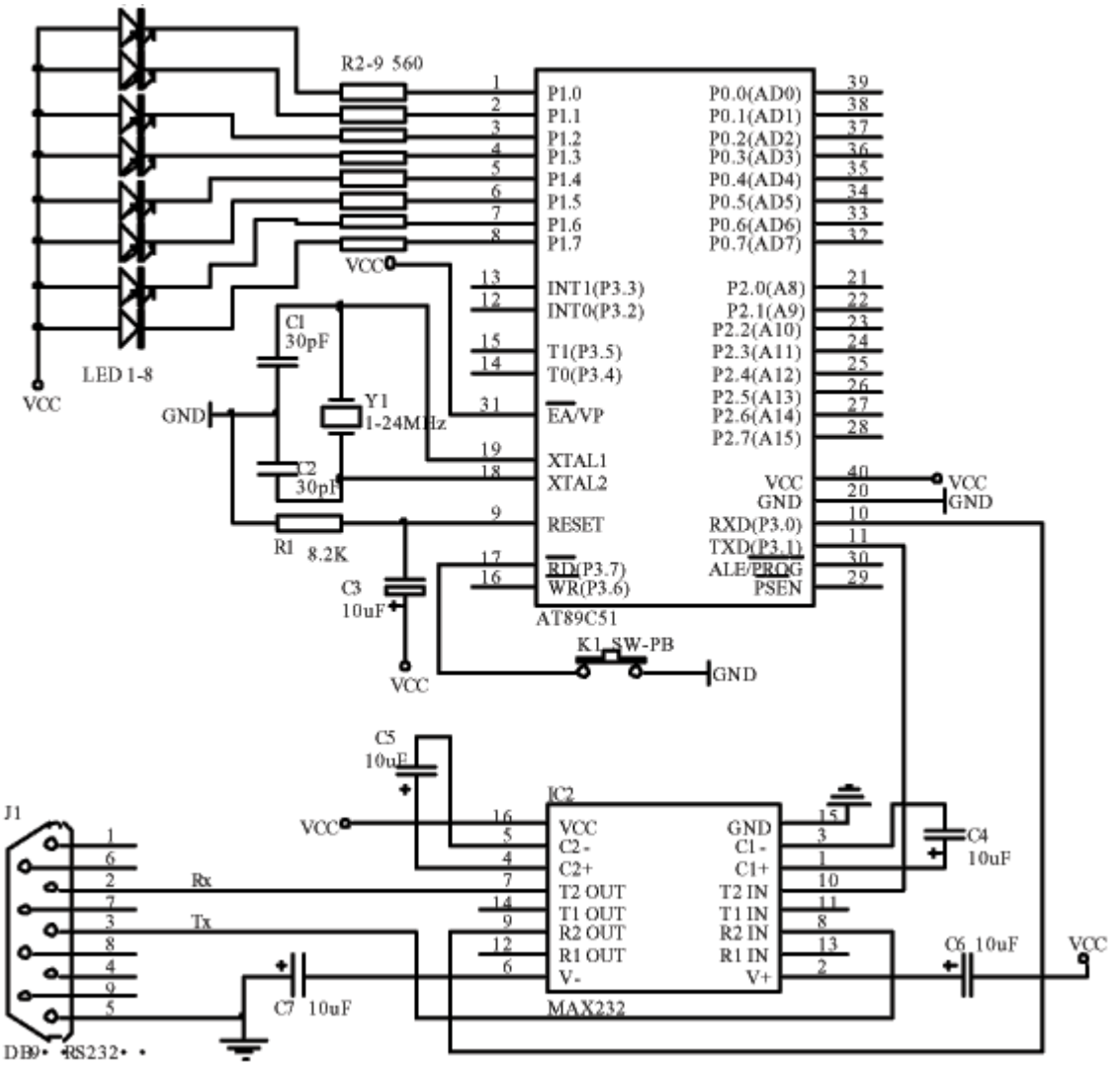


图8-1 加了按键的实验电路图

程序如下:

```
#include <AT89x51.h>

void main(void)

{
unsigned int a;
do
{
P1 = 0xFF; //关闭P1上的LED
while(P3_7); //空语句，等待P3_7按下为低电平，低电平时执行下面的语句
P1 = 0; //点亮LED
for(;a<60000;a++); //这也是空语句的用法,注意a的初值为当前值
} //这样第一次按下时会有一延时点亮一段时间，以后按多久就亮多久
while(1); //点亮一段时间后关闭再次判断P3_7,如此循环
}
```

上面的实验电路已加入了 RS232 串行口电路，只要稍微改变一下，就能变为具有仿真功能的 实验电路。这个改变的关键就是把芯片改用 SST89C58，并在芯片中烧入仿真监控程序。SST89C58 同样也是一种 51 架构的单片机，它具有 24K＋8K 的两个程序存储区，能选择其 一做为程序的启动区。只要把一个叫 SOFTICE.HEX 的监控程序用支持 SST89C58 的编程器烧录到芯片中（使用编程器或用 CA 版的 SST89C58 烧录 SOFTICE 的具体方法和文件能参考 / ），就 能把上 面 的电路升级为

MON51 仿真实验器。那么怎么用它和 KEIL 实现联机仿真呢？

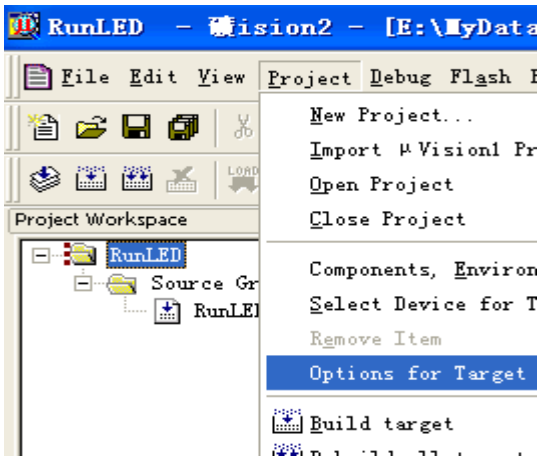


图 10－2 项目设置菜单

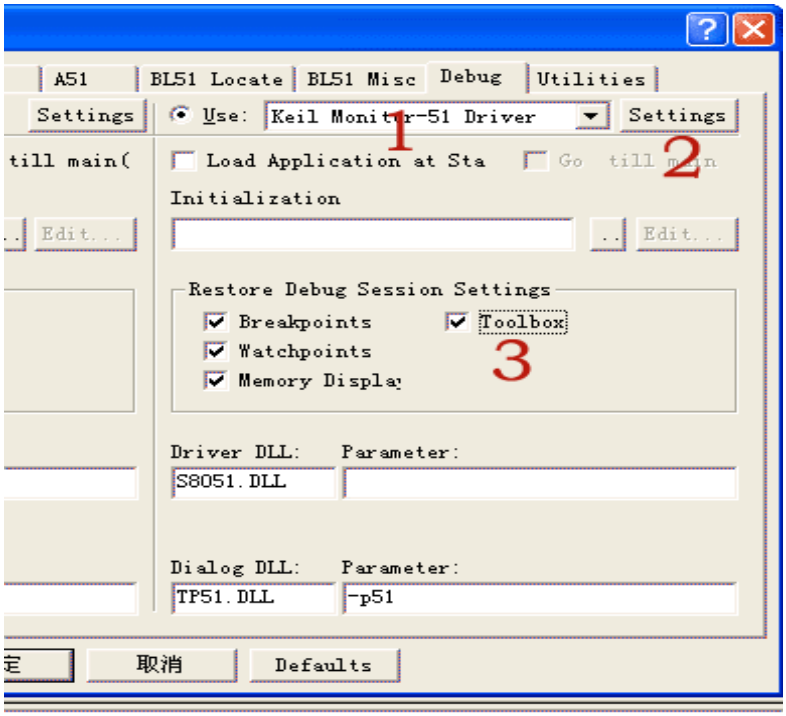


图 10－3　项目设置 首先要在你要仿真的程序项目设置仿真器所使用的驱动，在 Debug 页中选择对应本仿真器的 KeilMon51 驱 动，如图 10 中 1 所示。图 10-3 的 3 是选择在仿真时能使用的工具窗口，如内存显示，断点等等。按 2 进 行图 10-4 中的仿真器设置。设置好串行口号，波特率，晶体震荡器为 11.0592M 时选 38400。Cache Options 为仿真 缓选取后会加快仿真的运行的速度。设好后编译运行程序就能连接仿真器了，连接成功会出现如图 10－

5 的画面。如连接不成功就出现图 10－6 的图，这个时候能先复位电路再按"Try Again"，还不成功连接的话则 应检查软件设置和硬件电路。图 10－5 中 1 是指示仿真器的固件版本为 F-MON51V3.4 版。点击 3 中小红 点位置时为设置和取消断点，点击 2 则运行到下一个断点。图 10－7 则是变量和存储器的查看。仿真器在

