单片机的 C 语言轻松入门

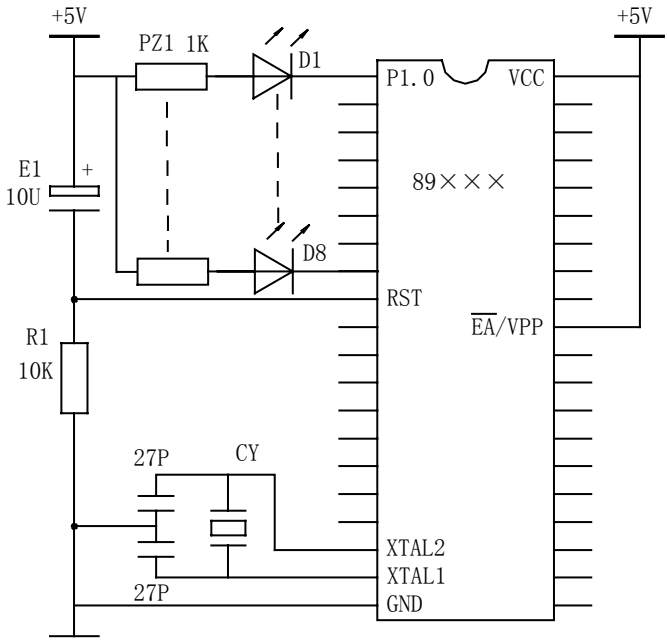
随着单片机开发技术的不断发展，目前已有越来越多的人从普遍使用汇编语言到逐渐使用高级语言开发，其中主要是以 C 语言为主，市场上几种常见的单片机均有其 C 语言开发环境。这里以最为流行的 80C51 单片机为例来学习单片机的 C 语言编程技术。

本书共分六章，每章一个专题，以一些待完成的任务为中心，围绕该任务介绍 C 语言的一些知识，每一个任务都是可以独立完成的，每完成一个任务，都能掌握一定的知识，等到所有的任务都完成后，即可以完成 C 语言的入门工作。

第 1 章　C 语言概述及其开发环境的建立

学习一种编程语言，最重要的是建立一个练习环境，边学边练才能学好。Keil 软件是目前最流行开发 80C51 系列单片机的软件，Keil 提供了包括 C 编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境（μVision）将这些部份组合在一起。

在学会使用汇编语言后，学习 C 语言编程是一件比较容易的事，我们将通过一系列的实例介绍 C 语言编程的方法。图 1-1 所示电路图使用 89S52 单片机作为主芯片，这种单片机性属于 80C51 系列，其内部有 8K 的 FLASH　ROM,可以反复擦写，并有 ISP 功能，支持在线下载，非常适于做实验。89S52 的 P1 引脚上接 8 个发光二极管，P3.2~P3.4 引脚上接 4 个按钮开关，我们的任务是让接在 P1 引脚上的发光二极管按要求发光。



1．1　简单的 C 程序介绍

例 1-1：　让接在 P1.0 引脚上的 LED 发光。

```
/*************************************************
    平凡单片机工作室
    http://www.mcustudio.com
    Copyright 2003 pingfan's mcustudio
    All rights Reserved
作者：周坚
dddl.c
单灯点亮程序
*************************************************/
```

```
#include “reg51.h”
sbit P1_0=P1^0;
void main()
{     P1_1=0;
}
```

这个程序的作用是让接在 P1.0 引脚上的 LED 点亮。下面来分析一下这个 C 语言程序包含了哪些信息。

1）“文件包含”处理。

程序的第一行是一个“文件包含”处理。

所谓“文件包含”是指一个文件将另外一个文件的内容全部包含进来，所以这里的程序虽然只有 4 行，但 C 编译器在处理的时候却要处理几十或几百行。这里程序中包含 REG51.h 文件的目的是为了要使用 P1 这个符号，即通知 C 编译器，程序中所写的 P1 是指 80C51 单片机的 P1 端口而不是其它变量。这是如何做到的呢？

