51 单片机 C 语言学习杂记

学习单片机实在不是件易事，一来要购买高价格的编程器，仿真器，二来要学习编程语言，还有众多种类的单片机选择真是件让人头痛的事。在众多单片机中 51 架构的芯片风行很久，学习资料也相对很多，是初学的较好的选择之一。51 的编程语言常用的有二种，一种是汇编语言，一种是 C 语言。汇编语言的机器代码生成效率很高但可读性却并不强，复杂一点的程序就更是难读懂，而 C 语言在大多数情况下其机器代码生成效率和汇编语言相当，但可读性和可移植性却远远超过汇编语言，而且 C 语言还可以嵌入汇编来解决高时效性的代码编写问题。对于开发周期来说，中大型的软件编写用 C 语言的开发周期通常要小于汇编语言很多。综合以上 C 语言的优点，我在学习时选择了 C 语言。以后的教程也只是我在学习过程中的一些学习笔记和随笔，在这里加以整理和修改，希望和大家一起分享，一起交流，一起学习，一起进步。

*注：可以肯定的说这个教程只是为初学或入门者准备的，笔者本人也只是菜鸟一只，有望各位大侠高手指点错误提出建议。

明浩　2003-3-30　pnzwzw@163.com

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，您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。或许您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！那您要从头学电脑了。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。

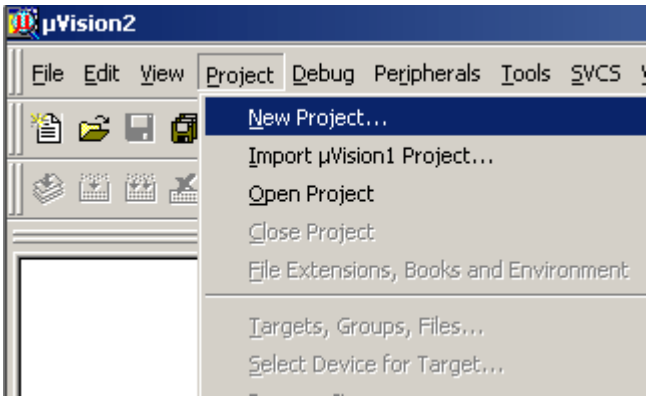

图 1－2　New Project 菜单

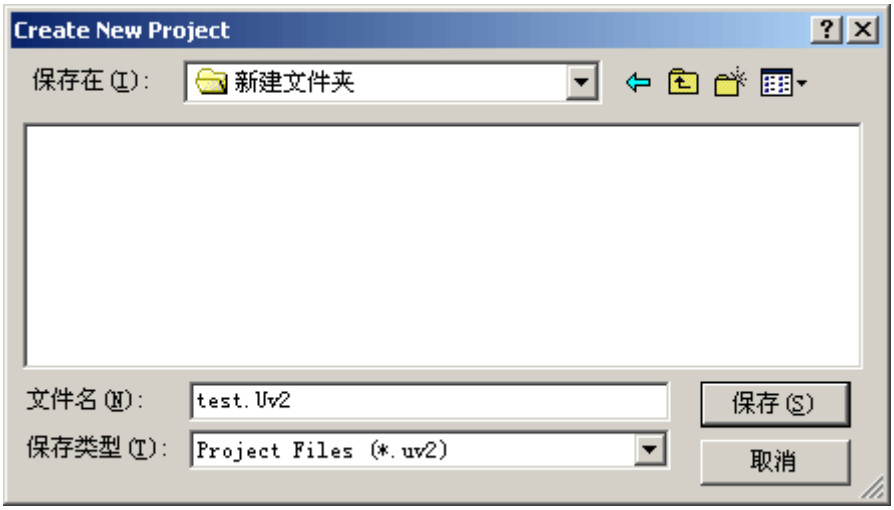

图 1－3　文件窗口

（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。

（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。如果您没有现成的程序，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，如果您看过别的程序书也许也有类似的程序：