软件大概的使用方法和软件仿真相差不多。

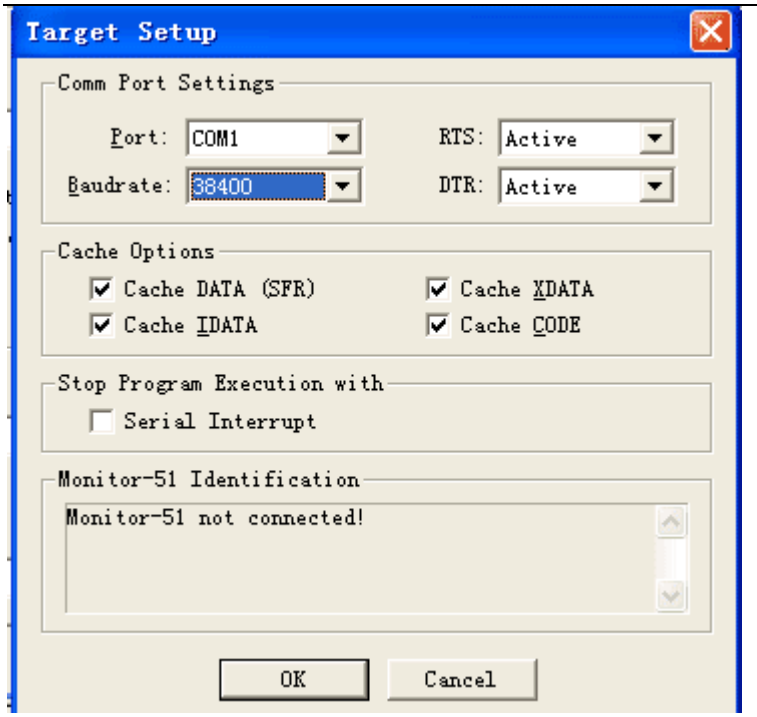


图 10－4 仿真器设置

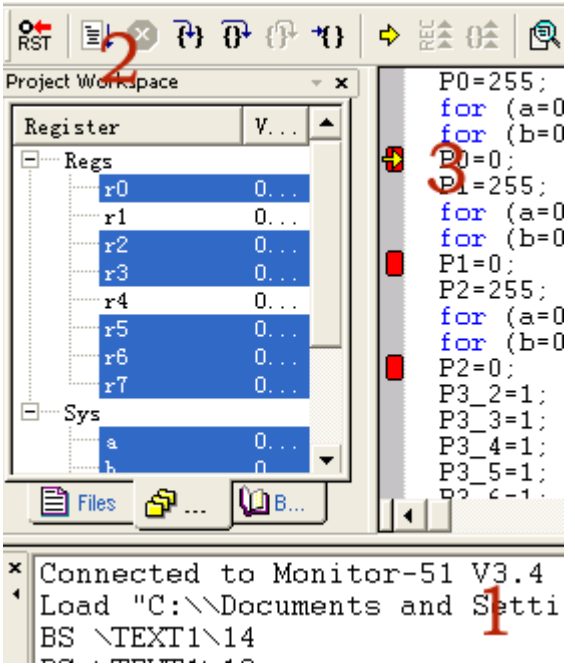


图 10－5 仿真器连接成功

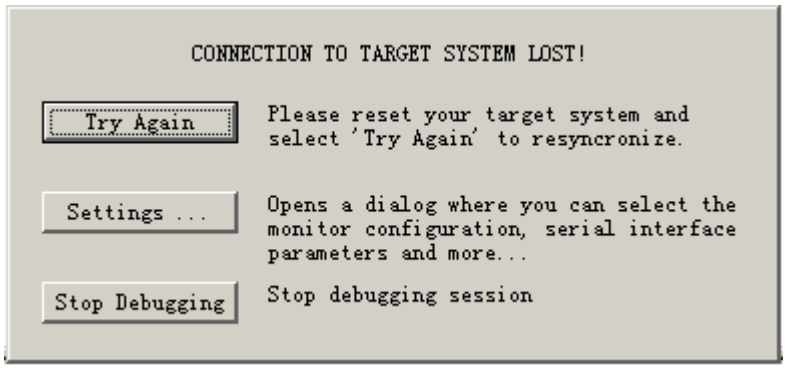


图 10－6 连接不成功提示

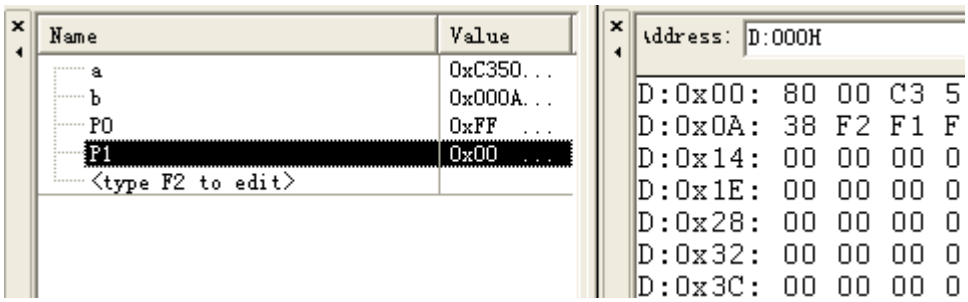


图 10－7 变量及内存查看

第八课 语 句(2)-复合语句

曾经在BBS上有朋友问过我{}是什么意思？什么作用？在C中是有不少的括号，如{},[],()等，确实会让一些初入门的朋友不解。在VB等一些语言中同一个()号会有不同的作用，它可以用于组合若干条语句形成功能块，可以用做数组的下标等，而在C中括号的分工较为明显，{}号是用于将若干条语句组合在一起形成一种功能块，这种由若干条语句组合而成的语句就叫复合语句。复合语句之间用{}分隔，而它内部的各条语句还是需要以分号";"结束。复合语句是允许嵌套的，也是就是在{}中的{}也是复合语句。复合语句在程序运行时，{}中的各行单语句是依次顺序执行的。以C语言中可以将复合语句视为一条单语句，也就是说在语法上等同于一条单语句。对于一个函数而言，函数体就是一个复合语句，也许大家会因此知道复合语句中不单可以用可执行语句组成，还可以用变量定义语句组成。要注意的是在复合语句中所定义的变量，称为局部变量，所谓局部变量就是指它的有效范围只在复合语句中，而函数也算是复合语句，所以函数内定义的变量有效范围也只在函数内部。关于局部变量和全局变量的具体用法会在说到函数时具体说明。下面用一段简单的例子简单说明复合语句和局部变量的使用。

```
#include <at89x51.h>

#include <stdio.h>

void main(void)

{
```

```
unsigned int a,b,c,d; //这个定义会在整个main函数中?

SCON = 0x50; //串口方式1,允许接收

TMOD = 0x20; //定时器1定时方式2
TH1 = 0xE8; //11.0592MHz 1200波特率
TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器

a = 5;

b = 6;
c = 7;
d = 8; //这会在整个函数有效
printf("0: %d,%d,%d,%d\n",a,b,c,d);
{ //复合语句1
unsigned int a,e; //只在复合语句1中有效
a = 10,e = 100;
printf("1: %d,%d,%d,%d,%d\n",a,b,c,d,e);
{ //复合语句2
unsigned int b,f; //只在复合语句2中有效
b = 11,f = 200;
printf("2: %d,%d,%d,%d,%d,%d\n",a,b,c,d,e,f);
}//复合语句2结束
printf("1: %d,%d,%d,%d,%d\n",a,b,c,d,e);
}//复合语句1结束
printf("0: %d,%d,%d,%d\n",a,b,c,d);

while(1);

}
```

运行结果：

0：5,6,7,8
1: 10,6,7,8,100
2: 10,11,7,8,100,200
1: 10,6,7,8,100
0：5,6,7,8

结合以上的说明想想为何结果会是这样。

第八课 语 句(3)-条件语句

看到题目后相信大家都会大概对条件语句这个概念有所认识。是的，就如学习语文中的条件语句一样，C语言也一样是"如果XX就XX"或是"如果XX就XX否则XX"。也就是当条件符合时就执行语句。条件语句又被称为分支语句，其关键字是由if构成。C语言提供了3种形式的条件语句:

1: if (条件表达式) 语句

当条件表达式的结果为真时，就执行语句，否则就跳过。

如 if (a= =b) a++; 当a等于b时，a就加1

2: if (条件表达式) 语句1

else 语句2

当条件表达式成立时，就执行语句1，否则就执行语句2

如 if (a= =b)

a++;

else

a--;

当a等于b时，a加1，否则a-1。

3：if (条件表达式1) 语句1

else if (条件表达式2) 语句2

else if (条件表达式3) 语句3

else if (条件表达式m) 语句n

else 语句m

这是由if else语句组成的嵌套，用来实现多方向条件分支，使用时因注意if和else的配对使用，要是少了一个就会语法出错，记住else总是与最临近的if相配对。一般条件 语句只会用作单一条件或少数量的分支，如果多数量的分支时则更多的会用到下一篇中的开 关语句。如果使用条件语句来编写超过 3 个以上的分支程序的话，会使程序变得不是那么清晰易读。

第八课 语 句(4)-开关语句

我们学习了条件语句，用多个条件语句可以实现多方向条件分支，但是可以发现使用过多的条件语句实现多方向分支会使条件语句嵌套过多，程序冗长，这样读起来也很不好读。这时使用开关语句同样可以达到处理多分支选择的目的，又可以使程序结构清晰。它的语法为下：

```
switch (表达式)
{
case 常量表达式1: 语句1; break;
case 常量表达式2: 语句2; break;
case 常量表达式3: 语句3; break;
```

case 常量表达式n: 语句n; break;
default: 语句
}

运行中switch后面的表达式的值将会做为条件，与case后面的各个常量表达式的值相对比，如果相等时则执行后面的语句，再执行break（间断语句）语句，跳出switch语句。如果case没有和条件相等的值时就执行default后的语句。当要求没有符合的条件时不做任何处理，则可以不写default语句。

在上面的课程中我们一直在用printf这个标准的C输出函数做字符的输出，使用它当然会很方便，但它的功能强大，所占用的存储空间自然也很大，要1K左右字节空间，如果再加上scanf输入函数就要达到2K左右的字节，这样的话如果要求用2K存储空间的芯片时就无法再使用这两个函数，例如AT89C2051。在这些小项目中，通常我们只是要求简单的字符输入输出，这里以笔者发表在《无线电杂志》的一个简单的串口应用实例为例，一来学习使用开关语句的使用，二来简单了解51芯片串口基本编程。这个实例是用PC串口通过上位机程序与由AT89C51组成的下位机相通讯，实现用PC软件控制AT89C51芯片的IO口，这样也就可以再通过相关电路实现对设备的控制(这里是控制继电器)。在笔者的网站http://www.cdle.net还可以查看相关文章。所使用的硬件还是用回我们以上课程中做好的硬件，以串口和PC连接，用LED查看实验的结果。下面是源代码。

```
/*---------------------------------------
CDLE-J20_Main.c
PC串口控制IO口电路
可以用字符控制和读取IO口
简单版本V2.0
更加好的单片机版本和PC控制软件和DLL动态库
请访问磁动力工作室http://www.cdle.net
Copyright 2003 http://www.cdle.net

All rights reserved.
明浩  E-mail: pnzwzw@163.com
pnzwzw@cdle.net
---------------------------------------*/
#include <AT89X51.h>
static unsigned char data CN[4];
static unsigned char data CT;
unsigned char TS[8] = {254,252,248,240,224,192,128,0};
void main(void)
{
```