打开 reg51.h 可以看到这样的一些内容：

```
/*--------------------------------------------------------------------------
REG51.H

Header file for generic 80C51 and 80C31 microcontroller.
Copyright (c) 1988-2001 Keil Elektronik GmbH and Keil Software, Inc.
All rights reserved.
--------------------------------------------------------------------------*/

/*    BYTE Register    */
sfr P0      = 0x80;
sfr P1      = 0x90;
sfr P2      = 0xA0;
sfr P3      = 0xB0;
sfr PSW     = 0xD0;
sfr ACC     = 0xE0;
sfr B       = 0xF0;
sfr SP      = 0x81;
sfr DPL     = 0x82;
sfr DPH     = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0     = 0x8A;
sfr TL1     = 0x8B;
sfr TH0     = 0x8C;
sfr TH1     = 0x8D;
sfr IE      = 0xA8;
sfr IP      = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
```

```
/*    BIT Register    */
/*    PSW      */
sbit CY      = 0xD7;
sbit AC      = 0xD6;
sbit F0      = 0xD5;
sbit RS1     = 0xD4;
sbit RS0     = 0xD3;
sbit OV      = 0xD2;
sbit P       = 0xD0;

/*    TCON    */
sbit TF1     = 0x8F;
sbit TR1     = 0x8E;
sbit TF0     = 0x8D;
sbit TR0     = 0x8C;
sbit IE1     = 0x8B;
sbit IT1     = 0x8A;
sbit IE0     = 0x89;
sbit IT0     = 0x88;

/*    IE      */
sbit EA      = 0xAF;
sbit ES      = 0xAC;
sbit ET1     = 0xAB;
sbit EX1     = 0xAA;
sbit ET0     = 0xA9;
sbit EX0     = 0xA8;

/*    IP      */
sbit PS      = 0xBC;
sbit PT1     = 0xBB;
sbit PX1     = 0xBA;
sbit PT0     = 0xB9;
sbit PX0     = 0xB8;

/*    P3    */
sbit RD      = 0xB7;
sbit WR      = 0xB6;
sbit T1      = 0xB5;
sbit T0      = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD     = 0xB1;
```

```
sbit RXD    = 0xB0;

/*    SCON    */
sbit SM0    = 0x9F;
sbit SM1    = 0x9E;
sbit SM2    = 0x9D;
sbit REN    = 0x9C;
sbit TB8    = 0x9B;
sbit RB8    = 0x9A;
sbit TI     = 0x99;
sbit RI     = 0x98;
```

熟悉 80C51 内部结构的读者不难看出，这里都是一些符号的定义，即规定符号名与地址的对应关系。注意其中有

```
sfr P1      = 0x90;
```

这样的一行（上文中用黑体表示），即定义 P1 与地址 0x90 对应，P1 口的地址就是 0x90（0x90 是 C 语言中十六进制数的写法，相当于汇编语言中写 90H）。

从这里还可以看到一个频繁出现的词：sfr

sfr 并标准 C 语言的关键字，而是 Keil 为能直接访问 80C51 中的 SFR 而提供了一个新的关键词，其用法是：

sfrt 变量名=地址值。

2）符号 P1_0 来表示 P1.0 引脚。

在 C 语言里，如果直接写 P1.0，C 编译器并不能识别，而且 P1.0 也不是一个合法的 C 语言变量名，所以得给它另起一个名字，这里起的名为 P1_0，可是 P1_0 是不是就是 P1.0 呢？你这么认为，C 编译器可不这么认为，所以必须给它们建立联系，这里使用了 Keil C 的关键字 sbit 来定义，sbit 的用法有三种：

第一种方法：sbit 位变量名＝地址值

第二种方法：sbit 位变量名＝SFR 名称^变量位地址值

第三种方法：sbit 位变量名＝SFR 地址值^变量位地址值

如定义 PSW 中的 OV 可以用以下三种方法：

```
sbit OV=0xd2        （1）说明：0xd2 是 OV 的位地址值
sbit OV=PSW^2       （2）说明：其中 PSW 必须先用 sfr 定义好
sbit OV=0xD0^2      （3）说明：0xD0 就是 PSW 的地址值
```

因此这里用 sfr P1_0=P1^0;就是定义用符号 P1_0 来表示 P1.0 引脚，如果你愿意也可以起 P10 一类的名字，只要下面程序中也随之更改就行了。

3）main 称为“主函数”。

每一个 C 语言程序有且只有一个主函数，函数后面一定有一对大括号“{}”，在大括号里面书写其它程序。

从上面的分析我们了解了部分 C 语言的特性，下面再看一个稍复杂一点的例子。

例 1-2 让接在 P1.0 引脚上的 LED 闪烁发光

```
/************************************************
    平凡单片机工作室
    http://www.mcustudio.com
    Copyright 2003 pingfan's mcustudio
    All rights Reserved
```

```
作者：周坚
ddss.c
单灯闪烁程序
**************************************************/
#include "reg51.h"
#define uchar unsigned char
#define uint    unsigned int
sbit  P10=P1^0;

/*延时程序
   由 Delay 参数确定延迟时间
*/
void mDelay(unsigned int Delay)
{     unsigned int i;
      for(;Delay>0;Delay--)
      {     for(i=0;i<124;i++)
            {;}
      }
}

void main()
{     for(;;)
      {     P10=!P10;             //取反 P1.0 引脚
            mDelay(1000);
      }
}
```