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

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
        }
}
```

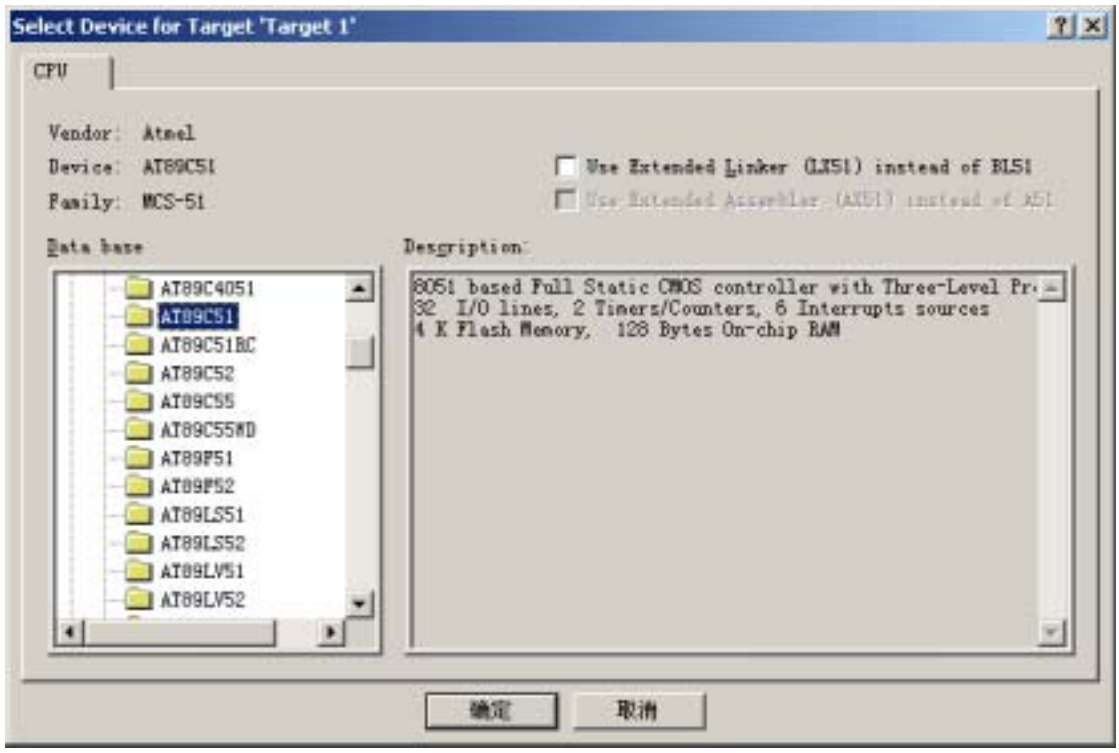

图 1－4 选取芯片

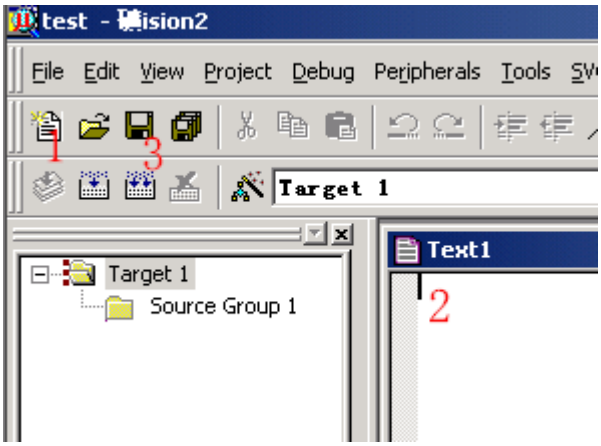

图 1－5 新建程序文件

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，这时您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。我们选“ 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。

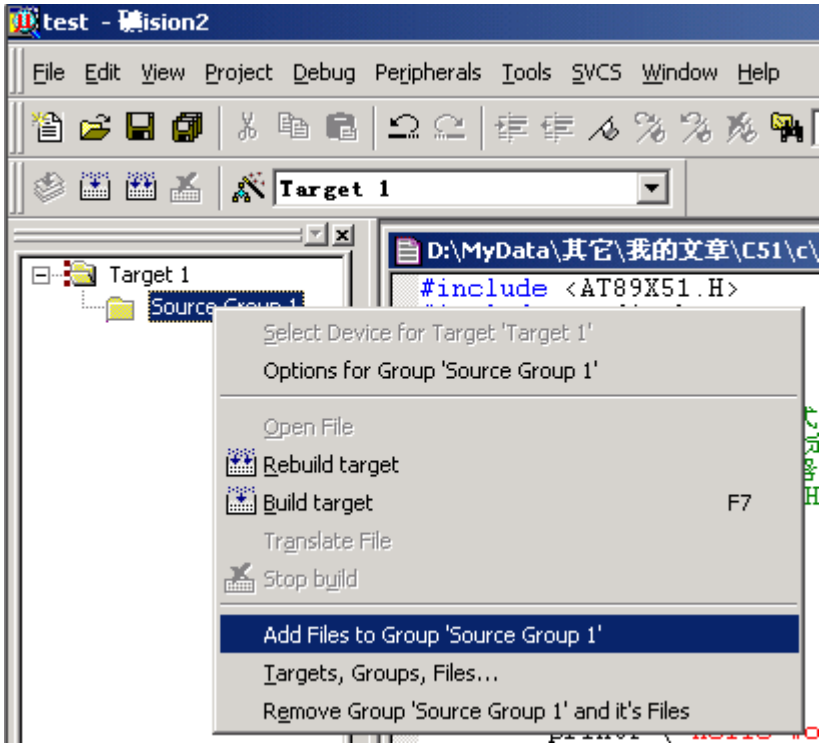

图 1－6 把文件加入到项目文件组中

（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，您按 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。

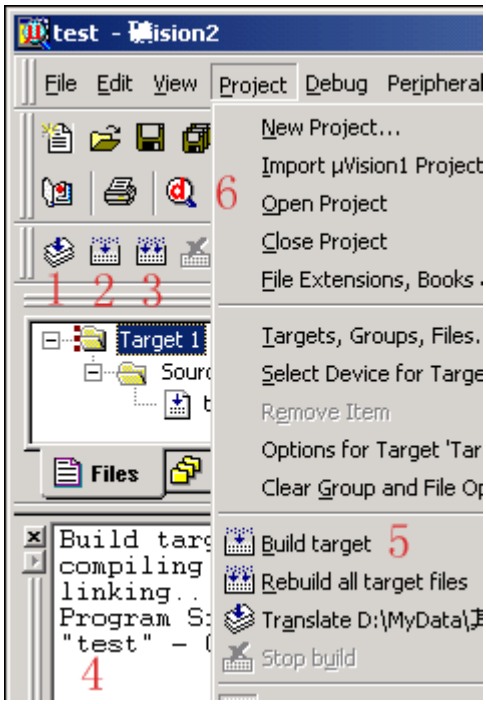

图 1－7 编译程序

(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印“ 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。

