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。

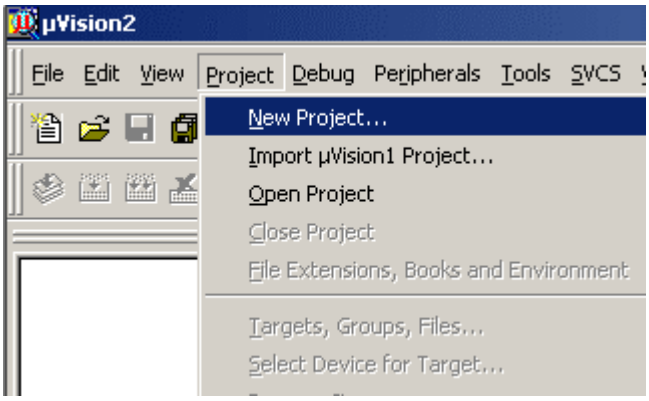

图 1－2　New Project 菜单

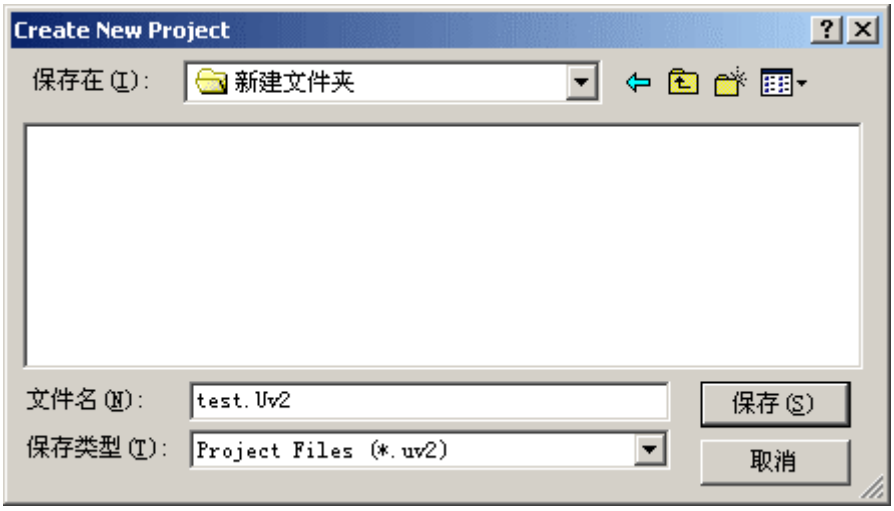

图 1－3　文件窗口

（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
        }
}
```

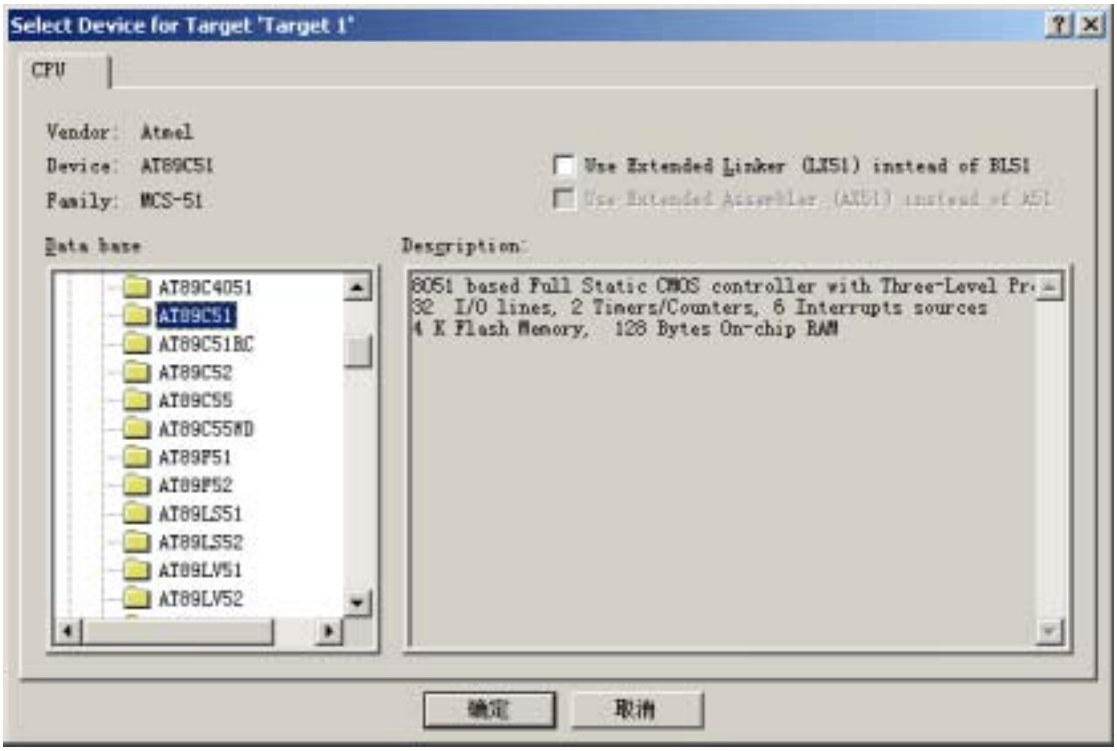

图 1－4 选取芯片

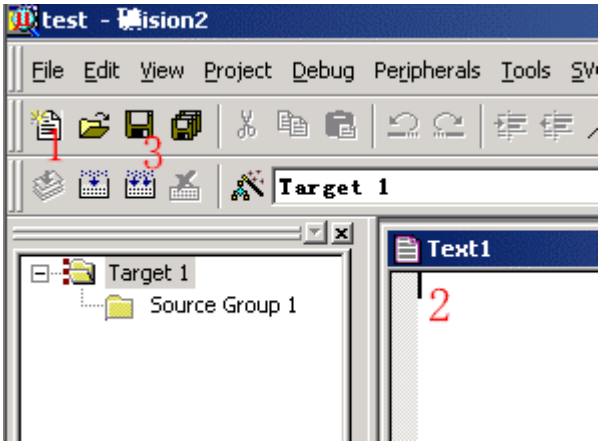

图 1－5 新建程序文件

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。

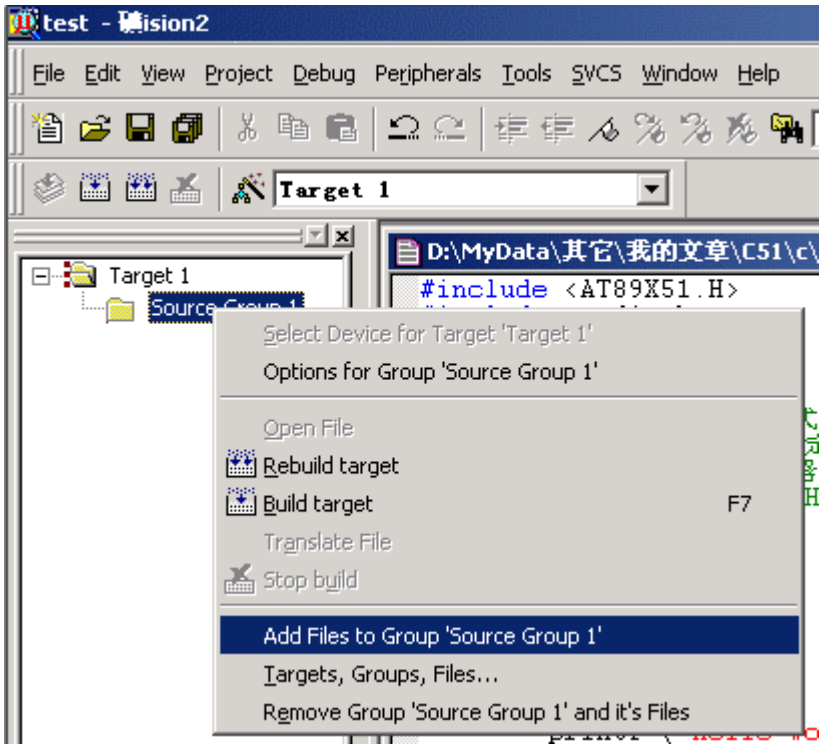

图 1－6 把文件加入到项目文件组中

（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。

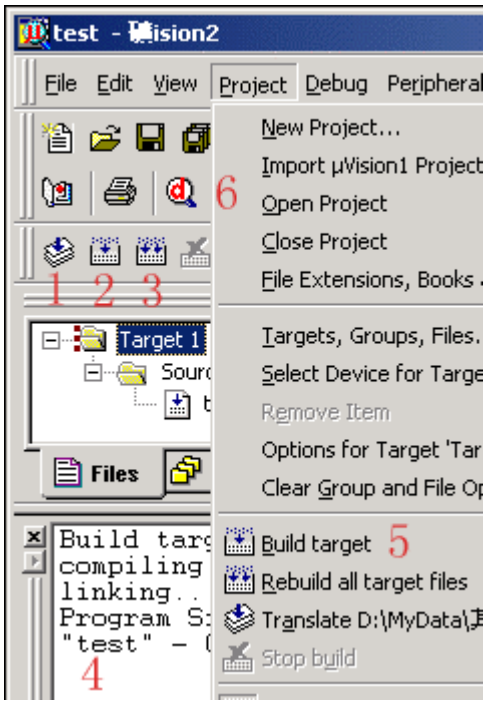

图 1－7 编译程序

(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。

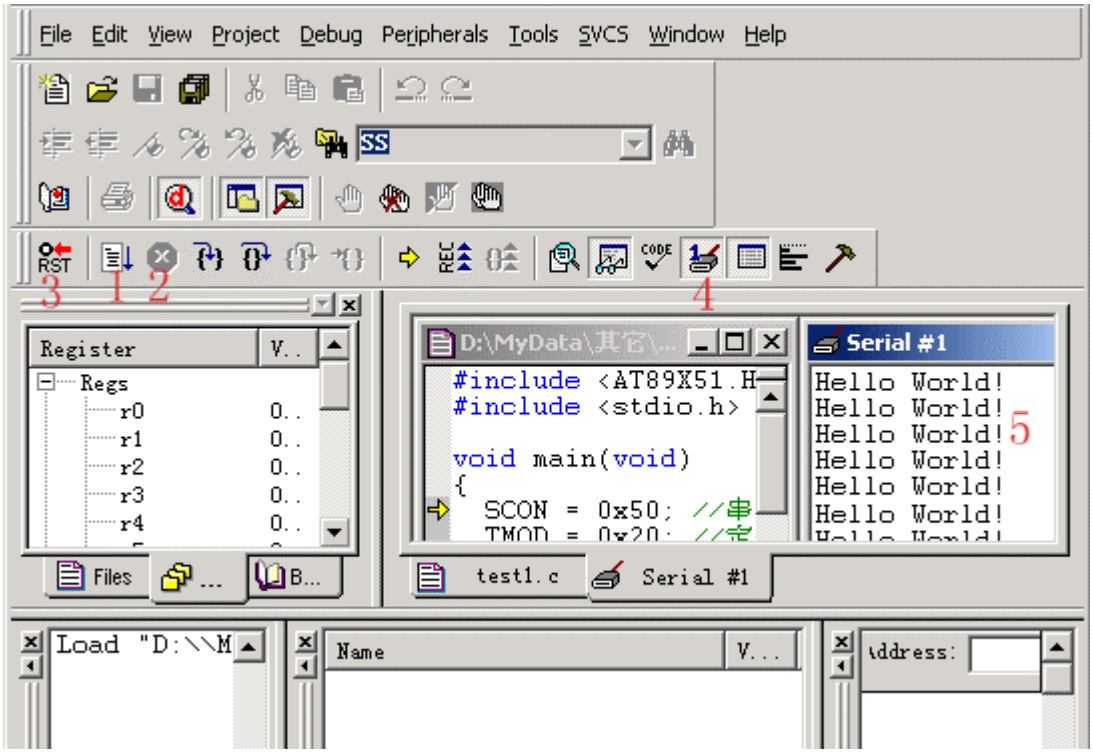

图 1－8 调试运行程序

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

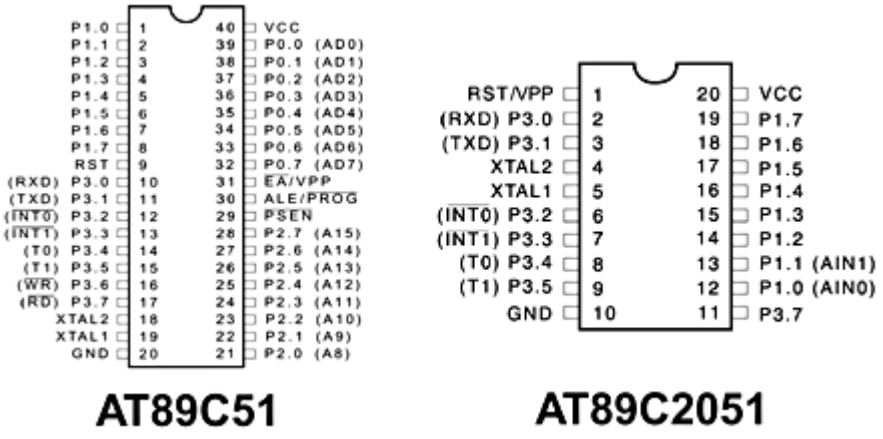

图 2－1　AT89C51 和 AT89C2051 引脚功能图

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