程序分析：主程序 main 中的第一行暂且不看，第二行是“P1_0=!P1_0;”，在 P1_0 前有一个符号“!”，符号“!”是 C 语言的一个运算符，就像数学中的“+”、“-”一样，是一种运算任号，意义是“取反”，即将该符号后面的那个变量的值取反。

注意：取反运算只是对变量的值而言的，并不会自动改变变量本身。可以认为 C 编译器在处理“！P1_0”时，将 P1_0 的值给了一个临时变量，然后对这个临时变量取反，而不是直接对 P1_0 取反，因此取反完毕后还要使用赋值符号（“＝”）将取反后的值再赋给 P1_0，这样，如果原来 P1.0 是低电平（LED 亮），那么取反后，P1.0 就是高电平（LED 灭），反之，如果 P1.0 是高电平，取反后，P1.0 就是低电平，这条指令被反复地执行，接在 P1.0 上灯就会不断“亮”、“灭”。

该条指令会被反复执行的关键就在于 main 中的第一行程序：for(;;)，这里不对此作详细的介绍，读者暂时只要知道，这行程序连同其后的一对大括号“{}”构成了一个无限循环语句，该大括号内的语句会被反复执行。

第三行程序是：“mDelay(1000);”，这行程序的用途是延时 1s 时间，由于单片机执行指令的速度很快，如果不进行延时，灯亮之后马上就灭，灭了之后马上就亮，速度太快，人眼根本无法分辨。

这里 mDelay(1000)并不是由 Keil C 提供的库函数，即你不能在任何情况下写这样一行程序以实现延时。如果在编写其它程序时写上这么一行，会发现编译通不过。那么这里为什么又是正确的呢？注意观察，可以发现这个程序中有 void mDelay(…)这样一行，可见，

mDelay 这个词是我们自己起的名字，并且为此编写了一些程序行，如果你的程序中没有这么一段程序行，那就不能使用 mDelay（1000）了。有人脑子快，可能马上想到，我可不可以把这段程序也复制到我其它程序中，然后就可以用 mDelay(1000)了呢？回答是，那当然就可以了。还有一点需要说明，mDelay 这个名称是由编程者自己命名的，可自行更改，但一旦更改了名称，main()函数中的名字也要作相应的更改。

mDelay 后面有一个小括号，小括号里有数据（1000），这个 1000 被称之“参数”，用它可以在一定范围内调整延时时间的长短，这里用 1000 来要求延时时间为 1000 毫秒，要做到这一点，必须由我们自己编写的 mDelay 那段程序决定的，详细情况在后面循环程序中再作分析，这里就不介绍了。

1．2　Keil 工程的建立

要使用 Keil 软件，首先要正确安装 Keil 软件，该软件的 Eval 版本可以直接去 http://www.keil.com 下载，安装时选择 Eval Vision，其它步骤与一般 Windows 程序安装类似，这里就不再赘述了。安装完成后，将 Ledkey.dll 文件复制到 Keil 安装目录下的 C51\BIN 文件夹下，这是作者提供的键盘与 LED 实验仿真板，可与 Keil 软件配合，在计算机上模拟 LED 和按键的功能。