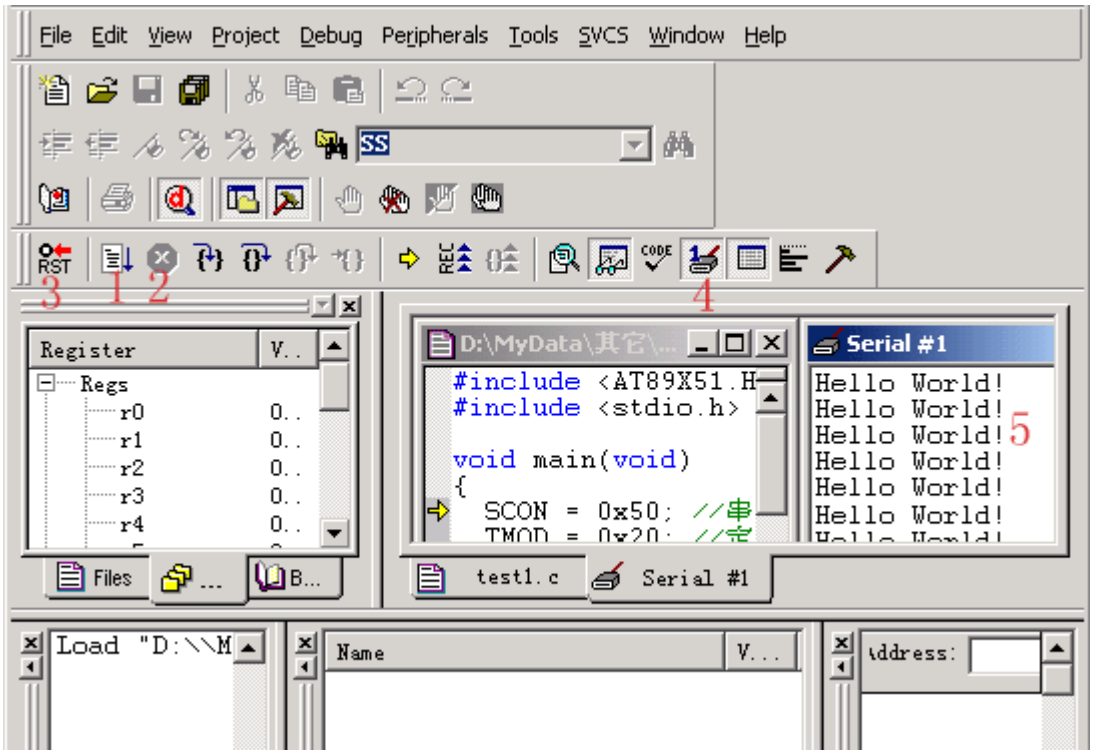

图 1－8 调试运行程序

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，如要对硬件编程您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

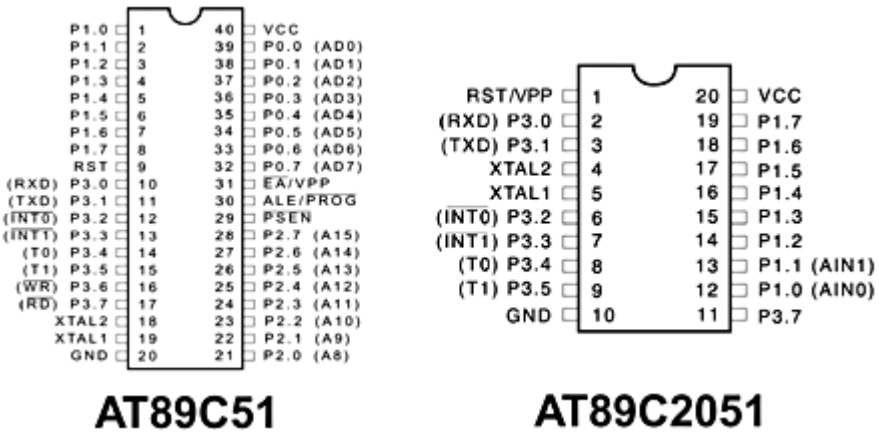

图 2－1　AT89C51 和 AT89C2051 引脚功能图

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

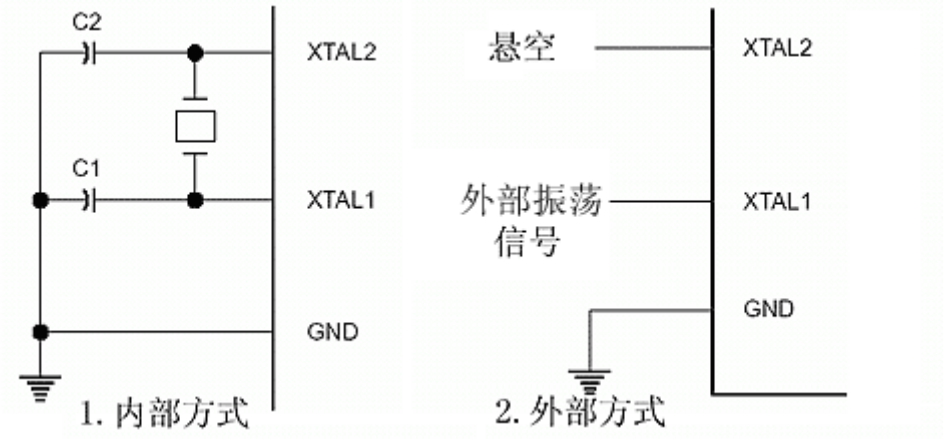

图 2－2　外接晶体引脚

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。

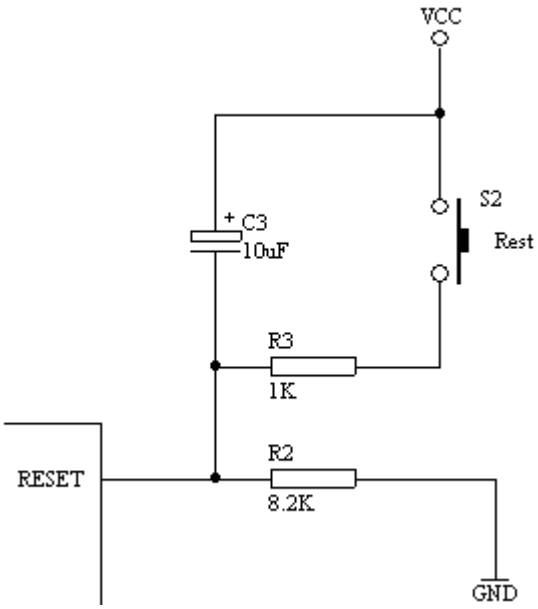

图 2－3　常用复位电路

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（ $\overline{\text{INT0}}$）

P3.3	外部中断 1（$\overline{INT1}$）
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 $\overline{WR}$
P3.7	外部数据存储器写选通 $\overline{RD}$