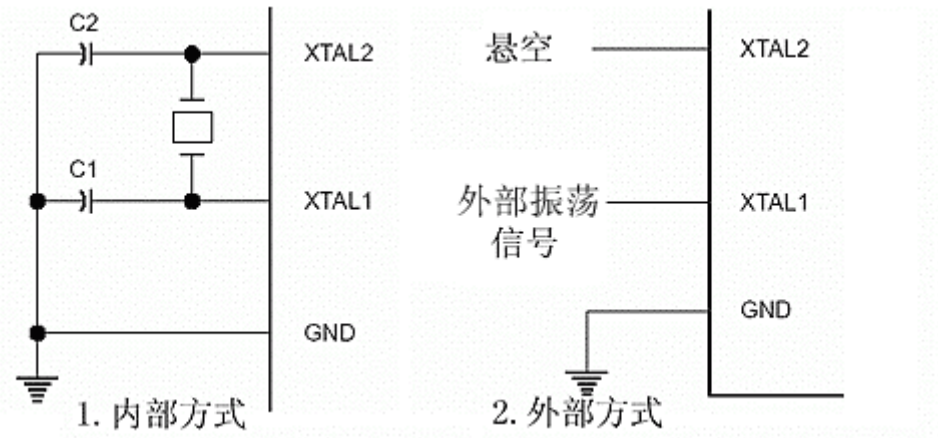

图 2－2　外接晶体引脚

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。

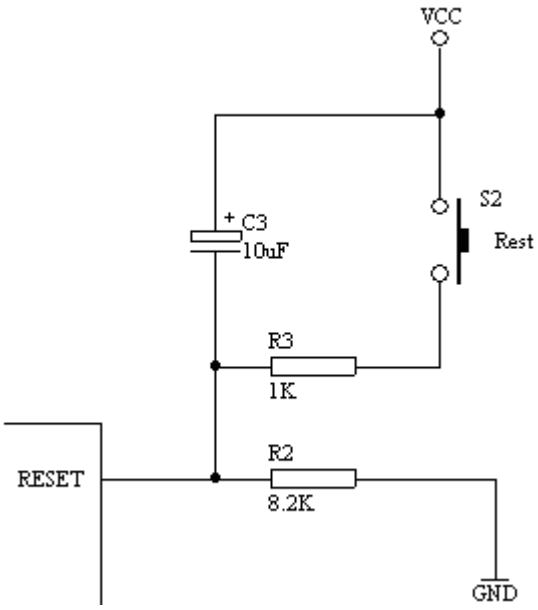

图 2－3 常用复位电路

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

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/$\overline{\text{PROG}}$ 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) $\overline{\text{PSEN}}$ 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) $\overline{\text{EA}}$/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。对 Flash 存储器编程时，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。）

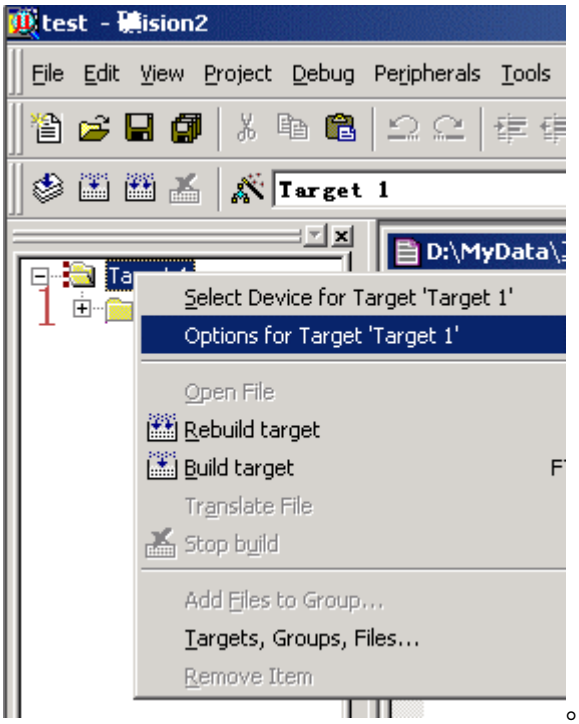

图3－1项目功能菜单

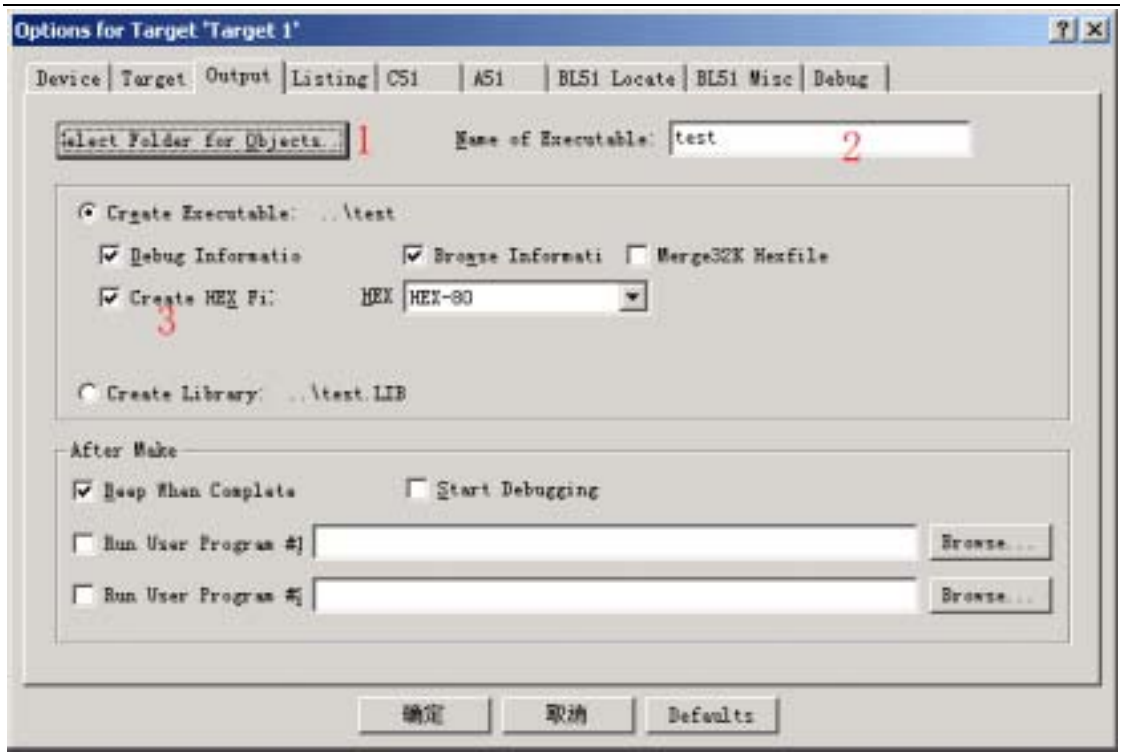

图3－2　项目选项窗口

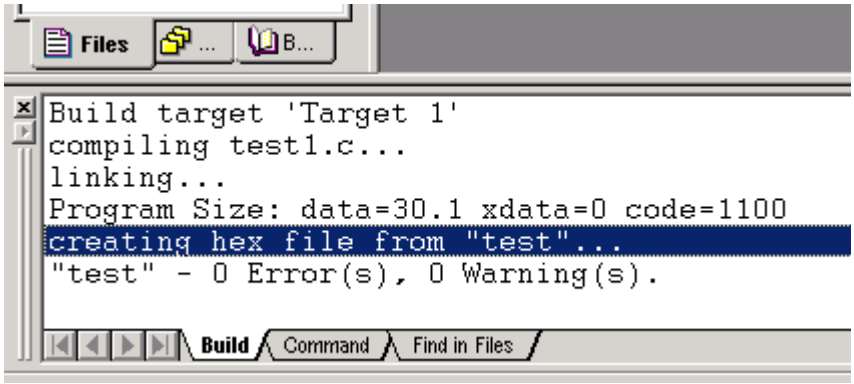

图 3－3　编译信息窗口

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://cdle.yeah.net 或 http://cdle.126.com ）。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);
}
```