2014.11.28
zdw

```
void InitCom(unsigned char BaudRate);
void ComOutChar(unsigned char OutData);
void CSToOut(void);
void CNToOut(void);
unsigned int a;

CT = 0; //接收字符序列
CN[0] = 0;
CN[1] = 51;
CN[2] = 51;
CN[3] = 0;
InitCom(6); //设置波特率为9600 1-8波特率300－57600
EA = 1;
ES = 1; //开串口中断
do
{
for (a=0; a<30000; a++)
P3_6 = 1;
for (a=0; a<30000; a++) //指示灯闪动
P3_6 = 0;
}
while(1);
}


//串口初始化  晶振为11.0592M   方式1  波特率300－57600
void InitCom(unsigned char BaudRate)
{
unsigned char THTL;
switch (BaudRate)
{
case 1: THTL = 64; break; //波特率300
case 2: THTL = 160; break; //600
case 3: THTL = 208; break; //1200
case 4: THTL = 232; break; //2400
case 5: THTL = 244; break; //4800
case 6: THTL = 250; break; //9600
case 7: THTL = 253; break; //19200
case 8: THTL = 255; break; //57600
default: THTL = 208;
```

```
}
SCON = 0x50; //串口方式1,允许接收
TMOD = 0x20; //定时器1定时方式2
TCON = 0x40; //设定时器1开始计数
TH1 = THTL;
TL1 = THTL;
PCON = 0x80; //波特率加倍控制,SMOD位
RI = 0; //清收发标志
TI = 0;
TR1 = 1; //启动定时器
}
//向串口输出一个字符（非中断方式）
void ComOutChar(unsigned char OutData)
{
SBUF = OutData; //输出字符
while(!TI); //空语句判断字符是否发完
TI = 0; //清TI
}
//串口接收中断
void ComInINT(void) interrupt 4 using 1
{
if (RI) //判断是不是收完字符
{
if (CT>3)
{
CT = 0; //收完一组数据，序列指针清零
CN[0] = 0;
CN[1] = 51;
CN[2] = 51;
CN[3] = 0;
}
CN[CT] = SBUF;
CT++;
RI = 0; //RI清零
if (CN[0]==0x61 && CN[3]==0x61) //用aXXa的简单方式保证接收的可靠性，可以满足业余的要求
{ //a也可以为板下的ID号，在同一个串行口上可以挂上一块以上的板
CSToOut(); //收到的数据格式正确时，调用控制输出函数
} //要想更为可靠的工作则要用到数据检验和通讯协议
```

```
}
}
//根据全局变量输出相应的控制信号
void CSToOut(void)
{
unsigned char data a;
unsigned int data b;
switch(CN[1]) //aXXa的格式定义是第一个X为端口,0为P0，1为P1，2为P2，3为关闭所有（同时要第2个X为3，XX＝33）
{ //XX＝44为测试用，5为读取端口状态，大于5则为无效数据，
case 0: //第一个X小于3时，第二个X为要输出的数据。
P0 = CN[2];
CNToOut();
break;
case 1:
P1 = CN[2];
CNToOut();
break;
case 2:
P2 = CN[2];
CNToOut();
break;
case 3:
P0 = 0xFF;
P1 = 0xFF;
P2 = 0xFF;
CNToOut();
break;
case 4:
P0 = 0xFF;
P1 = 0xFF;
P2 = 0xFF;
for (a=0; a<8; a++)
{
P0 = TS[a];
for (b=0; b<50000; b++);
}
P0 = 0xFF;
for (a=0; a<8; a++)
```

```
{
P1 = TS[a];
for (b=0; b<50000; b++);
}
P1 = 0xFF;
for (a=0; a<4; a++)
{
P2 = TS[a];
for (b=0; b<50000; b++);
}
P2 = 0xFF;
CNToOut();
break;
case 5: //根据CN[2]返回所要读取的端口值
switch(CN[2])
{
case 0:
ComOutChar(CN[0]);
ComOutChar(CN[1]);
ComOutChar(P0);
ComOutChar(CN[3]);
break;
case 1:
ComOutChar(CN[0]);
ComOutChar(CN[1]);
ComOutChar(P1);
ComOutChar(CN[3]);
break;
case 2:
ComOutChar(CN[0]);
ComOutChar(CN[1]);
ComOutChar(P2);
ComOutChar(CN[3]);
break;
case 3:
ComOutChar(CN[0]);
ComOutChar(CN[1]);
ComOutChar(P3);
ComOutChar(CN[3]);
break;
```

```
}
break;
}
}
void CNToOut(void)
{
ComOutChar(CN[0]);
ComOutChar(CN[1]);
ComOutChar(CN[2]);
ComOutChar(CN[3]);
}
```

代码中有多处使用开关语句的，使用它对不同的条件做不同的处理，如在CSToOut函数中根据CN[1]来选择输出到那个IO口，如CN[1]=0则把CN[2]的值送到P0，CN[1]=1则送到P1，这样的写法比起用if (CN[1]==0)这样的判断语句来的清晰明了。当然它们的效果没有太大的差别（在不考虑编译后的代码执行效率的情况下）。

在这段代码其主要的作用就是通过串口和上位机软件进行通讯，跟据上位机的命令字串，对指定的IO端口进行读写。InitCom函数，原型为void InitCom(unsigned char BaudRate)，其作用为初始化串口。它的输入参数为一个字节，程序就是用这个参数做为开关语句的选择参数。如调用InitCom(6),函数就会把波特率设置为9600。当然这段代码只使用了一种波特率，可以用更高效率的语句去编写，这里就不多讨论了。

看到这里，你也许会问函数中的SCON，TCON，TMOD，SCOM等是代表什么？它们是特殊功能寄存器，在以前也略提到过，51芯片的特殊功能寄存器说明可以参看附录二的'AT89C51特殊功能寄存器列表'，在这里简单的说说串口相关的硬件设置。

SBUF 数据缓冲寄存器 这是一个可以直接寻址的串行口专用寄存器。有朋友这样问起过“为何在串行口收发中，都只是使用到同一个寄存器SBUF？而不是收发各用一个寄存器。”实际上SBUF包含了两个独立的寄存器，一个是发送寄存，另一个是接收寄存器，但它们都共同使用同一个寻址地址－99H。CPU在读SBUF时会指到接收寄存器，在写时会指到发送寄存器，而且接收寄存器是双缓冲寄存器，这样可以避免接收中断没有及时的被响应，数据没有被取走，下一帧数据已到来，而造成的数据重叠问题。发送器则不需要用到双缓冲，一般情况下我们在写发送程序时也不必用到发送中断去外理发送数据。操作SBUF寄存器的方法则很简单，只要把这个99H地址用关键字sfr定义为一个变量就可以对其进行读写操作了，如sfr SBUF = 0x99;当然你也可以用其它的名称。通常在标准的reg51.h或at89x51.h等头文件中已对其做了定义，只要用#include引用就可以了。

SCON 串行口控制寄存器 通常在芯片或设备中为了监视或控制接口状态，都会引用到接口控制寄存器。SCON就是51芯片的串行口控制寄存器。它的寻址地址是98H，是一个可以位寻址的寄存器，作用就是监视和控制51芯片串行口的工作状态。51芯片的串口可以工作在几个不同的工作模式下，其工作模式的设置就是使用SCON寄存器。它的各个位的具体定义如下：

（MSB）							（LSB）
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

表8－1 串行口控制寄存器SCON

SM0、SM1 为串行口工作模式设置位，这样两位可以对应进行四种模式的设置。看表 8－2 串行口工作模式设置。

SM0	SM1	模 式	功 能	波特率
0	0	0	同步移位寄存器	fosc/12
0	1	1	8位UART	可变
1	0	2	9位UART	fosc/32或fosc/64
1	1	3	9位UART	可变

表8－2 串行口工作模式设置

在这里只说明最常用的模式1，其它的模式也就一一略过，有兴趣的朋友可以找相关的硬件资料查看。表中的fosc代表振荡器的频率，也就是晶振的频率。UART为(Universal Asynchronous Receiver）的英文缩写。

SM2在模式2、模式3中为多处理机通信使能位。在模式0中要求该位为0。

REM为允许接收位，REM置1时串口允许接收，置0时禁止接收。REM是由软件置位或清零。如果在一个电路中接收和发送引脚P3.0,P3.1都和上位机相连，在软件上有串口中断处理程序，当要求在处理某个子程序时不允许串口被上位机来的控制字符产生中断，那么可以在这个子程序的开始处加入REM=0来禁止接收，在子程序结束处加入REM=1再次打开串口接收。大家也可以用上面的实际源码加入REM=0来进行实验。

TB8发送数据位8，在模式2和3是要发送的第9位。该位可以用软件根据需要置位或清除，通常这位在通信协议中做奇偶位，在多处理机通信中这一位则用于表示是地址帧还是数据帧。

RB8接收数据位8，在模式2和3是已接收数据的第9位。该位可能是奇偶位，地址/数据标识位。在模式0中，RB8为保留位没有被使用。在模式1中，当SM2=0，RB8是已接收数据的停止位。

TI发送中断标识位。在模式0，发送完第8位数据时，由硬件置位。其它模式中则是在发送停止位之初，由硬件置位。TI置位后，申请中断，CPU响应中断后，发送下一帧数据。在任何模式下，TI都必须由软件来清除，也就是说在数据写入到SBUF后，硬件发送数据，中断响应（如中断打开），这时TI=1，表明发送已完成，TI不会由硬件清除，所以这时必须用软件对其清零。

RI接收中断标识位。在模式0，接收第8位结束时，由硬件置位。其它模式中则是在接收停止位的半中间，由硬件置位。RI=1，申请中断，要求CPU取走数据。但在模式1中，SM2=1时，当未收到有效的停止位，则不会对RI置位。同样RI也必须要靠软件清除。

常用的串口模式1是传输10个位的，1位起始位为0,8位数据位，低位在先，1位停止位为1。它的波特率是可变的，其速率是取决于定时器1或定时器2的定时值（溢出速率）。AT89C51和AT89C2051等51系列芯片只有两个定时器，定时器0和定时器1，而定时器2是89C52系列芯片才有的。

波特率 在使用串口做通讯时，一个很重要的参数就是波特率，只有上下位机的波特率一样时才可以进行正常通讯。波特率是指串行端口每秒内可以传输的波特位数。有一些初学的朋友认为波特率是指每秒传输的字节数，如标准9600会被误认为每秒种可以传送9600个字节，而实际上它是指每秒可以传送9600个二进位，而一个字节要8个二进位，如用串口模式1来传输那么加上起始位和停止位，每个数据字节就要占用10个二进位，9600波特率用模式1传输时，每秒传输的字节数是9600÷10＝960字节。51芯片的串口工作模式0的波特率是固定的，为fosc/12，以一个12M的晶振来计算，那么它的波特率可以达到1M。模式2的波特率是固定在fosc/64或fosc/32，具体用那一种就取决于PCON寄存器中的SMOD位，如SMOD为0，波特率为focs/64,SMOD为1，波特率为focs/32。模式1和模式3的波特率是可变的，取决于定时器1或2（52芯片）的溢出速率。那么我们怎么去计算这两个模式的波特率设置时相关的寄存器的值呢？可以用以下的公式去计算。