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/$\overline{PROG}$ 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) $\overline{PSEN}$ 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) $\overline{EA}$/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	030H=1EH	030H=1EH	030H=1EH
	031H=51H	031H=61H	031H=51H	031H=61H
	032H=FFH	032H=FFH	032H=05H	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，除非您想自己开发编程器。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。）

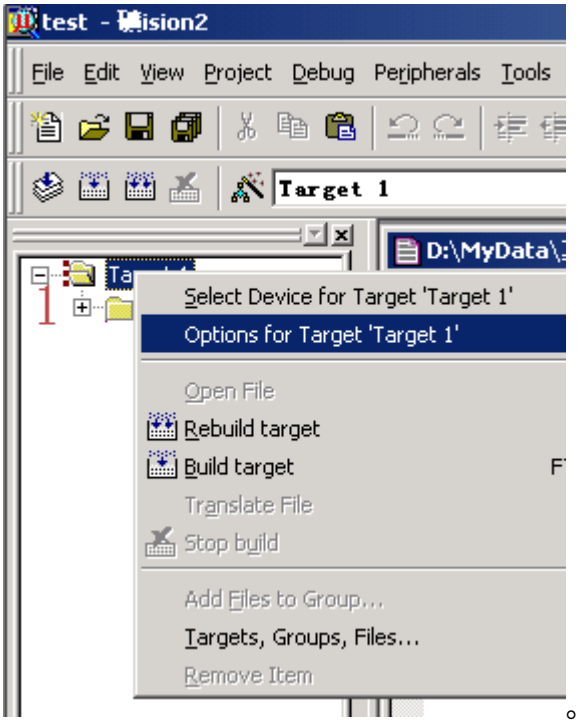

图3－1项目功能菜单

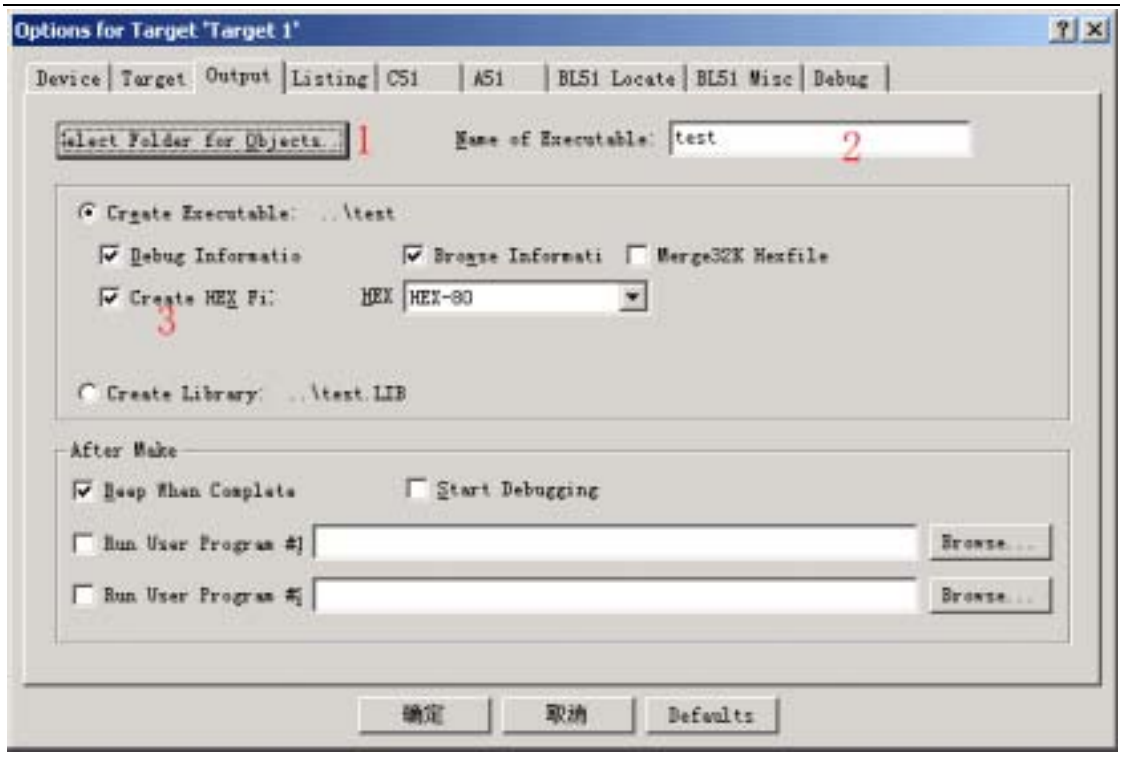

图3－2　项目选项窗口

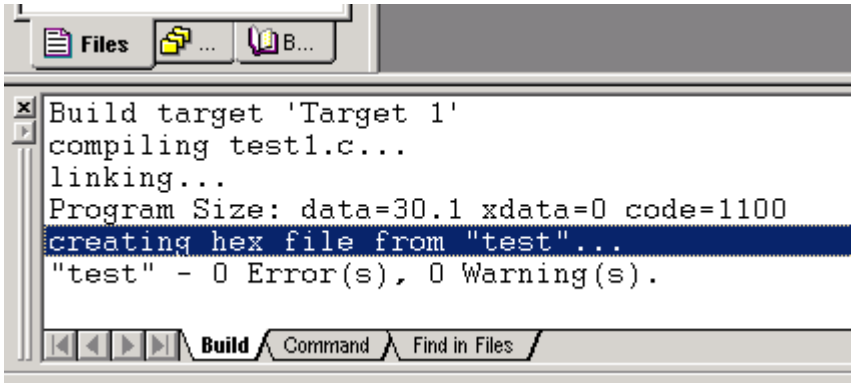

图 3－3　编译信息窗口

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 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://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);
}
```