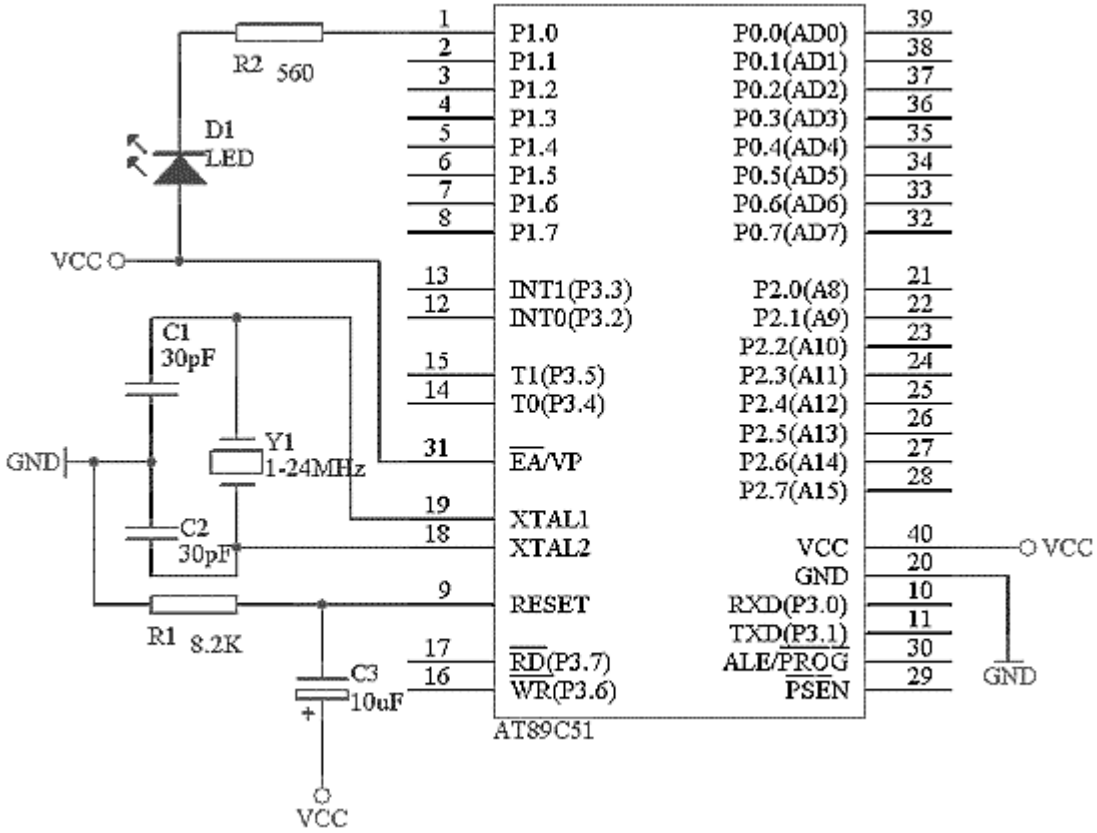

图 3－4　AT89C51 最小化系统

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。

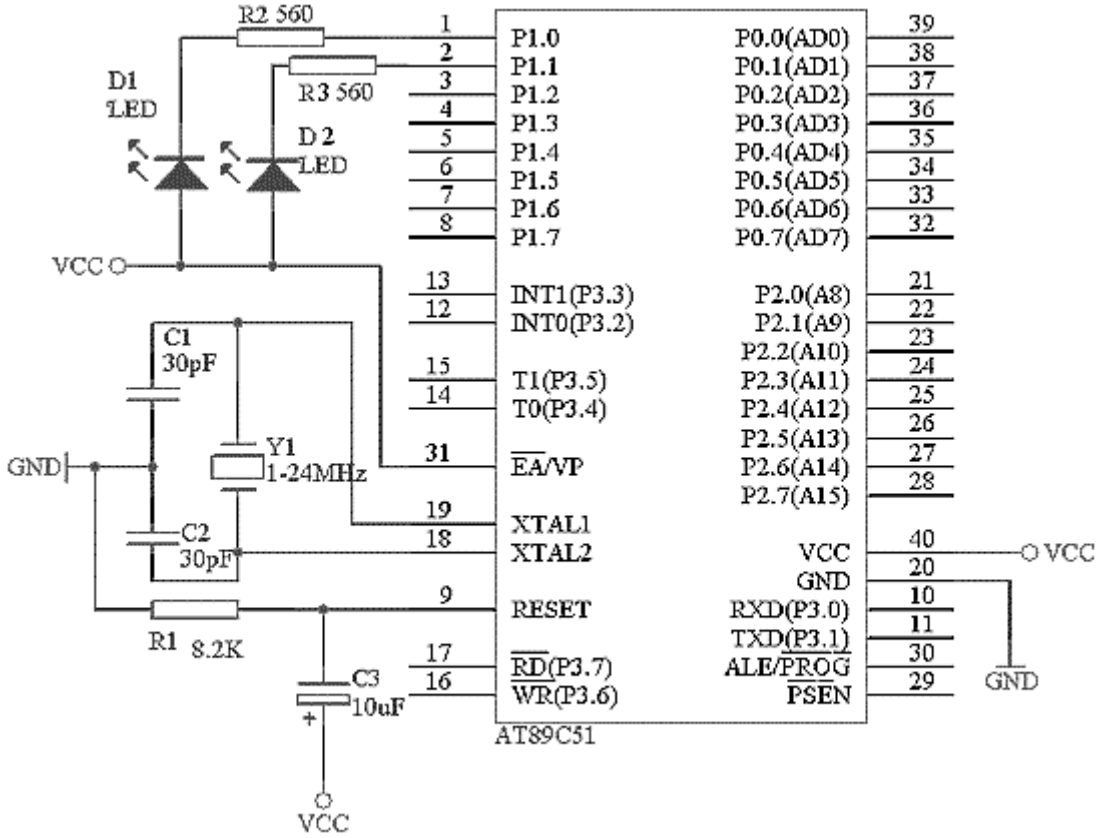

图 4－1　第 4 课实验用电路

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
  cons tunsigned 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。