波特率＝（2SMOD÷32）×定时器1溢出速率

上式中如设置了PCON寄存器中的SMOD位为1时就可以把波特率提升2倍。通常会使用定时器1工作在定时器工作模式2下，这时定时值中的TL1做为计数，TH1做为自动重装值 ，这个定时模式下，定时器溢出后，TH1的值会自动装载到TL1，再次开始计数，这样可以不用软件去干预，使得定时更准确。在这个定时模式2下定时器1溢出速率的计算公式如下：

溢出速率＝（计数速率）/(256－TH1)

上式中的“计数速率”与所使用的晶体振荡器频率有关，在51芯片中定时器启动后会在每一个机器周期使定时寄存器TH的值增加一，一个机器周期等于十二个振荡周期，所以可以得知51芯片的计数速率为晶体振荡器频率的1/12，一个12M的晶振用在51芯片上，那么51的计数速率就为1M。通常用11.0592M晶体是为了得到标准的无误差的波特率，那么为何呢？计算一下就知道了。如我们要得到9600的波特率，晶振为11.0592M和12M，定时器1为模式2，SMOD设为1，分别看看那所要求的TH1为何值。代入公式：

11.0592M

9600＝(2÷32)×((11.0592M/12)/(256-TH1))

TH1＝250 //看看是不是和上面实例中的使用的数值一样？

12M

9600＝(2÷32)×((12M/12)/(256-TH1))

TH1≈249.49

上面的计算可以看出使用12M晶体的时候计算出来的TH1不为整数，而TH1的值只能取整数，这样它就会有一定的误差存在不能产生精确的9600波特率。当然一定的误差是可以在使用中被接受的，就算使用11.0592M的晶体振荡器也会因晶体本身所存在的误差使波特率产生误差，但晶体本身的误差对波特率的影响是十分之小的，可以忽略不计。

这一节借着学习开关语句的机会，简略说明了串行的一些相关内容，但串口的工作方式设定有好种同时也要涉及到其它的相关寄存器，内容十分多，在此也不能一一做实例说明，下面的章节也会加入一些硬件方面的东西。

第八课 语 句(5)-循环语句

循环语句是几乎每个程序都会用到的，它的作用就是用来实现需要反复进行多次的操作。如一个12M的51芯片应用电路中要求实现1毫秒的延时，那么就要执行1000次空语句才可以达到延时的目的（当然可以使用定时器来做，这里就不讨论），如果是写1000条空语句那是多么麻烦的事情，再者就是要占用很多的存储空间。我们可以知道这1000条空语句，无非就是一条空语句重复执行1000次，因此我们就可以用循环语句去写，这样不但使程序结构清晰明了，而且使其编译的效率大大的提高。在C语言中构成循环控制的语句有while,do-while,for和goto语句。同样都是起到循环作用，但具体的作用和用法又大不一样。我们具体来看看。

goto语句

这个语句在很多高级语言中都会有，记得小时候用BASIC时就很喜欢用这个语句。它是一个无条件的转向语句，只要执行到这个语句，程序指针就会跳转到goto后的标号所在的程序段。它的语法如下：

goto 语句标号;

其中的语句标号为一个带冒号的标识符。示例如下

```
void main(void)
{
unsigned char a;
start: a++;
if (a= =10) goto end;
goto start;
end:;
}
```

上面一段程序可以说是一个死循环，没什么意思，只是说明一下goto的用法。这段程序的意

思是在程序开始处用标识符"start:"标识，表示程序这是程序的开始，"end:"标识程序的结束，标识符的定义应遵循前面所讲的标识符定义原则，不能用C的关键字也不能和其它变量和函数名相同，不然就会出错了。程序执行a++,a的值加1，当a等于10时程序会跳到end标识处结束程序，否则跳回到start标识处继续a++,直到a等于10。上面的示例说明goto不但可以无条件的转向,而且可以和if语句构成一个循环结构，这些在C程序员的程序中都不太常见，常见的goto语句用法是用它来跳出多重循环，不过它只可以从内层循环跳到外层循环，不能从外层循环跳到内层循环。在下面说到for循环语句时再略为提一提。为何大多数C程序员都不喜欢用goto语句？那是因为过多的使用它时会程序结构不清晰，过多的跳转就使程序又回到了汇编的编程风格，使程序失去了C的模块化的优点。

while 语句

while 语句的意思很不难理解，在英语中它的意思是“当…的时候…”，在这里我们可 以理解为“当条件为真的时候就执行后面的语句”，它的语法如下：

while (条件表达式) 语句;

使用 while 语句时要注意当条件表达式为真时，它才执行后面的语句，执行完后再次回

到 while 执行条件判断，为真时重复执行语句，为假时退出循环体。当条件一开始就为假时， 那么 while 后面的循环体（语句或复合语句）将一次都不执行就退出循环。在调试程序时要

注意 while 的判断条件不能为假而造成的死循环，调试时适当的在 while 处加入断点，也许 会使你的调试工作更加顺利。当然有时会使用到死循环来等待中断或 IO 信号等，如在第一 篇时我们就用了 while(1)来不停的输出“Hello World！”。下面的例子是显示从 1 到 10 的累 加和，读者能修改一下 while 中的条件看看结果会如果，从而体会一下 while 的使用方法。

```
#include <AT89X51.H>
#include <stdio.h>
void main(void)
{
unsigned int I = 1;
unsigned int SUM = 0; //设初值
SCON = 0x50; //串行口方式 1,允许接收
TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数
TH1 = 0xE8;   //11.0592MHz 1200 波特率
TL1 = 0xE8; TI = 1;
TR1 = 1; //启动定时器
while(I<=10)
```

```
{
SUM = I + SUM; //累加
printf ("%d SUM=%d\n",I,SUM); //显示
I++;
}
while(1); //这句是为了不让程序完后，程序指针继续向下造成程序“跑飞”
}
//最后运行结果是 SUM=55;
```

do while 语句

do while 语句能说是 while 语句的补充，while 是先判断条件是否成立再执行循环体，

而 do while 则是先执行循环体，再根据条件判断是否要退出循环。这样就决定了循环体无 论在任何条件下都会至少被执行一次。它的语法如下：

do 语句 while (条件表达式)

用 do while 怎么写上面那个例程呢？先想一想，再参考下面的程序。

```
#include <AT89X51.H>
#include <stdio.h>
void main(void)
{

unsigned int I = 1;
unsigned int SUM = 0; //设初值
SCON = 0x50; //串行口方式 1,允许接收 TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数
TH1 = 0xE8; //11.0592MHz 1200 波特率 TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器
do
{
SUM = I + SUM; //累加
printf ("%d SUM=%d\n",I,SUM); //显示 I++;
```

```
}
while(I<=10);
while(1);
}
```

在上面的程序看来 do while 语句和 while 语句似乎没有什么两样，但在实际的应用中要注意任何 do while 的循环体一定会被执行一次。如把上面两个程序中 I 的初值设为 11，那么 前一个程序不会得到显示结果，而后一个程序则会得到 SUM=11。

for 语句

在明确循环次数的情况下，for 语句比以上说的循环语句都要方便简单。它的语法如下：for ([初值设定表达式];[循环条件表达式];[条件更新表达式]) 语句 中括号中的表达式是可选的，这样 for 语句的变化就会很多样了。for 语句的执行：先代入初值，再判断条件是否为真，条件满足时执行循环体并更新条件，再判断条件是否为 真……直到条件为假时，退出循环。下面的例子所要实现的是和上二个例子一样的，对照着 看不难理解几个循环语句的差异。

```
#include <AT89X51.H>
#include <stdio.h>
void main(void)
{
unsigned int I;
unsigned int SUM = 0; //设初值
SCON = 0x50; //串行口方式 1,允许接收 TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数


TH1 = 0xE8; //11.0592MHz 1200 波特率 TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器
for (I=1; I<=10; I++) //这里能设初始值，所以变量定义时能不设
{
SUM = I + SUM; //累加
printf ("%d SUM=%d\n",I,SUM); //显示
```

```
}
while(1);
}
```

如果我们把程序中的 for 改成 for(; I<=10; I++)这样条件的初值会变成当前 I 变量的值。如果改成 for(;;)会怎么样呢？试试看。

continue 语句

continue 语句是用于中断的语句，通常使用在循环中，它的作用是结束本次循环，跳 过循环体中没有执行的语句，跳转到下一次循环周期。语法为：

continue;

continue 同时也是一个无条件跳转语句，但功能和前面说到的 break 语句有所不一样， continue 执行后不是跳出循环，而是跳到循环的开始并执行下一次的循环。在上面的例子 中的循环体加入 if (I==5) continue;看看什么结果？

return 语句

return 语句是返回语句，不属于循环语句，是要学习的最后一个语句所以一并写下了。 返回语句是用于结束函数的执行，返回到调用函数时的位置。语法有二种：

return (表达式);

return; 语法中因带有表达式，返回时先计算表达式，再返回表达式的值。不带表达式则返回的值不确定。

下面是一个同样是计算 1－10 的累加，所不一样是的用了函数的方式。

```
#include <AT89X51.H>
#include <stdio.h>
int Count(void); //声明函数
void main(void)
{
unsigned int temp;

SCON = 0x50; //串行口方式 1,允许接收 TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数
TH1 = 0xE8; //11.0592MHz 1200 波特率 TL1 = 0xE8;
TI = 1;
TR1 = 1; //启动定时器
```

```
temp = Count();
printf ("1-10 SUM=%d\n",temp); //显示
while(1);
}
int Count(void)
{
unsigned int I, SUM;
for (I=1; I<=10; I++)
{
SUM = I + SUM; //累加
}
return (SUM);
}
```

第9课 C51函数

上一篇的最后一个例子中有用到函数，其实一直出现在例子中的 main()也算是一个函数，只不过它比较特殊，编译时以它做为程序的开始段。有了函数 C 语言就有了模块化的优 点，一般功能较多的程序，会在编写程序时把每项单独的功能分成数个子程序模块，每个子 程序就能用函数来实现。函数还能被反复的调用，因此一些常用的函数能做成函数库 以供在编写程序时直接调用，从而更好的实现模块化的设计，大大提高编程工作的效率。 一．函数定义

通常 C 语言的编译器会自带标准的函数库，这些都是一些常用的函数，Keil uv 中也不 例外。标准函数已由编译器软件商编写定义，使用者直接调用就能了，而无需定义。但是 标准的函数不足以满足使用者的特殊要求，因此 C 语言允许使用者根据需要编写特定功能的 函数，要调用它必须要先对其进行定义。定义的模式如下：