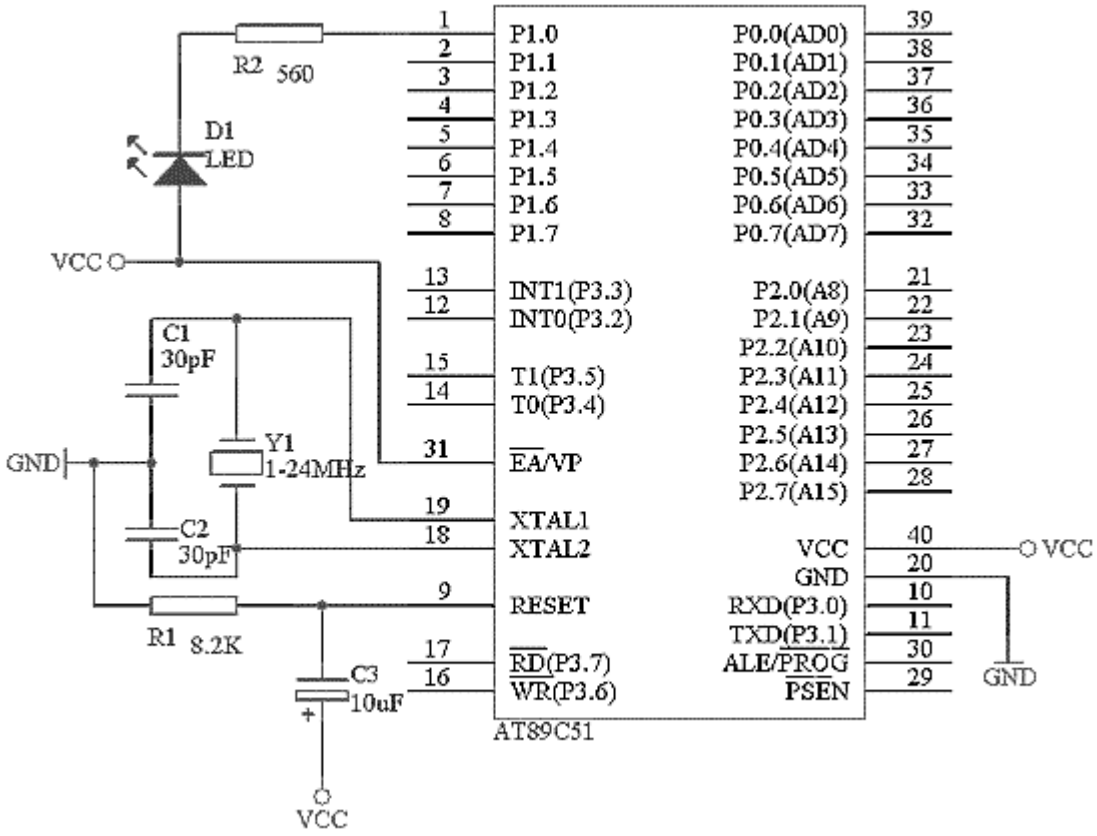

图 3－4　AT89C51 最小化系统

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相同。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，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，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。