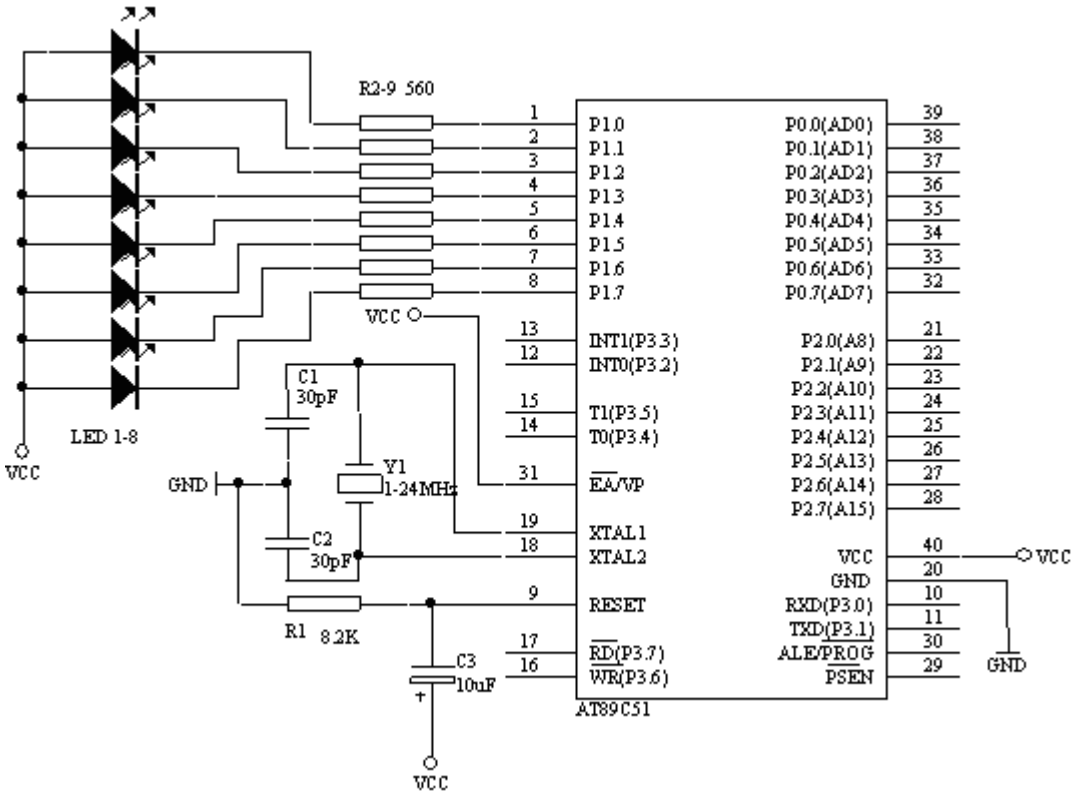

图 5－1　八路跑马灯电路

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 code 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。

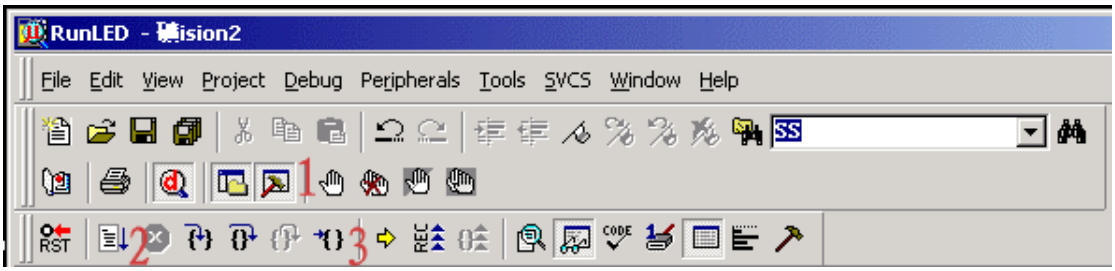

图 5－2　调试用快捷菜单栏

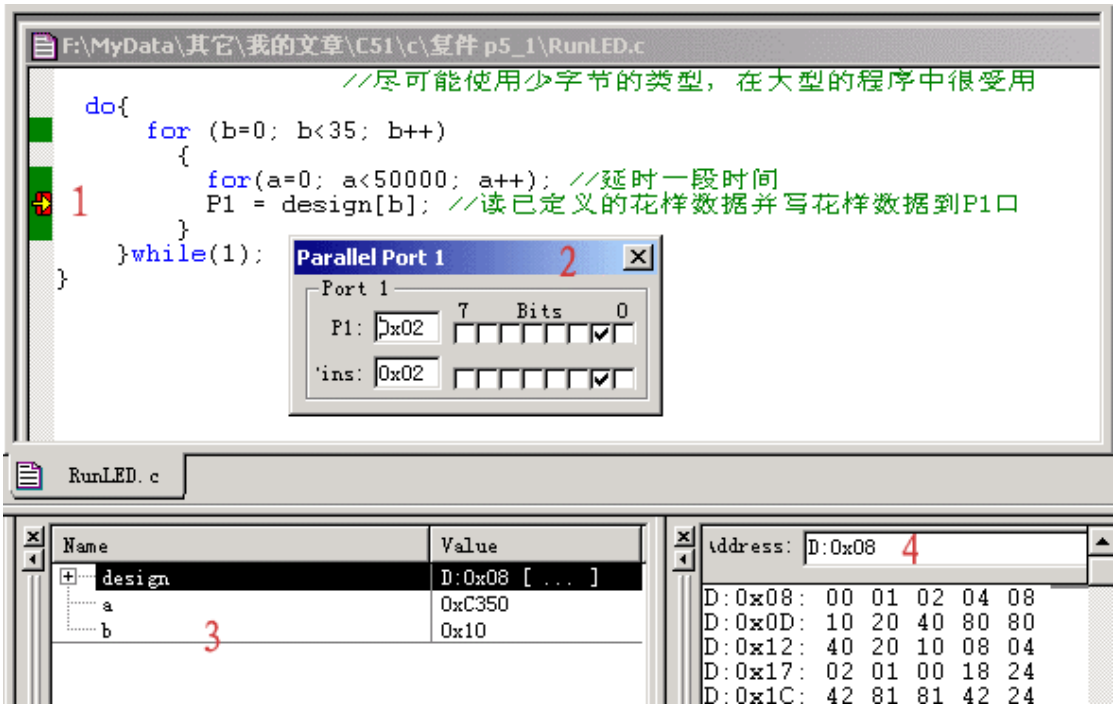

图 5－3　各调试窗口

Keil 的调试命令、在线汇编与断点设置

上一讲中我们学习了如何建立工程、汇编、连接工程，并获得目标代码，但是做到这一步仅仅代表你的源程序没有语法错误，至于源程序中存在着的其它错误，必须通过调试才能发现并解决，事实上，除了极简单的程序以外，绝大部份的程序都要通过反复调试才能得到正确的结果，因此，调试是软件开发中重要的一个环节，这一讲将介绍常用的调试命令、利用在线汇编、各种设置断点进行程序调试的方法，并通过实例介绍这些方法的使用。

一、常用调试命令

在对工程成功地进行汇编、连接以后，按 Ctrl+F5 或者使用菜单 Debug->Start/Stop Debug Session 即可进入调试状态，Keil 内建了一个仿真 CPU 用来模拟执行程序，该仿真 CPU 功能强大，可以在没有硬件和仿真机的情况下进行程序的调试，下面将要学的就是该模拟调试功能。不过在学习之前必须明确，模拟毕竟只是模拟，与真实的硬件执行程序肯定还是有区别的，其中最明显的就是时序，软件模拟是不可能和真实的硬件具有相同的时序的，具体的表现就是程序执行的速度和各人使用的计算机有关，计算机性能越好，运行速度越快。

进入调试状态后，界面与编缉状态相比有明显的变化，Debug 菜单项中原来不能用的命令现在已可以使用了，工具栏会多出一个用于运行和调试的工具条，如图 1 所示，Debug 菜单上的大部份命令可以在此找到对应的快捷按钮，从左到右依次是复位、运行、暂停、单步、过程单步、执行完当前子程序、运行到当前行、下一状态、打开跟踪、观察跟踪、反汇编窗口、观察窗口、代码作用范围分析、1＃串行窗口、内存窗口、性能分析、工具按钮等命令。

图 1 调试工具条

学习程序调试，必须明确两个重要的概念，即单步执行与全速运行。全速执行是指一行程序执行完以后紧接着执行下一行程序，中间不停止，这样程序执行的速度很快，并可以看到该段程序执行的总体效果，即最终结果正确还是错误，但如果程序有错，则难以确认错误出现在哪些程序行。单步执行是每次执行一行程序，执行完该行程序以后即停止，等待命令执行下一行程序，此时可以观察该行程序执行完以后得到的结果，是否与我们写该行程序所想要得到的结果相同，借此可以找到程序中问题所在。程序调试中，这两种运行方式都要用到。