函数类型 函数名称（形式参数表）

{

函数体

}

函数类型是说明所定义函数返回值的类型。返回值其实就是一个变量，只要按变量

类型来定义函数类型就行了。如函数不需要返回值函数类型能写作“void”表示该函数没 有返回值。注意的是函数体返回值的类型一定要和函数类型一致，不然会造成错误。函数名 称的定义在遵循 C 语言变量命名规则的同时，不能在同一程序中定义同名的函数这将会造成 编译错误（同一程序中是允许有同名变量的，因为变量有全局和局部变量之分）。形式参数 是指调用函数时要传入到函数体内参与运算的变量，它能有一个、几个或没有，当不需要 形式参数也就是无参函数，括号内能为空或写入“void”表示，但括号不能少。函数体中 能包含有局部变量的定义和程序语句，如函数要返回运算值则要使用 return 语句进行返 回。在函数的{}号中也能什么也不写，这就成了空函数，在一个程序项目中能写一些 空函数，在以后的修改和升级中能方便的在这些空函数中进行功能扩充。

二．函数的调用

函数定义好以后，要被其它函数调用了才能被执行。C 语言的函数是能相互调用的， 但在调用函数前，必须对函数的类型进行说明，就算是标准库函数也不例外。标准库函数的 说明会被按功能分别写在不一样的头文件中，使用时只要在文件最前面用#include 预处理语 句引入相应的头文件。如前面一直有使用的 printf 函数说明就是放在文件名为 stdio.h 的 头文件中。调用就是指一个函数体中引用另一个已定义的函数来实现所需要的功能，这个时候函 数体称为主调用函数，函数体中所引用的函数称为被调用函数。一个函数体中能调用数个 其它的函数，这些被调用的函数同样也能调用其它函数，也能嵌套调用。笔者本人认为 主函数只是相对于被调用函数而言。在 c51 语言中有一个函数是不能被其它函数所调用的， 它就是 main 主函数。调用函数的一般形式如下:

函数名 (实际参数表) “函数名”就是指被调用的函数。实际参数表能为零或多个参数，多个

参数时要用逗号隔开，每个参数的类型、位置应与函数定义时所的形式参数一一对应，它的作用就是把参 数传到被调用函数中的形式参数，如果类型不对应就会产生一些错误。调用的函数是无参函 数时不写参数，但不能省后面的括号。

在以前的一些例子我们也能看不一样的调用方式：

1．函数语句

如 printf ("Hello World!\n"); 这是在 我们的第一个程序中出现的，它以 "Hello World!\n"为参数调用 printf 这个库函数。在这里函数调用被看作了一条语句。

2．函数参数 “函数参数”这种方式是指被调用函数的返回值当作另一个被调用函数的实际参数，如 temp=StrToInt(CharB(16));CharB 的返回值作为 StrToInt 函数的实际参数传递。

3．函数表达式

而在上一篇的例子中有 temp = Count();这样一句，这个时候函数的调用作为一个运算 对象出现在表达式中，能称为函数表达式。例子中 Count()返回一个 int 类型的返回 值直接赋值给temp。注意的是这种调用方式要求被调用的函数能返回一个同类型的值， 不然会出现不可预料的错误。

前面说到调用函数前要对被调用的函数进行说明。标准库函数只要用#include 引入已 写好说明的头文件，在程序就能直接调用函数了。如调用的是自定义的函数则要用如下形 式编写函数类型说明

类型标识符 函数的名称(形式参数表); 这样的说明方式是用在被调函数定义和主调函数是在同一文件中。你也能把这些写到

文件名.h 的文件中用#include "文件名.h"引入。如果被调函数的定义和主调函数不是在同 一文件中的，则要用如下的方式进行说明，说明被调函数的定义在同一项目的不一样文件之上， 其实库函数的头文件也是如此说明库函数的，如果说明的函数也能称为外部函数。

extern 类型标识符 函数的名称(形式参数表); 函数的定义和说明是完全不一样的，在编译的角度上看函数的定义是把函数编译存放在

ROM 的某一段地址上，而函数说明是告诉编译器要在程序中使用那些函数并确定函数的地址。如果在同一文件中被调函数的定义在主调函数之前，这个时候能不用说明函数类型。也就 是说在 main 函数之前定义的函数，在程序中就能不用写函数类型说明了。能在一个函 数体调用另一个函数（嵌套调用），但不允许在一个函数定义中定义另一个函数。还要注意 的是函数定义和说明中的“类型、形参表、名称”等都要相一致。

三．中断函数 中断服务函数是编写单片机应用程序不可缺少的。

中断服务函数只有在中断源请求响应中断时才会被执行，这在处理突发事件和实时控制是十分有效的。例如：电路中一个按钮， 要求按钮后 LED 点亮，这个按钮何时会被按下是不可预知的，为了要捕获这个按钮的事件， 通常会有三种方法，一是用循环语句不断的对按钮进行查询，二是用定时中断在间隔时间内 扫描按钮，三是用外部中断服务函数对按钮进行捕获。在这个应用中只有单一的按钮功能， 那么第一种方式就能胜任了，程序也很简单，但是它会不停的

在对按钮进行查询浪费了

CPU 的时间。实际应用中一般都会还有其它的功能要求同时实现，这个时候能根据需要选用第二或第三种方式，第三种方式占用的 CPU 时间最少，只有在有按钮事件发生时，中断服务函 数才会被执行，其余的时间则是执行其它的任务。

如果你学习过汇编语言的话，刚开始写汇编的中断应用程序时，你一定会为出入堆栈的 问题而困扰过。单片机c语言 语言扩展了函数的定义使它能直接编写中断服务函数，你能不必考 虑出入堆栈的问题，从而提高了工作的效率。扩展的关键字是 interrupt，它是函数定义时 的一个选项，只要在一个函数定义后面加上这个选项，那么这个函数就变成了中断服务函数。在后面还能加上一个选项 using，这个选项是指定选用 51 芯片内部 4 组工作寄存器中的那个组。开始学习者能不必去做工作寄存器设定，而由编译器自动选择，避免产生不必要的错 误。定义中断服务函数时能用如下的形式。

函数类型 函数名 (形式参数) interrupt n [using n]

interrupt 关键字是不可缺少的，由它告诉编译器该函数是中断服务函数，并由后面的

n 指明所使用的中断号。n 的取值范围为 0－31,但具体的中断号要取决于芯片的型号，像 AT89c51 实际上就使用 0－4 号中断。每个中断号都对应一个中断向量，具体地址为 8n+3, 中断源响应后处理器会跳转到中断向量所处的地址执行程序，编译器会在这地址上产生一个 无条件跳转语句，转到中断服务函数所在的地址执行程序。下表是 51 芯片的中断向量和中 断号。

中断号	中断源	中断向量
0	外部中断 0	0003H
1	定时器/计数器 0	000BH
2	外部中断 1	0013H
3	定时器/计数器 1	001BH
4	串行口	0023H

表 9－1 AT89c51 芯片中断号和中断向量

使用中断服务函数时应注意:中断函数不能直接调用中断函数;不能通过形参传速参数; 在中断函数中调用其它函数，两者所使用的寄存器组应相同。限于篇幅其它与函数相关的知 识这里不能一一加以说明，如变量的传递、存储，局部变量、全部变量等，有兴趣的朋友可 以访问笔者的网站 阅读更多相关文章。

下面是简单的例子。首先要在前面做好的实验电路中加多一个按钮，接在 P3.2（12 引脚外 部中断 INT0）和地线之间。把编译好后的程序烧录到芯片后，当接在 P3.2 引脚的按钮接下 时，中断服务函数 Int0Demo 就会被执行，把 P3 当前的状态反映到 P1，如按钮接下后 P3.7

（之前有在这脚装过一按钮）为低，这个时候 P1.7 上的 LED 就会熄灭。放开 P3.2 上的按钮后，

P1LED 状态保持先前按下 P3.2 时 P3 的状态。

```
#include <at89x51.h>

unsigned char P3State(void); //函数的说明，中断函数不用说明
```

```
void main(void)
{
IT0 = 0; //设外部中断 0 为低电平触发
EX0 = 1; //允许响应外部中断 0
EA = 1; //总中断开关
while(1);
}
//外部中断 0 演示，使用 2 号寄存器组
void Int0Demo(void) interrupt 0 using 2

{
unsigned int Temp; //定义局部变量
P1 = ~P3State(); //调用函数取得 p2 的状态反相后并赋给 P1
for (Temp=0; Temp<50; Temp++); //延时    这里只是演示局部变量的使用
}
//用于返回 P3 的状态，演示函数的使用
unsigned char P3State(void)
{
unsigned char Temp;
Temp = P3; //读取 P3 的引脚状态并保存在变量 Temp 中
//这样只有一句语句实在没必要做成函数，这里只是学习函数的基本使用方法
return Temp;
}
```

第10课 C51数组的使用

前面的文章中，都是介绍单个数据变量的使用，在“走马灯”等的例子中略有使用到数组，不难看出，数组不过就是同一类型变量的有序集合。形象的能这样去理解，就像一个 学校在操场上排队，每一个级代表一个数据类型，每一个班级为一个数组，每一个学生就是 数组中的一个数据。数据中的每个数据都能用唯一的下标来确定其位置，下标能是一维 或多维的。就如在学校的方队中要找一个学生，这个学生在 I 年级 H 班 X 组 Y 号的，那么 能把这个学生看做在 I 类型的 H 数组中（X，Y）下标位置中。数组和普通变量一样，要

求先定义了才能使用，下面是定义一维或多维数组的方式：

数据类型	数组名	[常量表达式];
数据类型	数组名	[常量表达式 1]..... [常量表达式 N];

“数据类型”是指数组中的各数据单元的类型，每个数组中的数据单元只能是同一数据

类型。“数组名”是整个数组的标识，命名方法和变量命名方法是一样的。在编译时系统会 根据数组大小和类型为变量分配空间，数组名能说就是所分配空间的首地址的标识。“常 量表达式”是表示数组的长度和维数，它必须用“［］”括起，括号里的数不能是变量只能是 常量。

unsigned int xcount [10]; //定义无符号整形数组,有 10 个数据单元

char inputstring [5]; //定义字符形数组，有 5 个数据单元

float outnum [10],[10];//定义浮点型数组，有 100 个数据单元

在 C 语言中数组的下标是从 0 开始的而不是从 1 开始，如一个具有 10 个数据单元的数

组 count，它的下标就是从 count[0]到 count[9]，引用单个元素就是数组名加下标，如 count[1] 就是引用 count 数组中的第 2 个元素，如果错用了 count[10]就会有错误出现了。还有一点要 注意的就是在程序中只能逐个引用数组中的元素，不能一次引用整个数组，但是字符型的数 组就能一次引用整个数组。