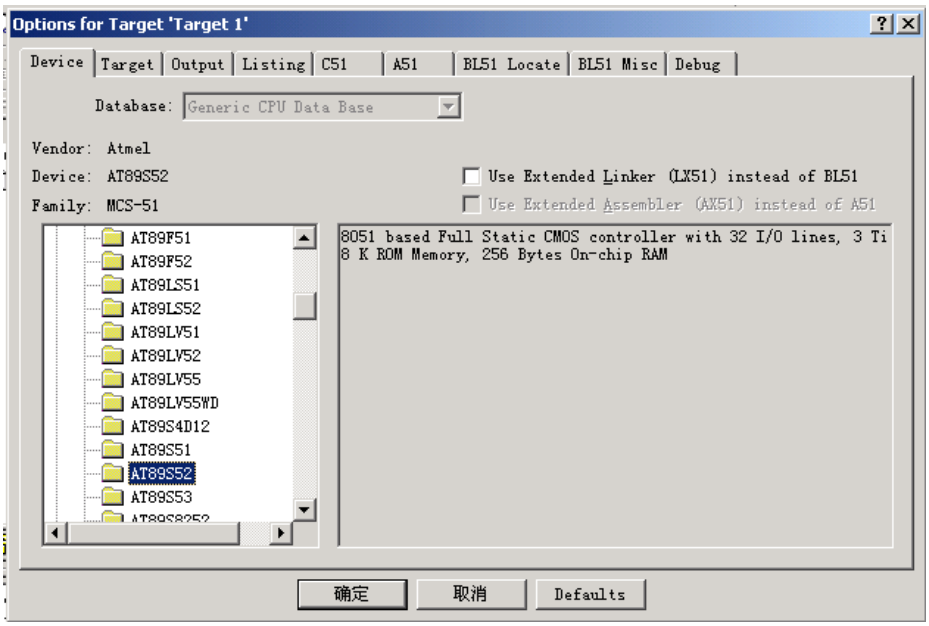
启动μVison，点击“File→New…”在工程管理器的右侧打开一个新的文件输入窗口，在这个窗口里输入例 1-2 中的源程序，注意大小写及每行后的分号，不要错输及漏输。

输入完毕之后，选择“File→Save”，给这个文件取名保存，取名字的时候必须要加上扩展名，一般 C 语言程序均以“.C”为扩展名，这里将其命名为 exam2.c，保存完毕后可以将该文件关闭。

Keil 不能直接对单个的 C 语言源程序进行处理，还必须选择单片机型号；确定编译、汇编、连接的参数；指定调试的方式；而且一些项目中往往有多个文件，为管理和使用方便，Keil 使用工程（Project）这一概念，将这些参数设置和所需的所有文件都加在一个工程中，只能对工程而不能对单一的源程序进行编译和连接等操作。

点击“Project->New Project…”菜单，出现对话框，要求给将要建立的工程起一个名字，这里起名为 exam2，不需要输入扩展名。点击“保存”按钮，出现第二个对话框，如图 1-2 所示，这个对话框要求选择目标 CPU（即你所用芯片的型号），Keil 支持的 CPU 很多，这里选择 Atmel 公司的 89S52 芯片。点击 ATMEL 前面的“+”号，展开该层，点击其中的 89S52，然后再点击“确定”按钮，回到主窗口，此时，在工程窗口的文件页中，出现了“Target 1”，前面有“+”号，点击“+”号展开，可以看到下一层的“Source Group1”，这时的工程还是一个空的工程，里面什么文件也没有，需要手动把刚才编写好的源程序加入，点击“Source Group1”使其反白显示，然后，点击鼠标右键，出现一个下拉菜单，如图 1-3 所示，选中其中的“Add file to Group”Source Group1”，出现一个对话框，要求寻找源文件。

双击 exam2.c 文件，将文件加入项目，注意，在文件加入项目后，该对话框并不消失，等待继续加入其它文件，但初学时常会误认为操作没有成功而再次双击同一文件，这时会出现如图 1-4 所示的对话框，提示你所选文件已在列表中，此时应点击“确定”，返回前一对话框，然后点击“Close”即可返回主接口，返回后，点击“Source Group 1”前的加号，exam3.c 文件已在其中。双击文件名，即打开该源程序。