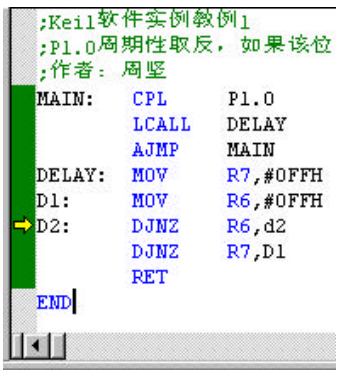

图 2 调试窗口

使用菜单 STEP 或相应的命令按钮或使用快捷键 F11 可以单步执行程序，使用菜单 STEP OVER 或功能键 F10 可以以过程单步形式执行命令，所谓过程单步，是指将汇编语言中的子程序或高级语言中的函数作为一个语句来全速执行。

按下 F11 键，可以看到源程序窗口的左边出现了一个黄色调试箭头，指向源程序的第一行，如图 2 所示。每按一次 F11，即执行该箭头所指程序行，然后箭头指向下一行，当箭头指向 LCALL　DELAY 行时，再次按下 F11，会发现，箭头指向了延时子程序 DELAY 的第一行。不断按 F11 键，即可逐步执行延时子程序。

通过单步执行程序，可以找出一些问题的所在，但是仅依靠单步执行来查错有时是困难的，或虽能查出错误但效率很低，为此必须辅之以其它的方法，如本例中的延时程序是通过

将 D2： DJNZ R6,D2 这一行程序执行六万多次来达到延时的目的，如果用按 F11 六万多次的方法来执行完该程序行，显然不合适，为此，可以采取以下一些方法，第一，用鼠标在子程序的最后一行（ret）点一下，把光标定位于该行，然后用菜单 Debug->Run to Cursor line（执行到光标所在行），即可全速执行完黄色箭头与光标之间的程序行。第二，在进入该子程序后，使用菜单 Debug->Step Out of Current Function（单步执行到该函数外），使用该命令后，即全速执行完调试光标所在的子程序或子函数并指向主程序中的下一行程序（这里是 JMP LOOP 行）。第三种方法，在开始调试的，按 F10 而非 F11，程序也将单步执行，不同的是，执行到 lcall delay 行时，按下 F10 键，调试光标不进入子程序的内部，而是全速执行完该子程序，然后直接指向下一行“ JMP LOOP ”。灵活应用这几种方法，可以大大提高查错的效率。

二、在线汇编

在进入 Keil 的调试环境以后，如果发现程序有错，可以直接对源程序进行修改，但是要使修改后的代码起作用，必须先退出调试环境，重新进行编译、连接后再次进入调试，如果只是需要对某些程序行进行测试，或仅需对源程序进行临时的修改，这样的过程未免有些麻烦，为此 Keil 软件提供了在线汇编的能力，将光标定位于需要修改的程序行上，用菜单 Debug->Inline Assambly…即可出现如图 3 的对话框，在 Enter New 后面的编辑框内直接输入需更改的程序语句，输入完后键入回车将自动指向下一条语句，可以继续修改，如果不再需要修改，可以点击右上角的关闭按钮关闭窗口。

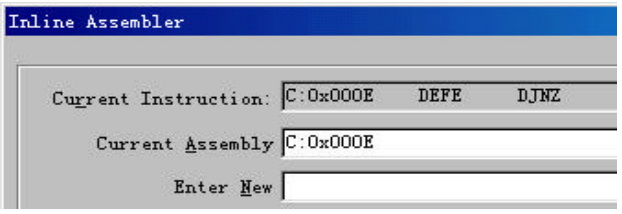

图 3 在线汇编窗口

三、断点设置

程序调试时，一些程序行必须满足一定的条件才能被执行到（如程序中某变量达到一定的值、按键被按下、串口接收到数据、有中断产生等），这些条件往往是异步发生或难以预先设定的，这类问题使用单步执行的方法是很难调试的，这时就要使用到程序调试中的另一种非常重要的方法——断点设置。断点设置的方法有多种，常用的是在某一程序行设置断点，设置好断点后可以全速运行程序，一旦执行到该程序行即停止，可在此观察有关变量值，以确定问题所在。在程序行设置/移除断点的方法是将光标定位于需要设置断点的程序行，使用菜单 Debug->Insert/Remove BreakPoint 设置或移除断点（也可以用鼠标在该行双击实现同样的功能）；Debug->Enable/Disable Breakpoint 是开启或暂停光标所在行的断点功能；Debug->Disable All Breakpoint 暂停所有断点；Debug->Kill All BreakPoint 清除所有的断点设置。这些功能也可以用工具条上的快捷按钮进行设置。

除了在某程序行设置断点这一基本方法以外，Keil 软件还提供了多种设置断点的方法，按 Debug->Breakpoints…即出现一个对话框，该对话框用于对断点进行详细的设置，如图 4 所示。

图 4 中 Expression 后的编辑框内用于输入表达式，该表达式用于确定程序停止运行的条件，这里表达式的定义功能非常强大，涉及到 Keil 内置的一套调试语法，这里不作详细说明，仅举若干实例，希望读者可以举一反三。

1） 在 Experssion 中键入 a==0xf7，再点击 Define 即定义了一个断点， 注意，a 后有两个等号，意即相等。该表达式的含义是：如果 a 的值到达 0xf7 则停止程序运行。除

使用相等符号之外，还可以使用>,>=,<,<=,!=（不等于）,&（两值按位与）,&&（两值相与）等运算符号。

2）在 Experssion 后中键入 Delay 再点击 Define，其含义是如果执行标号为 Delay 的行则中断。

3）在 Experssion 后中键入 Delay，按 Count 后的微调按钮，将值调到 3，其意义是当第三次执行到 Delay 时才停止程序运行。

4）在 Experssion 后键入 Delay，在 Command 后键入 printf("SubRoutine 'Delay' has been Called\n")主程序每次调用 Delay 程序时并不停止运行，但会在输出窗口 Command 页输出一行字符，即 SubRoutine 'Delay' has been Called。其中"\n"的用途是回车换行，使窗口输出的字符整齐。

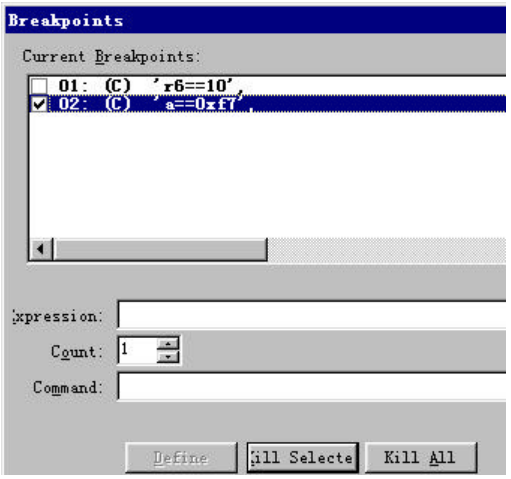

图 4 断点设置对话框

5）设置断点前先在输出窗口的 Command 页中键入 DEFINE int I，然后在断点设置时同 4），但是 Command 后键入 printf("SubRoutine 'Delay' has been Called %d times\n",++I)，则主程序每次调用 Delay 时将会在 Command 窗口输出该字符及被调用的次数，如 SubRoutine 'Delay' has been Called 10 times。

对于使用 C 源程序语言的调试，表达式中可以直接使用变量名，但必须要注意，设置时只能使用全局变量名和调试箭头所指模块中的局部变量名。

四、实例调试

为进行程序的调试，我们首先给源程序制造一个错误，将延时子程序的第三行"DJNZ R6,$"后的$改为 D1，然后重新编译，由于程序中并无语法错误，所以编译时不会有任何出错提示，但由于转移目的地出错，所以子程序将陷入无限循环中。

进入调试状态后，按 F10 以过程单步的形式执行程序，当执行到 LCALL DELAY 行时，程序不能继续往下执行，同时发现调试工具条上的 Halt 按钮变成了红色，说明程序在此不断地执行着，而我们预期这一行程序执行完后将停止，这个结果与预期不同，可以看出所调用的子程序出了差错。为查明出错原因，按 Halt 按钮使程序停止执行，然后按 RST 按钮使程序复位，再次按下 F10 单步执行，但在执行到 LCALL DELAY 行时，改按 F11 键跟踪到子程序内部（如果按下 F11 键没有反应，请在源程序窗口中用鼠标点一下），单步执行程序，可以发现在执行到"DJNZ R6,D1"行时，程序不断地从这一行转移到上一行，同时观察左侧的寄存器的值，会发现 R6 的值始终在 FFH 和 FEH 之间变化，不会减小，而我们的预期是 R6 的值不断减小，减到 0 后往下执行，因此这个结果与预期不符，通过这样的观察，不难发现问题是因为标号写错而产生的，发现问题即可以修改，为了验证即将进行的修改是否正确，可以先使用在线汇编功能测试一下。把光标定位于程序行"DJNZ R6,D1"，打开在线汇编的对话框，将程序改为"DJNZ R7,0EH"，即转回本条指令所在行继续执行，其中 0EH 是本条指令在程序存储器中的位置，这个值可以通过在线汇编窗口看到，如图 3 所示。然后关闭窗口，再进行调试，发现程序能够正确地执行了，这说明修改是正确的。注意，这时候的源程序并没有修改，此时应该退出调试程序，将源程序更改过来，并重新编译连接，以获得正确的目标代码。