数组也是能赋初值的。在上面介绍的定义方式只适用于定义在内存 DATA 存储器使 用的内存，有的时候我们需要把一些数据表存放在数组中，通常这些数据是不用在程序中改 变数值的，这个时候就要把这些数据在程序编写时就赋给数组变量。因为 51 芯片的片内 RAM 很有限，通常会把 RAM 分给参与运算的变量或数组，而那些程序中不变数据则应存放在片 内的 CODE 存储区，以节省宝贵的 RAM。赋初值的方式如下：

数据类型 [存储器类型] 数组名 [常量表达式] = {常量表达式};

数据类型 [存储器类型] 数组名 [常量表达式 1]..... [常量表达式 N]={{ 常量表达式}...{常量表达式 N}};

在定义并为数组赋初值时，开始学习的朋友一般会搞错初值个数和数组长度的关系，而致使 编译出错。初值个数必须小于或等于数组长度，不指定数组长度则会在编译时由实际的初值 个数自动设置。

unsigned char LEDNUM[2]={12,35}; //一维数组赋初值

int Key[2][3]={{1,2,4},{2,2,1}}; //二维数组赋初值

unsigned char IOStr[]={3,5,2,5,3}; //没有指定数组长度，编译器自动设置

unsigned char code skydata[]={0x02,0x34,0x22,0x32,0x21,0x12}; //数据保存在 code 区

下面的一个简单例子是对数组中的数据进行排序，使用的是冒泡法，一来了解数组的使 用，二来掌握基本的排序算法。冒泡排序算法是一种基本的排序算法，它每次顺序取数组中 的两个数，并按需要按其大小排列，在下一次循环中则取下一次的一个数和数组中下一个数 进行排序，直到数组中的数据全部排序完成。

```
#include <AT89X51.H>
#include <stdio.h>
void taxisfun (int taxis2[])
{
unsigned char TempCycA,TempCycB,Temp;
for (TempCycA=0; TempCycA<=8; TempCycA++)
for (TempCycB=0; TempCycB<=8-TempCycA; TempCycB++)
{//TempCycB<8-TempCycA 比用 TempCycB<=8 少用很多循环
if  (taxis2[TempCycB+1]>taxis2[TempCycB])  //当后一个数大于前一个 数
{
Temp = taxis2[TempCycB]; //前后 2 数交换
taxis2[TempCycB] = taxis2[TempCycB+1];
taxis2[TempCycB+1]  =  Temp;  //因函数参数是数组名调用形
参的变动影响实参
}
}
}
void main(void)
{
int taxis[] = {113,5,22,12,32,233,1,21,129,3};
char Text1[] = {"source data:"}; //"源数据"
char Text2[] = {"sorted data:"}; //"排序后数据"
```

```
unsigned char TempCyc;
SCON = 0x50; //串行口方式 1,允许接收
TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数
TH1 = 0xE8;   //11.0592MHz 1200 波特率
TL1 = 0xE8; TI = 1;
TR1 = 1; //启动定时器
printf("%s\n",Text1); //字符数组的整体引用
for (TempCyc=0; TempCyc<10; TempCyc++)
printf("%d ",taxis[TempCyc]);
printf("\n----------\n");
taxisfun (taxis); //以实际参数数组名 taxis 做参数被函数调用
printf("%s\n",Text2);
for (TempCyc=0; TempCyc<10; TempCyc++) //调用后 taxis 会被改变
printf("%d ",taxis[TempCyc]);

while(1);
}
```

例子中能看出，数组同样能作为函数的参数进行传递。数组做参数时是用数组名进

行传递的，一个数组的数组名表示该数组的首地址，在用数组名作为函数的调用参数时，它 的传递方式是采用了地址传递，就是将实际参数数组的首地址传递给函数中的形式参数数 组，这个时候实际参数数组和形式参数数组实际上是使用了同一段内存单元，当形式参数数组在 函数体中改变了元素的值，同时也会影响到实际参数数组，因为它们是存放在同一个地址的。 上面的例子同时还使用到字符数组。字符数组中每一个数据都是一个字符，这样一个一 维的字符数组就组成了一个字符串，在 C 语言中字符串是以字符数组来表达处理的。为了 能测定字符串的长度，C 语言中规定以‘\o’来做为字符串的结束标识，编译时会自动在字 符串的最后加入一个‘\o’，那么要注意的是如果用一个数组要保存一个长度为 10 字节的字 符串则要求这个数组至少能保存 11 个元素。‘\o’是转义字符，它的含义是空字符，它的 ASCII 码为 00H，也就是说当每一个字符串都是以数据 00H 结束的，在程序中操作字符数 据组时要注意这一点。字符数组除了能对数组中单个元素进行访问，还能访问整个数组， 其实整个访问字符数组就是把数组名传到函数中，数组名是一个指向数据存放空间的地址指 针，函数根据这个指针和‘/o’就能完整的操作这个字符数组。对于这一段所说的，能 参看下面一例 1602LCD 显示模块的驱动演示例子进行理解。这里要注意就是能用单个字

符数组元素来进行运算，但不能用整个数组来做运算，因为数组名是指针而不是数据。

```
/*==========================================================
使用 1602 液晶显示的实验例子  明浩  2004/2/27
==========================================================
SMC1602A(16*2)模拟口线接线方式 连接线图:
-------------------------------------------------
|LCM-----51        | LCM-----51          |    LCM------51 |
-----------------------------------------|
|DB0-----P1.0 | DB4-----P1.4 | RW-------P2.0 |
|DB1-----P1.1 | DB5-----P1.5 | RS-------P2.1 |
|DB2-----P1.2 | DB6-----P1.6 | E--------P2.2 |
|DB3-----P1.3 | DB7-----P1.7 | VLCD 接 1K 电阻到 GND|
-------------------------------------------------
[注:AT89S51 使用 12M 晶体震荡器]
==========================================================*/
#define LCM_RW P2_0 //定义引脚
#define LCM_RS P2_1
#define LCM_E P2_2
#define LCM_Data P1
#define Busy 0x80 //用于检测 LCM 状态字中的 Busy 标识
#include <at89x51.h>

void WriteDataLCM(unsigned char WDLCM);
void WriteCommandLCM(unsigned char WCLCM,BuysC);
unsigned char ReadDataLCM(void); unsigned char ReadStatusLCM(void); void LCMInit(void);
void DisplayOneChar(unsigned char X, unsigned char Y, unsigned char DData);
void DisplayListChar(unsigned char X, unsigned char Y, unsigned char code *DData);
void Delay5Ms(void);
void Delay400Ms(void);
unsigned char code cdle_net[] = {"www.51hei.com"};
unsigned char code email[] = {"pnzwzw@51hei.com"};
```

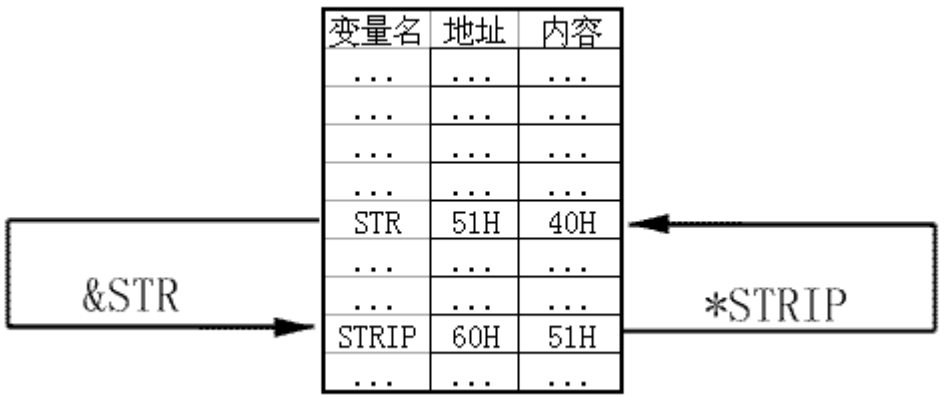
```
void main(void)
{
Delay400Ms(); //启动等待，等 LCM 讲入工作状态
LCMInit(); //LCM 初始化
Delay5Ms(); //延时片刻(可不要)
DisplayListChar(0, 0, cdle_net); DisplayListChar(0, 1, email); ReadDataLCM();//测试用句无意义
while(1);
}
//写数据
void WriteDataLCM(unsigned char WDLCM)
{
ReadStatusLCM(); //检测忙 LCM_Data = WDLCM; LCM_RS = 1;
LCM_RW = 0;
LCM_E = 0; //若晶体震荡器速度太高能在这后加小的延时
LCM_E = 0; //延时
LCM_E = 1;
}
//写指令
void WriteCommandLCM(unsigned char WCLCM,BuysC) //BuysC 为 0 时忽略忙检测
{
if (BuysC) ReadStatusLCM(); //根据需要检测忙
LCM_Data = WCLCM; LCM_RS = 0; LCM_RW = 0;
LCM_E = 0;

LCM_E = 0; LCM_E = 1;
}
//读数据
unsigned char ReadDataLCM(void)
{
LCM_RS = 1; LCM_RW = 1; LCM_E = 0; LCM_E = 0; LCM_E = 1; return(LCM_Data);
}
```

```
//读状态
unsigned char ReadStatusLCM(void)
{
LCM_Data = 0xFF; LCM_RS = 0; LCM_RW = 1; LCM_E = 0; LCM_E = 0; LCM_E = 1;
while (LCM_Data & Busy); //检测忙信号
return(LCM_Data);
}
void LCMInit(void) //LCM 初始化
{
LCM_Data = 0;
WriteCommandLCM(0x38,0); //三次显示模式设置，不检测忙信号
Delay5Ms(); WriteCommandLCM(0x38,0); Delay5Ms(); WriteCommandLCM(0x38,0); Delay5Ms();
WriteCommandLCM(0x38,1); //显示模式设置,开始要求每次检测忙信号
WriteCommandLCM(0x08,1); // 关闭显示 WriteCommandLCM(0x01,1); // 显示清屏 WriteCommandLCM(0x06,1); // 显示光标移动设置 WriteCommandLCM(0x0C,1); // 显示开及光标设置
}

//按指定位置显示一个字符
void DisplayOneChar(unsigned char X, unsigned char Y, unsigned char DData)
{
Y &= 0x1;
X &= 0xF; //限制 X 不能大于 15，Y 不能大于 1
if (Y) X |= 0x40; //当要显示第二行时地址码+0x40; X |= 0x80; //算出指令码
WriteCommandLCM(X, 0); //这里不检测忙信号，发送地址码
WriteDataLCM(DData);
}
//按指定位置显示一串字符
void DisplayListChar(unsigned char X, unsigned char Y, unsigned char code *DData)
{
unsigned char ListLength;
```

```
ListLength = 0; Y &= 0x1;
X &= 0xF; //限制 X 不能大于 15，Y 不能大于 1
while (DData[ListLength]>0x20) //若到达字串尾则退出
{
if (X <= 0xF) //X 坐标应小于 0xF
{
DisplayOneChar(X, Y, DData[ListLength]); //显示单个字符
ListLength++; X++;
}
}
}
//5ms 延时
void Delay5Ms(void)
{
unsigned int TempCyc = 5552;
while(TempCyc--);
}
//400ms 延时
void Delay400Ms(void)
{
unsigned char TempCycA = 5; unsigned int TempCycB; while(TempCycA--)
{
TempCycB=7269;

while(TempCycB--);
};
}
```