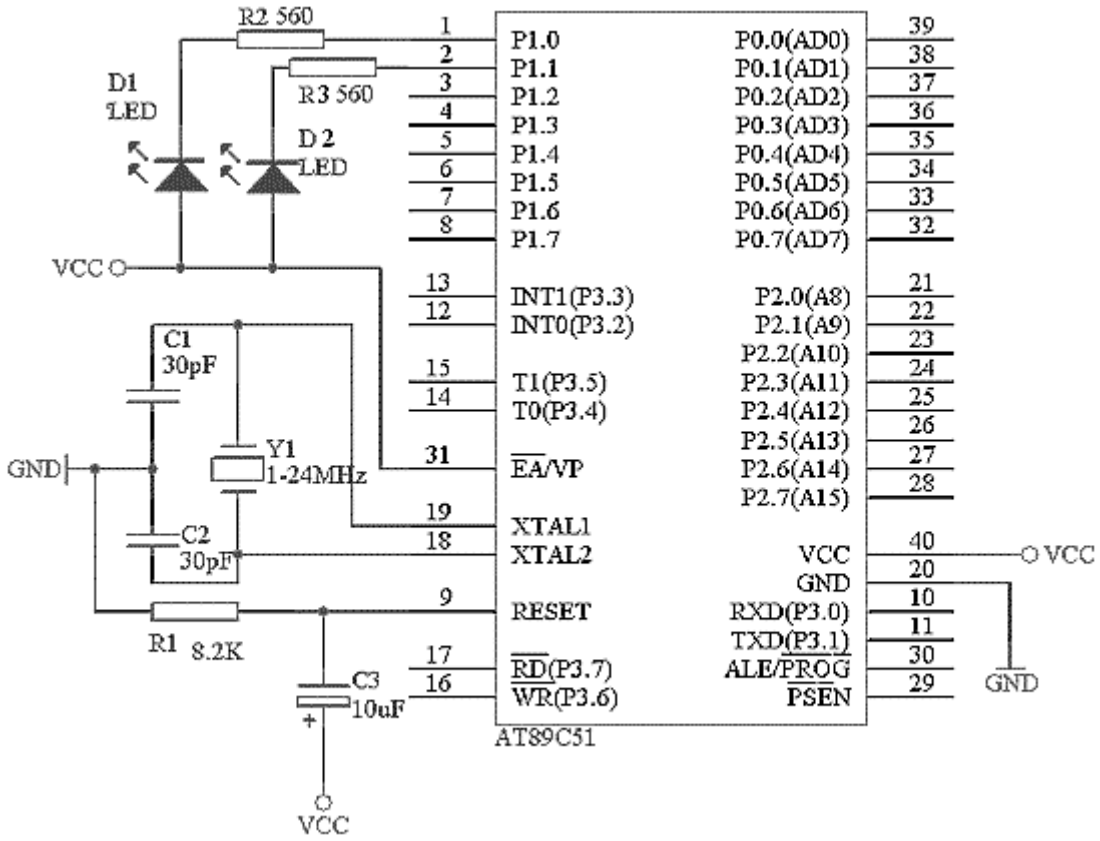

图 4－1　第 4 课实验用电路

我们把这个项目称为 TwoLED,实验程序如下：

```
#include <AT89X51.h> //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，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，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 为 P1 端口在片内的寄存器，在后面的语句中我们用以用 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，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 可录址位

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。

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 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。

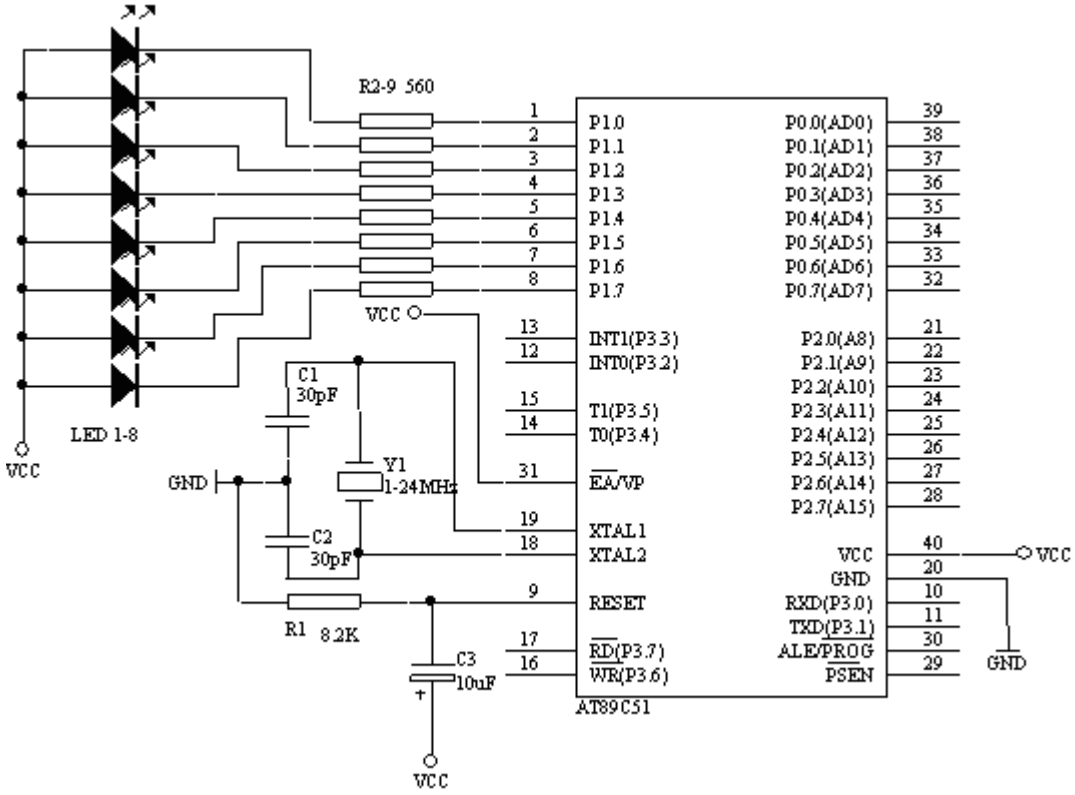

图 5－1　八路跑马灯电路

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。我们在“ 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。

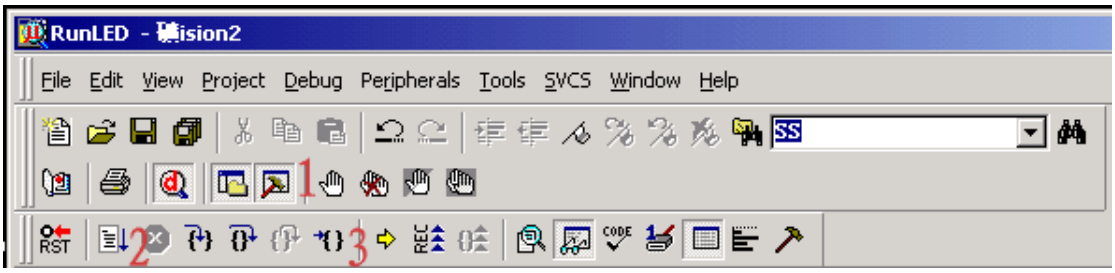

图 5－2　调试用快捷菜单栏

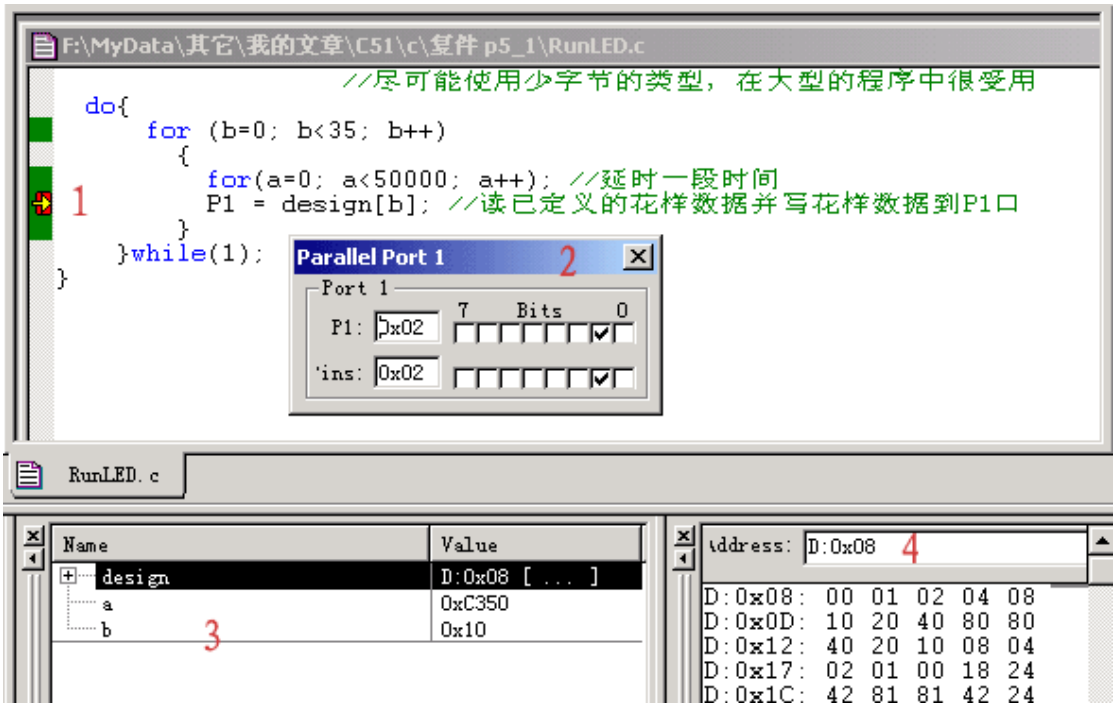

图 5－3　各调试窗口

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 = 0x91;
```

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。

++ 增量运算符
-- 减量运算符

这两个运算符是 C 语言中特有的一种运算符。在 VB，PASCAL 等都是没有的。作用就是对运算对象作加 1 和减 1 运算。要注意的是运算对象在符号前或后，其含义都是不同的，虽然同是加 1 或减 1。如：I++，++I，I--，--I。