Keil 程序调试窗口

上一讲中我们学习了几种常用的程序调试方法，这一讲中将介绍 Keil 提供各种窗口如输出窗口、观察窗口、存储器窗口、反汇编窗口、串行窗口等的用途，以及这些窗口的使用方法，并通过实例介绍这些窗口在调试中的使用。

一、程序调试时的常用窗口

Keil 软件在调试程序时提供了多个窗口，主要包括输出窗口（Output Windows）、观察窗口（Watch&Call Statck Windows）、存储器窗口（Memory Window）、反汇编窗口（Dissambly Window）串行窗口（Serial Window）等。进入调试模式后，可以通过菜单 View 下的相应命令打开或关闭这些窗口。

图 1 是输出窗口、观察窗口和存储器窗口，各窗口的大小可以使用鼠标调整。进入调试程序后，输出窗口自动切换到 Command 页。该页用于输入调试命令和输出调试信息。对于初学者，可以暂不学习调试命令的使用方法。

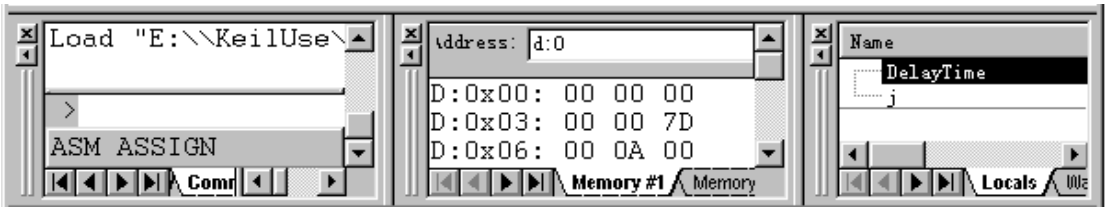

图 1 调试窗口（命令窗口、存储器窗口、观察窗口）

1、存储器窗口

存储器窗口中可以显示系统中各种内存中的值，通过在 Address 后的编缉框内输入“字母：数字”即可显示相应内存值，其中字母可以是 C、D、I、X，分别代表代码存储空间、直接寻址的片内存储空间、间接寻址的片内存储空间、扩展的外部 RAM 空间，数字代表想要查看的地址。例如输入 D：0 即可观察到地址 0 开始的片内 RAM 单元值、键入 C：0 即可显示从 0 开始的 ROM 单元中的值，即查看程序的二进制代码。该窗口的显示值可以以各种形式显示，如十进制、十六进制、字符型等，改变显示方式的方法是点鼠标右键，在弹出的快捷菜单中选择，该菜单用分隔条分成三部份，其中第一部份与第二部份的三个选项为同一级别，选中第一部份的任一选项，内容将以整数形式显示，而选中第二部份的 Ascii 项则将以字符型式显示，选中 Float 项将相邻四字节组成的浮点数形式显示、选中 Double 项则将相邻 8 字节组成双精度形式显示。第一部份又有多个选择项，其中 Decimal 项是一个开关，如果选中该项，则窗口中的值将以十进制的形式显示，否则按默认的十六进制方式显示。Unsigned 和 Signed 后分别有三个选项：Char、Int、Long，分别代表以单字节方式显示、将相邻双字节组成整型数方式

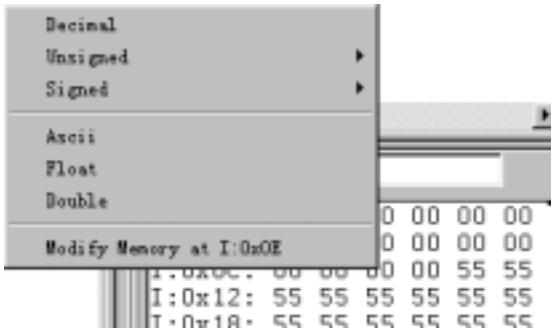

图 2 存储器数值各种方式显示选择

显示、将相邻四字节组成长整型方式显示，而 Unsigned 和 Signed 则分别代表无符号形式和有符号形式，究竟从哪一个单元开始的相邻单元则与你的设置有关，以整型为例，如果你输入的是 I:0，那么 00H 和 01H 单元的内容将会组成一个整型数，而如果你输入的是 I：1，01H 和 02H 单元的内容全组成一个整型数，以此类推。有关数据格式与 C 语言规定相同，请参考 C 语言书籍，默认以无符号单字节方式显示。第三部份的 Modify Memory at X:xx 用于更改鼠标处的内存单元值，选中该项即出现如图 3 所示的对话框，可以在对话框内输入要修改的内容。

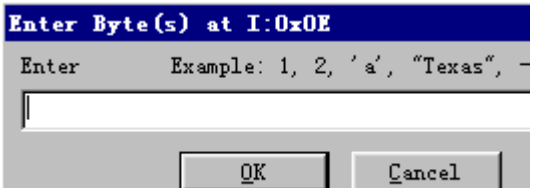

图 3 存储器的值的修改

2、工程窗口寄存器页

图 4 是工程窗口寄存器页的内容，寄存器页包括了当前的工作寄存器组和系统寄存器，系统寄存器组有一些是实际存在的寄存器如 A、B、DPTR、SP、PSW 等，有一些是实际中并不存在或虽然存在却不能对其操作的如 PC、Status 等。每当程序中执行到对某寄存器的操作时，该寄存器会以反色（蓝底白字）显示，用鼠标单击然后按下 F2 键，即可修改该值。

3、观察窗口

观察窗口是很重要的一个窗口，工程窗口中仅可以观察到工作寄存器和有限的寄存器如 A、B、DPTR 等，如果需要观察其它的寄存器的值或者在高级语言编程时需要直接观察变量，就要借助于观察窗口了。

其它窗口将在以下的实例中介绍。

一般情况下，我们仅在单步执行时才对变量的值的变化感兴趣，全速运行时，变量的值是不变的，只有在程序停下来之后，才会将这些值最新的变化反映出来，但是，在一些特殊场合下我们也可能需要在全速运行时观察变量的变化，此时可以点击 View->Periodic Window Updata（周期更新窗口），确认该项处于被选中状态，即可在全速运行时动态地观察有关值的变化。但是，选中该项，将会使程序模拟执行的速度变慢。

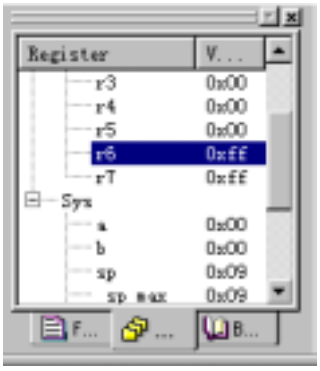

图 4 工程窗口寄存器页

二、各种窗口在程序调试中的用途

以下通过一个高级语言程序来说明这些窗口的使用。例 2：

```
#include "reg51.h"
sbit P1_0=P1^0;     //定义 P1.0
void mDelay(unsigned char DelayTime)
{     unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(j=0;j<125;j++)    {;}     }
}
void main()
{     unsigned int i;
      for(;;){    mDelay(10);    // 延 时   10 毫秒
                  i++;
                  if(i==10)
                  {       P1_0=!P1_0;
                          i=0;        }
      }   }
```

这个程序的工作过程是：不断调用延时程序，每次延时 10 毫秒，然后将变量 I 加 1，随后对变量 I 进行判断，如果 I 的值等于 10，那么将 P1.0 取反，并将 I 清 0，最终的执行效果

是 P1.0 每 0.1S 取反一次。

输入源程序并以 exam2.c 为文件名存盘，建立名为 exam2 的项目，将 exam2.c 加入项目，编译、连接后按 Ctrl+F5 进入调试，按 F10 单步执行。注意观察窗口，其中有一个标签页为 Locals，这一页会自动显示当前模块中的变量名及变量值。可以看到窗口中有名为 I 的变量，其值随着执行的次数而逐渐加大，如果在执行到 mDelay(10)行时按 F11 跟踪到 mDelay 函数内部，该窗口的变量自动变为 DelayTime 和 j。另外两个标签页 Watch #1 和 Watch #2 可以加入自定义的观察变量，点击“ type F2 to edit ”然后再按 F2 即可输入变量，试着在 Watch #1 中输入 I，观察它的变化。在程序较复杂，变量很多的场合，这两个自定义观察窗口可以筛选出我们自己感兴趣的变量加以观察。观察窗口中变量的值不仅可以观察，还可以修改，以该程序为例，I 须加 10 次才能到 10，为快速验证是否可以正确执行到 P1_0=!P1_0 行，点击 I 后面的值，再按 F2，该值即可修改，将 I 的值改到 9，再次按 F10 单步执行，即可以很快执行到 P1_0=!P1_0 程序行。该窗口显示的变量值可以以十进制或十六进制形式显示，方法是在显示窗口点右键，在快捷菜单中选择如图 5 所示。