第11课 C51指针的使用

指针就是指变量或数据所在的存储区地址。如一个字符型的变量 STR 存放在内存单元 DATA 区的 51H 这个地址中，那么 DATA 区的 51H 地址就是变量 STR 的指针。在 C 语言中 指针是一个很重要的概念，正确有效的使用指针类型的数据，能更有效的表达复杂的数据 结构，能更有效的使用数组或变量，能方便直接的处理内存或其它存储区。指针之所以 能这么有效的操作数据，是因为无论程序的指令、常量、变量或特殊寄存器都要存放在内 存单元或相应的存储区中，这些存储区是按字节来划分的，每一个存储单元都能用唯一的 编号去读或写数据，这个编号就是常说的存储单元的地址，而读写这个编号的动作就叫做寻 址，通过寻址就能访问到存储区中的任一个能访问的单元，而这个功能是变量或数组等 是不可能代替的。C 语言也因此引入了指针类型的数据类型，专门用来确定其他类型数据的 地址。用一个变量来存放另一个变量的地址，那么用来存放变量地址的变量称为“指针变量”。 如用变量 STRIP 来存放文章开头的 STR 变量的地址 51H，变量 STRIP 就是指针变量。下面 用一个图表来说明变量的指针和指针变量两个不一样的概念。



变量的指针就是变量的地址，用取地址运算符‘&’取得赋给指针变量。&STR 就是把 变量 STR 的地址取得。用语句 STRIP = &STR 就能把所取得的 STR 指针存放在 STRIP 指 针变量中。STRIP 的值就变为 51H。可见指针变量的内容是另一个变量的地址，地址所属的 变量称为指针变量所指向的变量。

要访问变量 STR 除了能用‘STR’这个变量名来访问之外，还能用变量地址来访 问。方法是先用&STR 取变量地址并赋于 STRIP 指针变量，然后就能用*STRIP 来对 STR 进行访问了。‘*’是指针运算符，用它能取得指针变量所指向的地址的值。在上图中指针 变量 STRIP 所指向的地址是 51H，而 51H 中的值是 40H，那么*STRIP 所得的值就是 40H。 使用指针变量之前也和使用其它类型的变量那样要求先定义变量，而且形式也相类似，

一般的形式如下：

数据类型 ［存储器类型］ * 变量名;

unsigned char xdata *pi //指针会占用二字节，指针自身存放在编译器默认存储区，指向 xdata 存储区的 char 类型

unsigned char xdata * data pi; //除指针自身指定在 data 区，其它同上

int * pi; //定义为一般指针，指针自身存放在编译器默认存储区，占三个字节 在定义形式中“数据类型”是指所定义的指针变量所指向的变量的类型。“存储器类型”是编译器编译时的一种扩展标识，它是可选的。在没有“存储器类型”选项时，则定义为一般指针，如有“存储器类型”选项时则定义为基于存储器的指针。限于 51 芯片的寻址范围，

指针变量最大的值为 0xFFFF，这样就决定了一般指针在内存会占用 3 个字节，第一字节存 放该指针存储器类型编码，后两个则存放该指针的高低位址。而基于存储器的指针因为不用 识别存储器类型所以会占一或二个字节，idata,data,pdata 存储器指针占一个字节，code,xdata 则会占二个字节。由上可知，明确的定义指针，能节省存储器的开销，这在严格要求程序 体积的项目中很有用处。

指针的使用方法很多，限于篇幅以上只能对它做一些基础的介绍。下面用在讲述常量时 的例程改动一下，用以说明指针的基本使用方法。

```
#include <AT89X51.H> //预处理文件里面定义了特殊寄存器的名称如 P1 口定义为 P1
void main(void)
{
//定义花样数据，数据存放在片内 CODE 区中
unsigned char code design[]={0xFF,0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,
0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,0xFF,
0xFF,0xFE,0xFC,0xF8,0xF0,0xE0,0xC0,0x80,0x0,
0xE7,0xDB,0xBD,0x7E,0xFF};
unsigned int a;     //定义循环用的变量
unsigned char b;
unsigned char code * dsi;     //定义基于 CODE 区的指针


do{
dsi = &design[0];     //取得数组第一个单元的地址
for (b=0; b<32; b++)
```

```
{



}
}while(1);
}
for(a=0; a<30000; a++); //延时一段时间
P1 = *dsi;          //从指针指向的地址取数据到 P1 口
dsi++; //指针加一，
```

为了能清楚的了解指针的工作原理，能使用 keil uv2 的软件仿真器查看各变量和存储器的值。编译程序并执行，然后打开变量窗口，如图。用单步执行，就能查到到指针的变量。 如图中所示的是程序中循环执行到第二次，这个时候指针 dsi 指向 c:0x0004 这个地址，这个地址 的值是 0xFE。在存储器窗口则能察看各地址单元的值。使用这种方法不但在学习时能 帮助更好的了解语法或程序的工作，而且在实际使用中更能让你更快更准确的编写程序或解 决程序中的问题。

第12课 C51结构、联合和枚举的使用

前面的文章中介绍了 C 语言的基本数据类型,为了更有效的处理更复杂的数据,C 语 言引入了构造类型的数据类型。构造类型就是将一批各种类型的数据放在一起形成一种特殊 类型的数据。之前讨论过的数组也算是一种构造类型的数据,单片机c语言 中的构造类型还有结构、枚举和联合。

结构

结构是一种数据的集合体,它能按需要将不一样类型的变量组合在一起,整个集合体用 一个结构变量名表示,组成这个集合体的各个变量称为结构成员。理解结构的概念,能用 班级和学生的关系去理解。班级名称就相当于结构变量名,它代表所有同学的集合,而每个 同学就是这个结构中的成员。使用结构变量时,要先定义结构类型。一般定义格式如下:

struct 结构名 {结构元素表};

例子:struct FileInfo

{

unsigned char FileName[4]; unsigned long Date; unsigned int Size;

}

上面的例子中定义了一个简单的文件信息结构类型,它可用于定义用于简单的单片机文 件信息,结构中有三个元素,分别用于操作文件名、日期、大小。因为结构中的每个数据成 员能使用不一样的数据类型,所以要对每个数据成员进行数据类型定义。定义好一个结构类 型后,能按下面的格式进行定义结构变量,要注意的是只有结构变量才能参与程序的执 行,结构类型只是用于说明结构变量是属于那一种结构。

struct 结构名 结构变量名 1,结构变量名 2……结构变量 N; 例子:struct FileInfo NewFileInfo, OleFileInfo;

通过上面的定义 NewFileInfo 和 OleFileInfo 都是 FileInfo 结构,都具有一个字符型数组 一个长整型和一个整形数据。定义结构类型只是给出了这个结构的组织形式,它不会占用存 储空间,也就说结构名是不能进行赋值和运算等操作的。结构变量则是结构中的具体成员, 会占用空间,能对每个成员进行操作。

结构是允许嵌套的,也就是说在定义结构类型时,结构的元素能由另一个结构构成。 如:

struct clock

{

unsigned char sec, min, hour;

}

struct date

{

unsigned int year;

unsigned char month, day;

struct clock Time; //这是结构嵌套

}

struct date NowDate; //定义 data 结构变量名为 NowDate

开始学习的朋友看到这可能会发问：“各个数据元素要如何引用、赋值呢？”使用结构变量 时是通过对它的结构元素的引用来实现的。引用的方法是使用存取结构元素成员运算符“.” 来连接结构名和元素名，格式如下：

结构变量名.结构元素

要存取上例结构变量中的月份时，就要写成 NowDate..year。而嵌套的结构，在引用元 素时就要使用多个成员运算符，一级一级连接到最低级的结构元素。要注意的是在 单片机c语言 中只能对最低级的结构元素进行访问，而不可能对整个结构进行操作。操作例子：

NowDate.year = 2005;

NowDate.month = OleMonth+ 2; //月份数据在旧的基础上加 2

NowDate.Time.min++; //分针加 1，嵌套时只能引用最低一级元素 一个结构变量中元素的名字能和程序中其他地方使用的变量同名，因为元素是属于它所在 的结构中，使用时要用成员运算符指定。

结构类型的定义还能有如下的两种格式。

struct

{

结构元素表

} 结构变量名 1，结构变量名 2……结构变量名 N;

例：struct

{

unsigned char FileName[4]; unsigned long Date; unsigned int Size;

} NewFileInfo, OleFileInfo;

这一种定义方式定义没有使用结构名，称为无名结构。通常会用于程序中只有几个确定 的结构变量的场合，不能在其它结构中嵌套。

另一种定义方式如下：

struct 结构名

{

结构元素表

} 结构变量名 1，结构变量名 2……结构变量名 N;

例：struct FileInfo

{

unsigned char FileName[4]; unsigned long Date; unsigned int Size;

} NewFileInfo, OleFileInfo;

使用结构名能便于阅读程序和便于以后要在定义其它结构中使用。 枚举

在程序中经常要用到一些变量去做程序中的判断标志。如经常要用一个字符或整型变量

去储存 1 和 0 做判断条件真假的标志，但我们也许会疏忽这个变量只有当等于 0 或 1 才是有

效的，而将它赋上别的值，而使程序出错或变的混乱。这个时候能使用枚举数据类型去定义变 量，限制错误赋值。枚举数据类型就是把某些整型常量的集合用一个名字表示，其中的整型 常量就是这种枚举类型变量的可取的合法值。枚举类型的二种定义格式如下：

enum 枚举名 {枚举值列表} 变量列表;

例 enum TFFlag {False, True} TFF;

enum 枚举名 {枚举值列表};

emum 枚举名 变量列表;