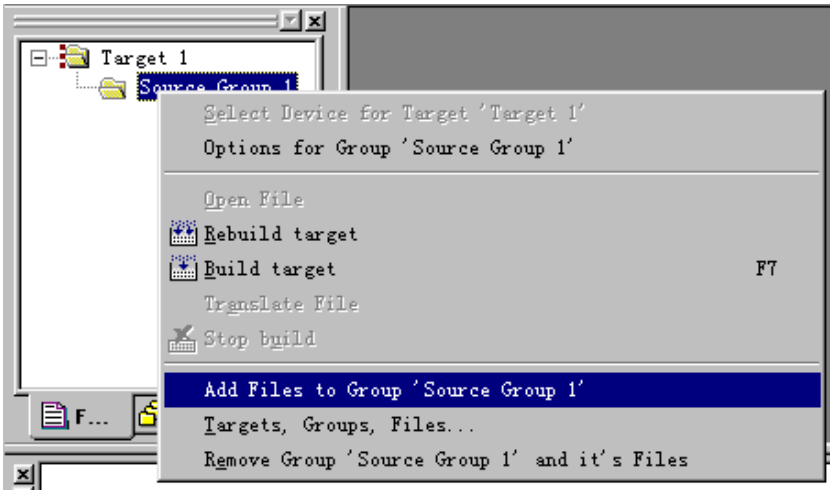


1．3　工程的详细设置

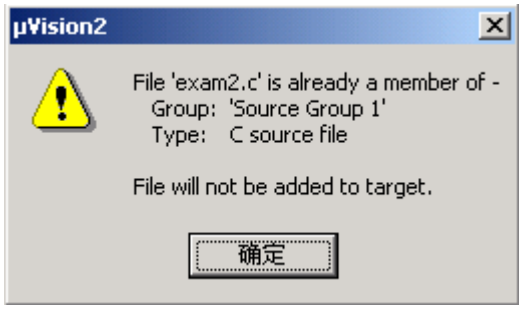
工程建立好以后，还要对工程进行进一步的设置，以满足要求。

首先点击左边 Project 窗口的 Target 1，然后使用菜单“Project->Option for target ‘target1’”即出现对工程设置的对话框，这个对话框共有 8 个页面，大部份设置项取默认值就行了。

Target 页



如图 1-5 所示，Xtal 后面的数值是晶振频率值，默认值是所选目标 CPU 的最高可用频率值，该值与最终产生的目标代码无关，仅用于软件模拟调试时显示程序执行时间。正确设置该数值可使显示时间与实际所用时间一致，一般将其设置成与你的硬件所用晶振频率相同，如果没必要了解程序执行的时间，也可以不设。