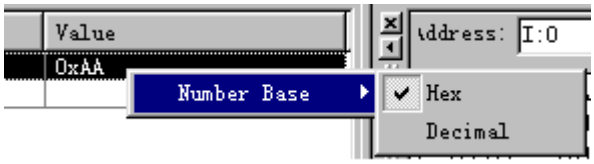

图 5 设定观察窗的显示方式

点击 View->Dissambly Window 可以打开反汇编窗口，该窗口可以显示反汇编后的代码、源程序和相应反汇编代码的混合代码，可以在该窗口进行在线汇编、利用该窗口跟踪已找行的代码、在该窗口按汇编代码的方式单步执行，这也是一个重要的窗口。打开反汇编窗口，点击鼠标右键，出现快捷菜单，如图 6 所示，其中 Mixed Mode 是以混合方式显示，Assembly Mode 是以反汇编码方式显示。

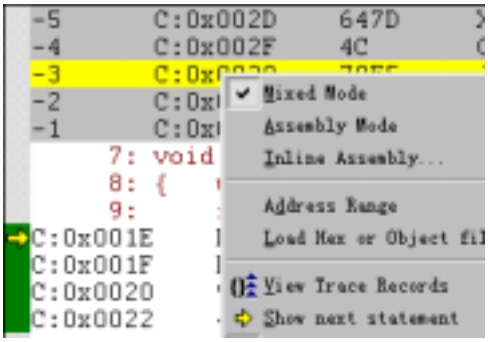

图 6 反汇编窗口

程序调试中常使用设置断点然后全速运行的方式，在断点处可以获得各变量值，但却无法知道程序到达断点以前究竟执行了哪些代码，而这往往是需要了解的，为此，Keil 提供了跟踪功能，在运行程序之前打开调试工具条上的允许跟踪代码开关，然后全速运行程序，当程序停止运行后，点击查看跟踪代码按钮，自动切换到反汇编窗口，如图 6 所示，其中前面标有“ - ”号的行就是中断以前执行的代码，可以按窗口边的上卷按钮向上翻查看代码执行记录。

利用工程窗口可以观察程序执行的时间，下面我们观察一下该例中延时程序的延时时间是否满足我们的要求，即是否确实延时 10 毫秒，展开工程窗口 Regs 页中的 Sys 目录树，其中的 Sec 项记录了从程序开始执行到当前程序流逝的秒数。点击 RST 按钮以复位程序，Sec 的值回零，按下 F10 键，程序窗口中的黄色箭头指向 mDelay(10)行，此时，记录下 Sec 值为 0.00038900，然后再按 F10 执行完该段程序，再次查看 Sec 的值为 0.01051200，两者相减大约是 0.01 秒，所以延时时间大致是正确的。读者可以试着将延时程序中的 unsigned int 改为 unsigned char 试试看时间是否仍正确。注意，使用这一功能的前提是在项目设置中正确设置晶振的数值。

Keil 提供了串行窗口，我们可以直接在串行窗口中键入字符，该字符虽不会被显示出来，但却能传递到仿真 CPU 中，如果仿真 CPU 通过串行口发送字符，那么这些字符会在串行窗口显示出来，用该窗口可以在没有硬件的情况下用键盘模拟串口通讯。下面通过一个例子说明 Keil 串行窗口的应用。该程序实现一个行编缉功能，每键入一个字母，会立即回显到窗

口中。编程的方法是通过检测 RI 是否等于 1 来判断串行口是否有字符输入，如果有字符输入，则将其送到 SBUF，这个字符就会在串行窗口中显示出来。其中 ser_init 是串行口初始化程序，要使用串行口，必须首先对串行口进行初始化。例 3：

```
    MOV     SP,#5FH    ;堆栈初始化
    CALL    SER_INIT ;串行口初始化
LOOP:
    JBC     RI,NEXT  ;如果串口接收到字
    符，转
    JMP     LOOP     ;否则等待接收字符
NEXT:
    MOV     A,SBUF    ;从 SBUF 中取字符
    MOV     SBUF,A    ;回送到发送 SBUF 中
SEND:
    JBC     TI,LOOP    ;发送完成，转 LOOP
    JMP     SEND       ;否则等待发送完
SER_INIT:              ;中断初始化
    MOV     SCON,#50H
    ORL     TMOD,#20H
    ORL     PCON,#80H
    MOV     TH1,#0FDH ;设定波特率
    SETB    TR1     ;定时器 1 开始运行
    SETB    REN     ;允许接收
    SETB    SM2
    RET
    END
```

输入源程序，并建立项目，正确编译、连接，进入调试后，全速运行，点击串行窗口 1 按钮，即在原源程序窗口位置出现一个空白窗口，击键，相应的字母就会出现在该窗口中。在窗口中击鼠标右键，出现一个弹出式菜单，选择“ Ascii Mode ”即以 Ascii 码的方式显示接收到的数据；选择“ Hex Mode ”以十六进制码方式显示接收到的数据；选择“ Clear Window ”可以清除窗口中显示的内容。

由于部份 CPU 具有双串口，故 Keil 提供了两个串行窗口，我们选用的 89C51 芯片只有一个串行口，所以 Serial 2 串行窗口不起作用。

小技巧：凡是鼠标单击然后按 F2 的地方都可以用鼠标连续单击两次（注意：不是双击）来替代。

Keil 的辅助工具和部份高级技巧

在前面的几讲中我们介绍了工程的建立方法，常用的调试方法，除此之外，Keil 还提供了一些辅助工具如外围接口、性能分析、变量来源分析、代码作用分析等，帮助我们了解程的性能、查找程序中的隐藏错误，快速查看程序变量名信息等，这一讲中将对这些功工具作一介绍，另外还将介绍 Keil 的部份高级调试技巧。

一、 辅助工具

这部份功能并不是直接用来进行程序调试的，但可以帮助我们进行程序的调试、程序性能的分析，同样是一些很有用的工具。

1、外围接口

为了能够比较直观地了解单片机中定时器、中断、并行端口、串行端口等常用外设的使用情况，Keil 提供了一些外围接口对话框，通过 Peripherals 菜单选择，该菜单的下拉菜单内容与你建立项目时所选的 CPU 有关，如果是选择的 89C51 这一类“ 标准 ”的 51 机，那么将会有 Interrupt（中断）、I/O Ports（并行 I/O 口）、Serial（串行口）、Timer（定时/计数器）这四个外围设备菜单。打开这些对话框，列出了外围设备的当前使用情况，各标志位的情况等，可以在这些对话框中直观地观察和更改各外围设备的运行情况。

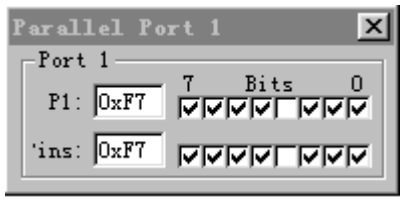

图 1 外围设备之并行端口

下面我们通过一个简单例子看一看并行端口的外围设备对话框的使用。例 4：

```
        MOV     A,#0FEH
LOOP:   MOV     P1,A
        RL      A
        CALL    DELAY   ;延时 100 毫秒
        JMP     LOOP
```

其中延时 100 毫秒的子程序请自行编写。

编译、连接进入调试后，点击 Peripherals->I/O-Ports->Port 1 打开，如图 1 所示，全速运行，可以看到代表各位的勾在不断变化（如果看不到变化，请点击 View->Periodic Window Updata），这样可以形象地看出程序执行的结果。

注：如果你看到的变化极快，甚至看不太清楚，那么说明你的计算机性能好，模拟执行的速度快，你可以试着将加长延时程序的时间以放慢速度。模拟运行速度与实际运行的速度无法相同是软件模拟的一个固有弱点。

点击 Peripherals->I/O-Ports->Timer0 即出现图 2 所示定时/计数器 0 的外围接口界面，可以直接选择 Mode 组中的下拉列表以确定定时/计数工作方式，0-3 四种工作方式，

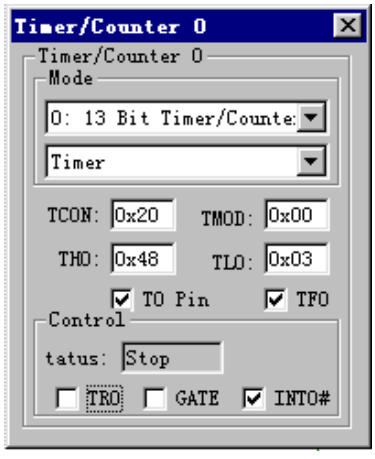

图 2 外围设备之定时器

设定定时初值等，点击选中 TR0，status 后的 stop 就变成了 run，如果全速运行程序，此时 th0,tl0 后的值也快速地开始变化（同样要求 Periodic Window Updata 处于选中状态），直观地演示了定时/计数器的工作情况（当然，由于你的程序未对此写任何代码，所以程序不会对此定时/计数器的工作进行处理）。

2、性能分析

Keil 提供了一个性能分析工具，利用该工具，我们可以了解程序中哪些部份的执行时间最长，调用次数最多，从而了解影响整个程序中执行速度的瓶颈。下面通过一个实例来看一看这个工具如何使用，例 5：

```
#include "reg51.h"
sbit P1_0=P1^0;     //定义 P1.0
void mDelay(unsigned char DelayTime)
{     unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(j=0;j<125;j++)    {;}     }
}
void mDelay1(unsigned char DelayTime)
{     unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(j=0;j<125;j++)    {;}     }
}
void main()
{     unsigned int i;
      for(;;){    mDelay(10);   // 延 时  10 毫秒
            i++;
            if(i==10)
            {       P1_0=!P1_0;
                  i=0;
                  mDelay1(10);}
 } }
```