例 enum Week {Sun, Mon, Tue, Wed, Thu, Fri, Sat};

enum Week OldWeek, NewWeek;

看了上面的例子，你也许有一个地方想不通，那就是为什么枚举值不用贬值就能使 用？那是因为在枚举列表中，每一项名称代表一个整数值，在默认的情况下，编译器会自动 为每一项赋值，第一项赋值为 0，第二项为 1…...如 Week 中的 Sun 为 0，Fri 为 5。C 语言也 允许对各项值做初始化赋值，要注意的是在对某项值初始化后，它的后续的各项值也随之递 增。如：

enum Week {Mon=1, Tue, Wed, Thu, Fri, Sat, Sun};

上例的枚举就使 Week 值从 1 到 7，这样会更符合我们的习惯。使用枚举就如变量一样， 但在程序中不能为其赋值。

联合

联合同样是 C 语言中的构造类型的数据结构。它和结构类型一样能包含不一样类型的 数据元素，所不一样的是联合的数据元素都是从同一个数据地址开始存放。结构变量占用的内 存大小是该结构中数据元素所占内存数的总和，而联合变量所占用内存大小只是该联合中最 长的元素所占用的内存大小。如在结构中定义了一个 int 和一个 char，那么结构变量就会占

用 3 个字节的内存，而在联合中同样定义一个 int 和一个 char，联合变量只会占用 2 个字节。这种能充分利用内存空间的技术叫‘内存覆盖技术’，它能使不一样的变量分时的使用同一 个内存空间。使用联合变量时要注意它的数据元素只能是分时使用，而不能同时使用。举个 简单的例子，程序先为联合中的 int 赋值 1000，后来又为 char 赋值 10,那么这个时候就不能引用

int 了，不然程序会出错，起作用的是最后一次赋值的元素，而上一次赋值的元素就失效了。 使用中还要注意定义联合变量时不能对它的值初始化、能使用指向联合变量的指针对其操 作、联合变量不能作为函数的参数进行传递，数组和结构能出现在联合中。

联合类型变量的定义方法和结构的定义方法差不多，只要把关键字 struct 换用 union 就 能了。联合变量的引用方法除也是使用‘.’成员运算符。

下面就用一个综合的例子说明三种类型的简单使用。

```
#include <AT89X51.H>
#include <stdio.h>
void main(void)
{
enum TF {
False, True} State; //定义一个枚举，使程序更易读
union File { //联合中包含一数组和结构，


unsigned char Str[11]; //整个联合共用 11 个字节内存
struct FN {
unsigned char Name[6],EName[5];} FileName;
} MyFile;
unsigned char Temp;
SCON = 0x50; //串行口方式 1,允许接收
TMOD = 0x20; //定时器 1 定时方式 2
TCON = 0x40; //设定时器 1 开始计数
TH1 = 0xE8;   //11.0592MHz 1200 波特率
TL1 = 0xE8; TI = 1;
TR1 = 1; //启动定时器
State = True; //这里演示 State 只能赋为 False,True 两个值,其它无效
//State = 3;这样是错误的
```

```
printf ("Input File Name 5Byte: \n");
scanf("%s", MyFile.FileName.Name); //保存 5 字节字符串要 6 个字节
printf ("Input File ExtendName 4Byte: \n");
scanf("%s", MyFile.FileName.EName);
if (State == True)
{
printf ("File Name : ");
for (Temp=0; Temp<12; Temp++)
printf ("%c", MyFile.Str[Temp]); //这里列出所有的字节
printf ("\n      Name :");
printf ("%s", MyFile.FileName.Name);
printf ("\n      ExtendName :");
printf ("%s", MyFile.FileName.EName);
}
while(1);
}
```

图 17-1 所示是运行的结果，A 中所示是说明例程中联合中的数组和结构占用的是同一段地址的内存空间，而结构中的两数组是各占两段不一样内存空间。

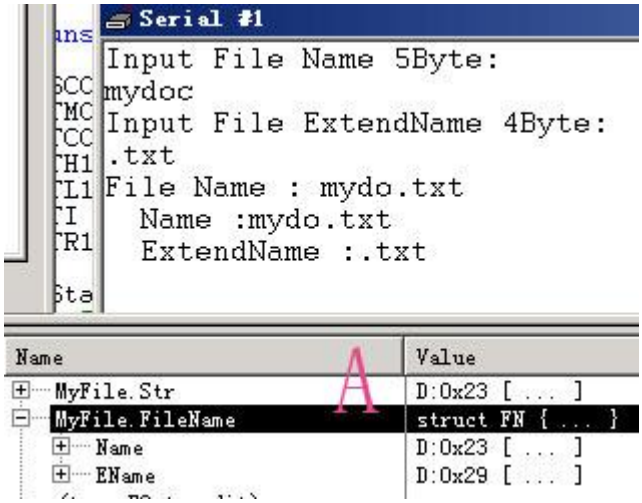


图 17-1

在此简单的单片机C语言教程就结束了，限于作者的水平不能详尽书写。作者本人也是 一名业余的单片机爱好者，希望能和更多相同兴趣的朋友学习交流，读者朋友也能访问网站 http://www.51hei.com 或电邮 51hei@163.com，得到本文相关的更多资讯。本教程所涉及c51源代

码请点此下载

附录一 C51中的关键字

附表1－1 ANSIC标准关键字

关键字	用 途	说 明
auto	存储种类说明	用以说明局部变量，缺省值为此
break	程序语句	退出最内层循环
case	程序语句	Switch语句中的选择项
char	数据类型说明	单字节整型数或字符型数据
const	存储类型说明	在程序执行过程中不可更改的常量值
continue	程序语句	转向下一次循环
default	程序语句	Switch语句中的失败选择项
do	程序语句	构成do..while循环结构
double	数据类型说明	双精度浮点数
else	程序语句	构成if..else选择结构
enum	数据类型说明	枚举
extern	存储种类说明	在其他程序模块中说明了的全局变量
flost	数据类型说明	单精度浮点数
for	程序语句	构成for循环结构
goto	程序语句	构成goto转移结构
if	程序语句	构成if..else选择结构
int	数据类型说明	基本整型数
long	数据类型说明	长整型数
register	存储种类说明	使用CPU内部寄存的变量
return	程序语句	函数返回
short	数据类型说明	短整型数
signed	数据类型说明	有符号数，二进制数据的最高位为符号位
sizeof	运算符	计算表达式或数据类型的字节数
static	存储种类说明	静态变量
struct	数据类型说明	结构类型数据
swicth	程序语句	构成switch选择结构
typedef	数据类型说明	重新进行数据类型定义
union	数据类型说明	联合类型数据
unsigned	数据类型说明	无符号数数据
void	数据类型说明	无类型数据
volatile	数据类型说明	该变量在程序执行中可被隐含地改变
while	程序语句	构成while和do..while循环结构

附表1－2 C51编译器的扩展关键字

关键字	用 途	说 明
bit	位标量声明	声明一个位标量或位类型的函数
sbit	位标量声明	声明一个可位寻址变量
Sfr	特殊功能寄存器声明	声明一个特殊功能寄存器
Sfr16	特殊功能寄存器声明	声明一个16位的特殊功能寄存器
data	存储器类型说明	直接寻址的内部数据存储器
bdata	存储器类型说明	可位寻址的内部数据存储器
idata	存储器类型说明	间接寻址的内部数据存储器
pdata	存储器类型说明	分页寻址的外部数据存储器
xdata	存储器类型说明	外部数据存储器
code	存储器类型说明	程序存储器
interrupt	中断函数说明	定义一个中断函数
reentrant	再入函数说明	定义一个再入函数
using	寄存器组定义	定义芯片的工作寄存器

附录二 AT89C51特殊功能寄存器列表（适用于同一架构的芯片）

符 号	地 址	注 释
*ACC	E0H	累加器
*B	F0H	乘法寄存器
*PSW	D0H	程序状态字
SP	81H	堆栈指针
DPL	82H	数据存储器指针低8位
DPH	83H	数据存储器指针高8位
*IE	A8H	中断允许控制器
*IP	D8H	中断优先控制器
*P0	80H	端口0
*P1	90H	端口1
*P2	A0H	端口2
*P3	B0H	端口3
PCON	87H	电源控制及波特率选择
*SCON	98H	串行口控制器
SBUF	99H	串行数据缓冲器
*TCON	88H	定时器控制
TMOD	89H	定时器方式选择
TL0	8AH	定时器0低8位
TL1	8BH	定时器1低8位
TH0	8CH	定时器0低8位
TH1	8DH	定时器1高8位

带*号的特殊功能寄存器都是可以位寻址的寄存器

附录三 运算符优先级和结合性

级 别	类 别	名 称	运算符	结合性
1	强制转换、数组、结构、联合	强制类型转换	( )	右结合
		下标	[ ]	
		存取结构或联合成员	->或.	
2	逻 辑	逻辑非	!	左结合
	字 位	按位取反	~	
	增 量	加一	++	
	减 量	减一	--	
	指 针	取地址	&	
	取内容	*		
	算 术	单目减	-	
	长度计算	长度计算	sizeof	
3	算 术	乘	*	右结合
		除	/	
		取模	%	
4	算术和指针运算	加	+	
		减	-	
5	字 位	左移	<<	
		右移	>>	
6	关系	大于等于	>=	
		大于	>	
		小于等于	<=	
		小于	<	
7		恒等于	==	
		不等于	!=	
8	字 位	按位与	&	
9		按位异或	^	
10		按位或	|	
11	逻 辑	逻辑与	&&	左结合
12		逻辑或	||	
13	条 件	条件运算	?:	
14	赋 值	赋值	=	右结合
		复合赋值	Op=	
15	逗 号	逗号运算	,	右结合

附录四 字符串定时常用的转义字符表

转义字符	含义	ASCII码（16/10进制）
\o	空字符(NULL)	00H/0
\n	换行符(LF)	0AH/10
\r	回车符(CR)	0DH/13
\t	水平制表符(HT)	09H/9
\b	退格符(BS)	08H/8
\f	换页符(FF)	0CH/12
\'	单引号	27H/39
\"	双引号	22H/34
\\	反斜杠	5CH/92

附录五 C语言的数据类型

数据类型	长 度	值 域
unsigned char	单字节	0～255
signed char	单字节	-128～+127
unsigned int	双字节	0～65535
signed int	双字节	-32768～+32767
unsigned long	四字节	0～4294967295
signed long	四字节	-2147483648～+2147483647
float	四字节	±1.175494E-38～±3.402823E+38
*	1～3字节	对象的地址
bit	位	0或1
sfr	单字节	0～255
sfr16	双字节	0～65535
sbit	位	0或1