I++（或 I--） 是先使用 I 的值，再执行 I+1（或 I-1）

++I（或--I） 是先执行 I+1（或 I-1），再使用 I 的值。

增减量运算符只允许用于变量的运算中，不能用于常数或表达式。

:(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。

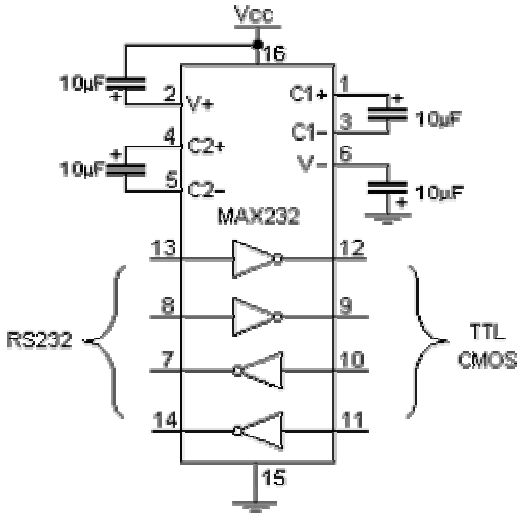

图 7－1　MAX232

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

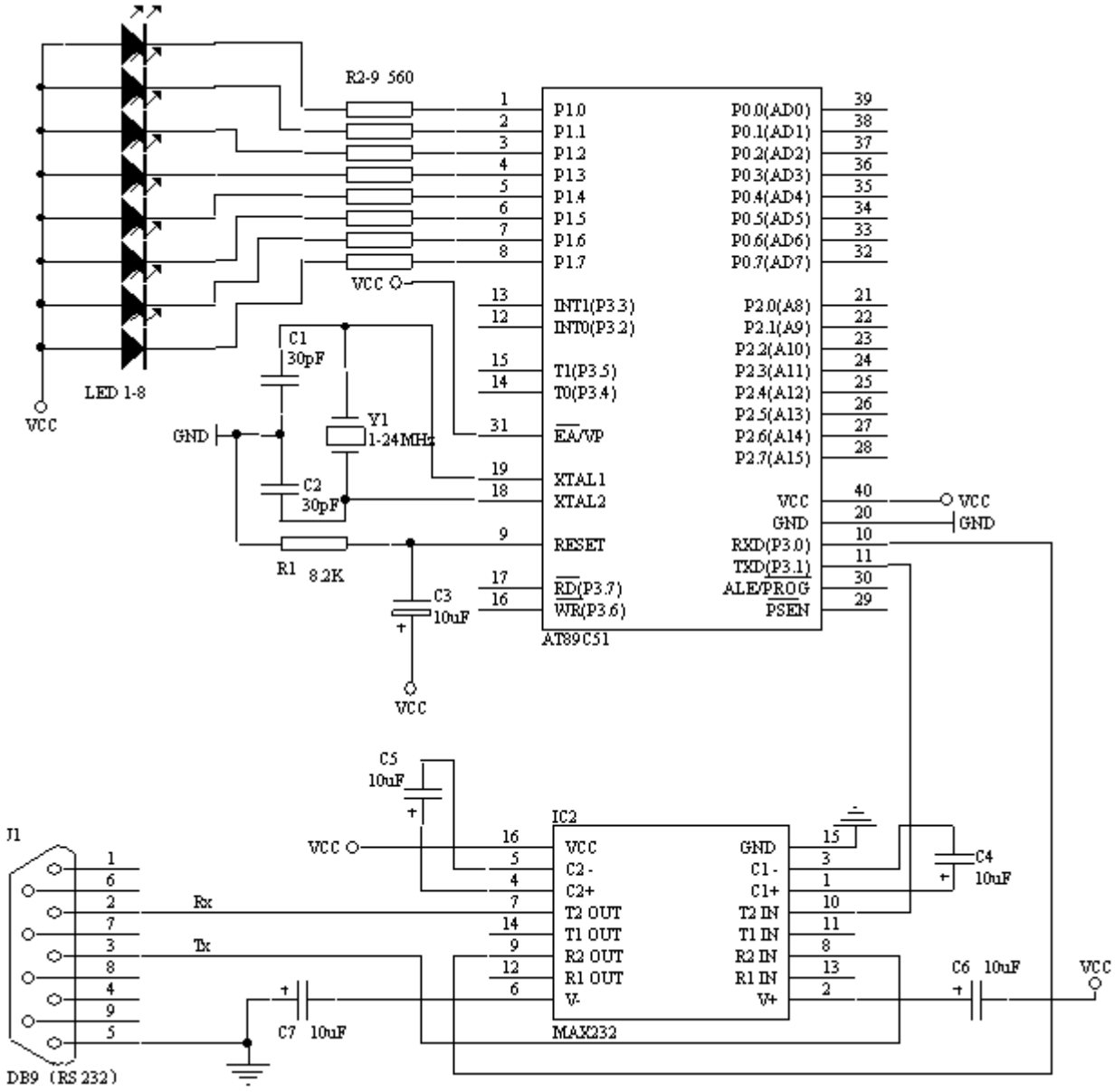

图 7－3　加上了 MAX232 的实验电路

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。

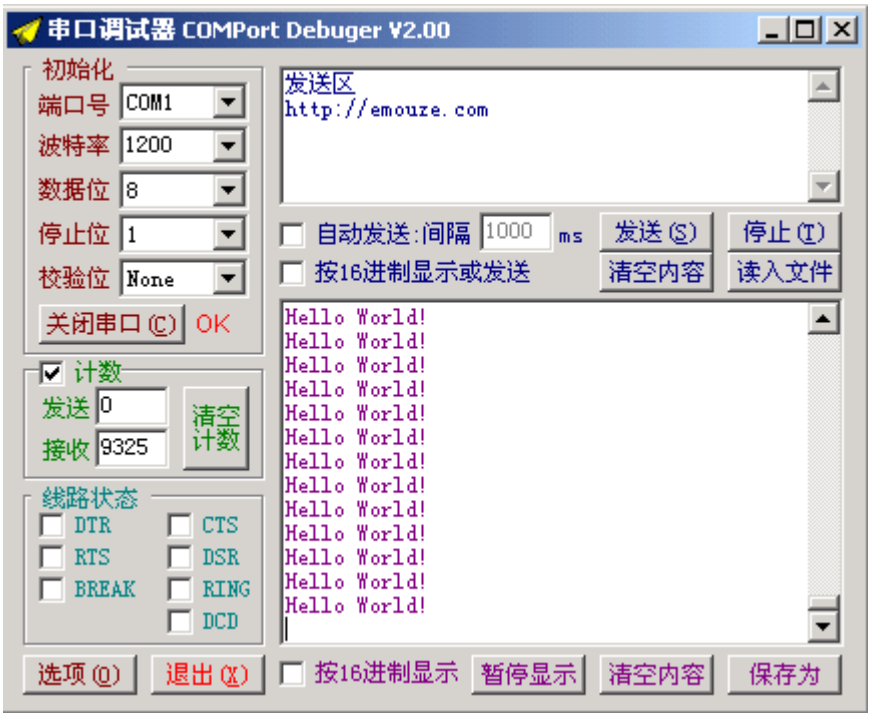

图 7-4　调试结果

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。也许你会对为什么“ 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 <at89x51.h>
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｜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 0 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<b)
    min = a;
else
    min = b; //这一段的意思是当 a<b 时 min 的值为 a 的值，否则为 b 的值。
```

用条件运算符去构成条件表达式就变得简单明了了：