编译连接。进入调试状态后使用菜单 View->Performance Analyzer Window，打开性能分析对话框，进入该对话框后，只有一项 unspecified，点鼠标右键，在快捷菜单中选择 Setup PA 即打开性能分析设置对话框，对于 C 语言程序，该对话框右侧的“ Function Symbol ”下的列表框给出函数符号，双击某一符号，该符号即出现在 Define Performance Analyzer 下的编缉框中，每输入一个符号名字，点击 Define 按钮，即将该函数加入其上的分析列表框。对于汇编语言源程序，Function Symbol 下的列表框中不会出现子程序名，可以直接在编缉框中输入子程序名，点击 Close 关闭窗口，回到性能分析窗口，此时窗口共有 4 个选项。全速执行程序，可以看到 mDelay 和 mDelay1 后出现一个蓝色指示条，配合上面的标尺可以直观地看出每个函数占整个执行时间的比例，点击相应的函数名，可以在该窗口的状态栏看到更详细的数据，其中各项的含义如下：

Min：该段程序执行所需的最短时间；Max：该段程序执行所需的最长时间；Avg：该段程序执行所花平均时间；Total：该段程序到目前为目总共执行的时间；%：占整个执行时间的百分比；count：被调用的次数。

本程序中，函数 mDelay 和 mDelay1 每次被调用都花费同样的时间，看不出 Min、Max、和 Avg 的意义，实际上，由于条件的变化，某些函数执行的时间不一定是一个固定的值，借助于这些信息，可以对程序有更详细的了解。下面将 mDelay1 函数略作修改作一演示。

```
void mDelay1(unsigned char DelayTime)
{     static unsigned char k;
      unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(;j<k;j++)
            {;}
      }     k++; }
```

程序中定义了一个静态变量 K，每次调用该变量加 1，而 j 的循环条件与 k 的大小有关，

这使每次执行该程序所花的时间不一样。编译、执行该程序，再次观察性能分析窗口，可以看出 Min、Max、Avg 的意义。

3、变量来源浏览

该窗口用于观察程序中变量名的有关信息，如该变量名在那一个函数中被定义、在哪里被调用，共出现多少次等。在 Source Browse 窗口中提供了完善的管理方法，如过滤器可以分门别类地列出各种类别的变量名，可以对这些变量按 Class（组）、Type（类型）、Space（所在空间）、Use（调用次数）排序，点击变量名，可以在窗口的右侧看到该变量名的更详细的信息。

4、代码作用范围分析

在你写的程序中，有些代码可能永远不会被执行到（这是无效的代码），也有一些代码必须在满足一定条件后才能被执行到，借助于代码范围分析工具，可以快速地了解代码的执行情况。

进入调试后，全速运行，然后按停止按钮，停下来后，可以看到在源程序的左列有三种颜色，灰、淡灰和绿，其中淡灰所指的行并不是可执行代码，如变量或函数定义、注释行等等，而灰色行是可执行但从未执行过的代码，而绿色则是已执行过的程序行。使用调试工具条上的 Code Coverage Window 可打开代码作用范围分析的对话框，里面有各个模块代码执行情况的更详细的分析。如果你发现全速运行后有一些未被执行到的代码，那么就要仔细分析，这些代码究竟是无效的代码还是因为条件没有满足而没有被执行到。

二、部份高级调试技巧

Keil 内置了一套调试语言，很多高级调试技巧与此有关，但是全面学习这套语言并不现实，这不是这么几期连载可以胜任的，这里仅介绍部份较为实用的功能，如要获得更详细的信息，请参考 Keil 自带的帮助文件 GS51.PDF。

1、串行窗口与实际硬件相连

Keil 的串行窗口除可以模拟串行口的输入和输出功能外还可以与 PC 机上实际的串口相连，接受串口输入的内容，并将输出送到串口。这需要在 Keil 中进行设置。方法是首先在输出窗口的 Command 页用 MODE 命令设置串口的工作方式，然后用 ASSIGN 命令将串行窗口与实际的串口相关联，下面我们通过一个实例来说明如何操作。例 6：

```
ORG     0000H
    JMP START
    ORG     3+4*8    ;串行中断入口
    JMP SER_INT
START:
    MOV     SP,#5FH   ;堆栈初始化
    CALL    SER_INIT ;串行口初始化 A
    SETB    EA  ;
    SETB    ES  ;
    JMP $    ;主程序到此结束
SER_INT:
    JBC     RI,NEXT  ;如果串口接收到字符，转
    JMP     SEND     ;否则转发送处理
NEXT:
    MOV    A,SBUF   ;从 SBUF 中取字符
```

```
    MOV   SBUF,A   ;回送到发送 SBUF 中
    JMP   OVER
SEND:
    clr   ti
OVER:
    reti

SER_INIT:              ;中断初始化
     MOV       SCON,#50H
     ORL       TMOD,#20H
     ORL       PCON,#80H
     MOV        TH1,#0FDH ;设定波特率
     SETB      TR1    ;定时器 1 开始运行
     SETB      REN    ;允许接收
     SETB      SM2
     RET
                        END
```

这个程序使用了中断方式编写串行口输入/输出程序，它的功能是将接串行口收到的字符回送，即再通过串行口发送出去。

正确输入源文件、建立工程、编译连接没有错后，可进行调试，使用 Keil 自带的串行窗口测试功能是否正确，如果正确，可以进行下一步的连机试验。

为简单实用，我们不借助于其它的硬件，而是让 PC 机上的两个串口互换数据，即 COM1 发送 COM2 接收，而 COM2 发送则由 COM1 接收，为此，需要做一根连接线将这两个串口连起来，做法很简单，找两个可以插入 PC 机串口的 DIN9 插座（母），然后用一根 3 芯线将它们连起来，连线的方法是：

2——3

3——2

5——5

接好线把两个插头分别插入 PC 机上的串口 1 与串口 2。找一个 PC 机上的串口终端调试软件，如串口精灵之类，运行该软件，设置好串口参数，其中串口选择 2，串口参数设置为：

19200，n，8，1 其含义是波特率为 19200，无奇偶校验，8 位数据，1 位停止位。

在 Keil 调试窗口的 command 页中输入：

```
>mode com1 19200,0,8,1
>assign com1 <sin>sout
```

注意两行最前面的“ > ”是提示符，不要输入，第二行中的“ < ”和“ > ”即“ 小于 ”和“ 大于 ”符号，中间的是字母“ s ”和“ input ”的前两个字母，最后是字母“ s ”和“ output ”的前三个字母。

第一行命令定义串口 1 的波特率为 19200，无奇偶校验，8 位数据，1 位停止位。第二行是将串口 1（com1）分配给串行窗口。

全速运行程序，然后切换串口精灵，开始发送，会看到发送后的数据会立即回显到窗口中，说明已接收到了发送过来的数据。切换到 uVison，查看串行窗口 1，会看到这里的确接收到了串口精灵送来的内容。

2、从端口送入信号

程序调试中如果需要有信号输入，比如数据采集类程序，需要从外界获得数据，由于 Keil 的调试完全是一个软件调试工具，没有硬件与之相连，所以不可能直接获得数据，为此必须采用一些替代的方法，例如，某电路用 P1 口作为数据采集口，那么可以使用的一种方法是利用外围接口，打开 PORT 1，用鼠标在点击相应端口位，使其变为高电平或低电平，就能输入数据。显然，这种方法对于要输获得数据而不是作位处理来说太麻烦了，另一种方法是直接在 command 页输入 port1=数值，以下是一个小小的验证程序。例 7：

```
LOOP:   MOV     A,P1
```

```
        JZ      NEXT
        MOV     R0,#55H
        JMP     LOOP
NEXT:   MOV     R0,#0AAH
        JMP     LOOP
        END
```

该程序从 P1 口获得数据，如果 P1 口的值是 0，那么就让 R0 的值为 0AAH，否则让 R0 的值为 55H。输入源程序并建立工程，进入调试后，在观察窗口加入 R0，然后全速运行程序，注意确保 View->Periodic Window Updata 处于选中状态，然后在 Command 后输入 PORT1=0 回车后可以发现观察窗口中的 R0 的值变成了 0AAH，然后再输入 PORT1=1 或其它非零值，则 R0 的值会变为 55H。

同样的道理，可以用 port0、port2、port3 分别向端口 0、2、3 输入信号。

3、直接更改内存值

在程序运行中，另一种输入数据的方法是直接更改相应的内存单元的值，例如，某数据采集程序，使用 30H 和 31H 作为存储单元，采入的数据由这两个单元保存，那么我们更改了 30H 和 31H 单元的值就相当于这个数据采集程序采集到了数据，这可以在内存窗口中直接修改（参考上一讲），也可以通过命令进行修改，命令的形式是： _WBYTE (地址,数据)，其中地址是指待写入内存单元的地址，而数据则是待写入该地址的数据。例如 _WBYTE(0x30,11)会将值 11 写入内存地址十六进制 30H 单元中。