```
min = (a<b)?a : b
```

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 <at89x51.h>
#include <stdio.h>

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。

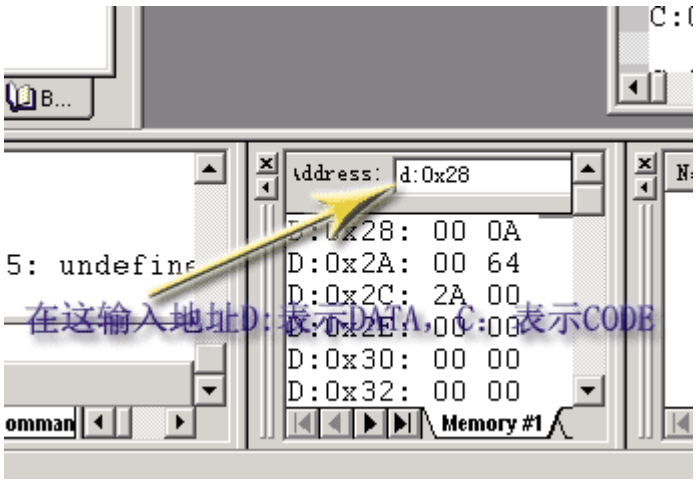

图 7－6　存储器查看窗

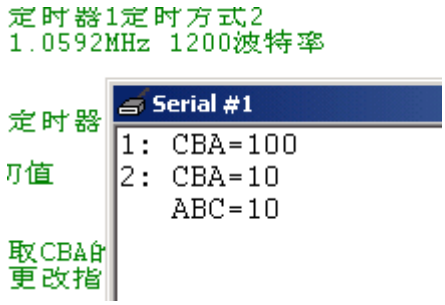

图 7－7　在串行调试窗口的最终结果

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：

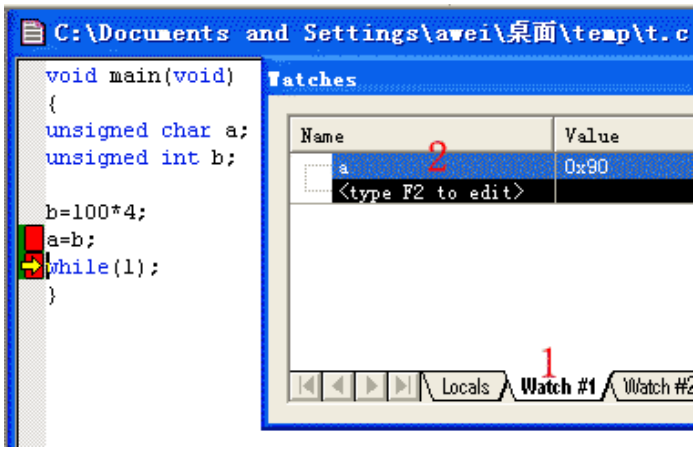

图 7－8　小程序的运行情况

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 <at89x51.h>
#include <stdio.h>

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。

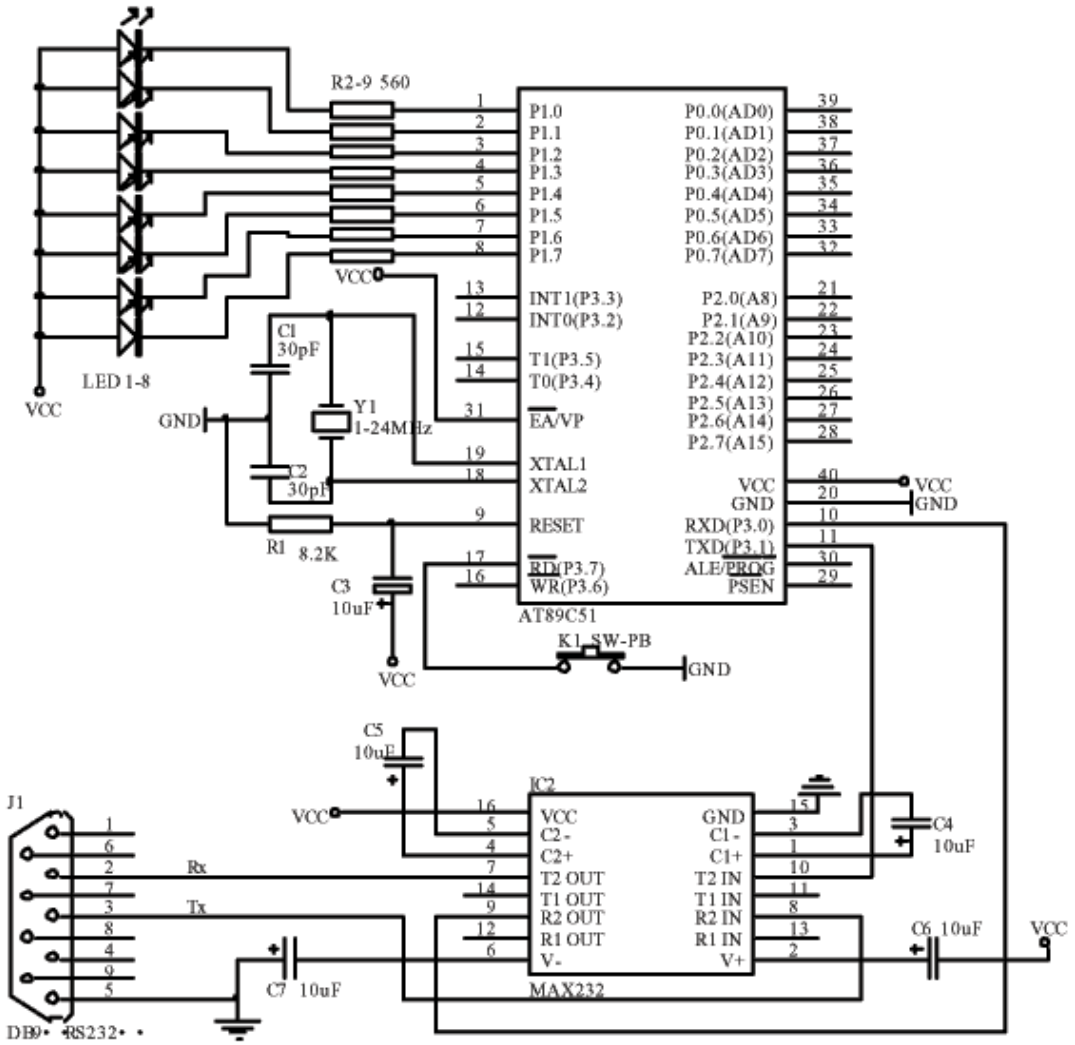

图 8-1 加了按键的实验电路图

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
}
```

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，使用应注意 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。

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)
{
    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。

相关文章请看 http://www.cdle.net/alldata/mywz/04032401_1.htm