Memory Model 用于设置 RAM 使用情况，有三个选择项：

Small： 所有变量都在单片机的内部 RAM 中；

Compact：可以使用一页（256 字节）外部扩展 RAM；

Larget： 可以使用全部外部的扩展 RAM。

Code Model 用于设置 ROM 空间的使用，同样也有三个选择项：

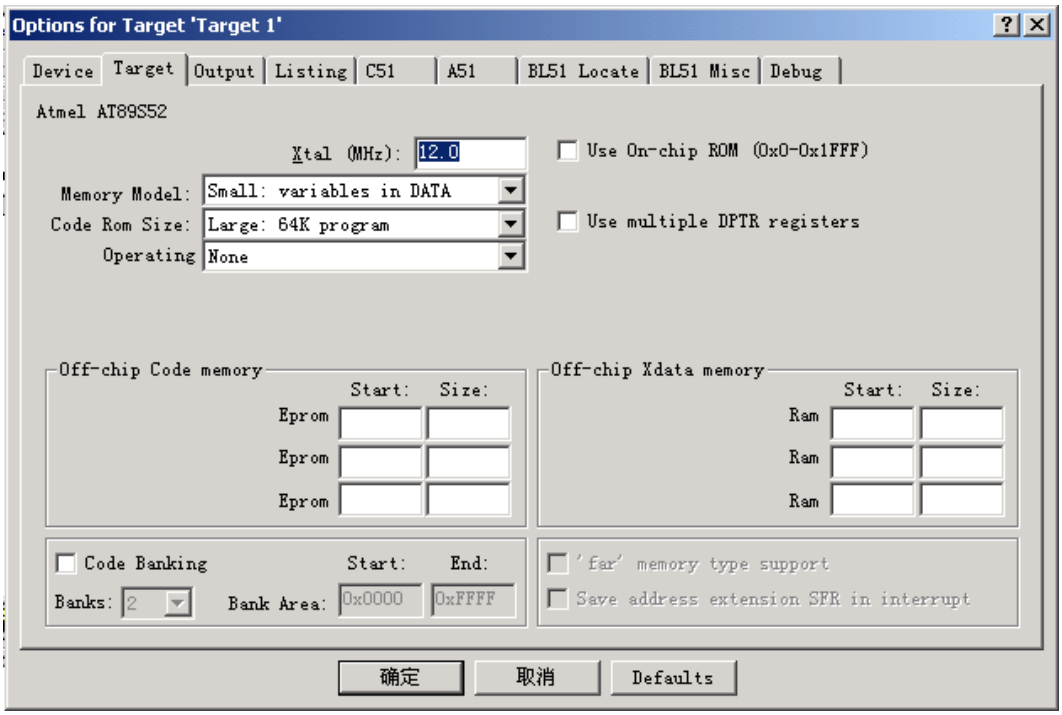
Small：只用低于 2K 的程序空间；

Compact：单个函数的代码量不能超过 2K，整个程序可以使用 64K 程序空间；

Larget：可用全部 64K 空间；

这些选择项必须根据所用硬件来决定，由于本例是单片应用，所以均不重新选择，按默认值设置。

Operating：选择是否使用操作系统，可以选择 Keil 提供了两种操作系统：Rtx tiny 和



Rtx full，也可以不用操作系统（None），这里使用默认项 None，即不用操作系统。

OutPut 页

如图 1-6 所示，这里面也有多个选择项，其中 Creat Hex file 用于生成可执行代码文件，该文件可以用编程器写入单片机芯片，其格式为 intelHEX 格式，文件的扩展名为.HEX，默认情况下该项未被选中，如果要写片做硬件实验，就必须选中该项。

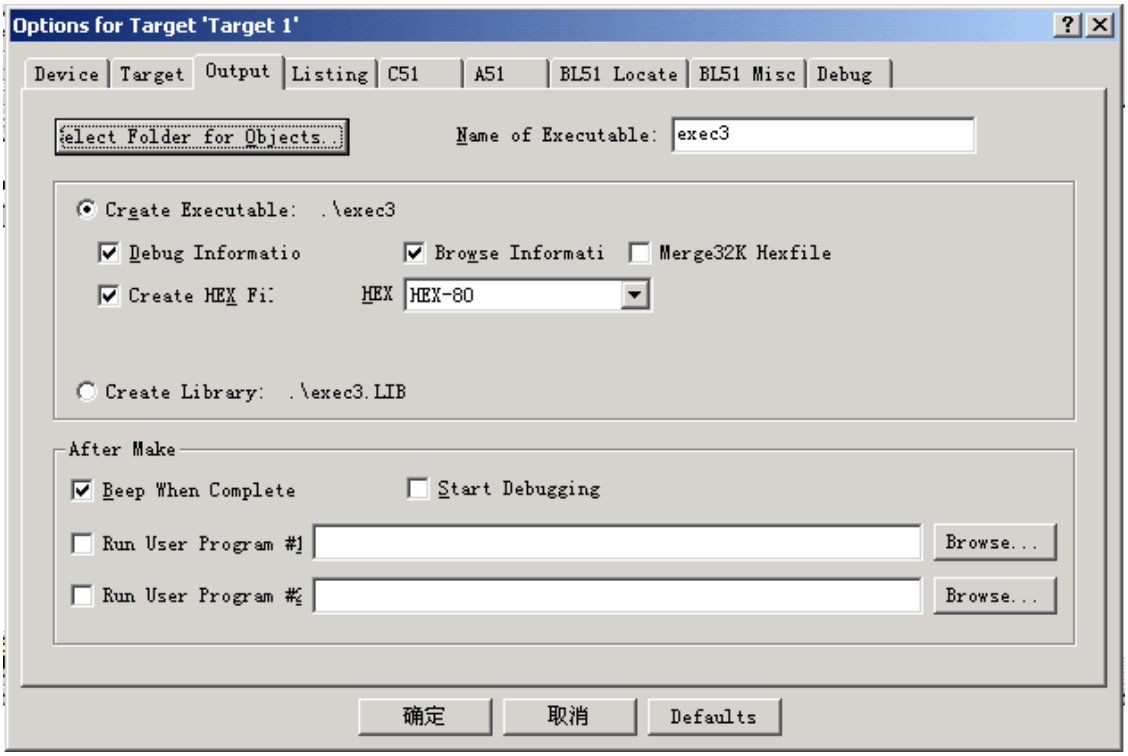
工程设置对话框中的其它各页面与 C51 编译选项、A51 的汇编选项、BL51 连接器的连接选项等用法有关，这里均取默认值，不作任何修改。以下仅对一些有关页面中常用的选项作一个简单介绍。

Listing 页

该页用于调整生成的列表文件选项。在汇编或编译完成后将产生（*.lst）的列表文件，在连接完成后也将产生（*.m51）的列表文件，该页用于对列表文件的内容和形式进行细致的调节，其中比较常用的选项是“C Compile Listing”下的“Assamble Code”项，选中该项可以在列表文件中生成 C 语言源程序所对应的汇编代码，建议会使用汇编语言的 C 初学者选中该项，在编译完成后多观察相应的 List 文件，查看 C 源代码与对应汇编代码，对于提高 C 语言编程能力大有好处。

C51 页

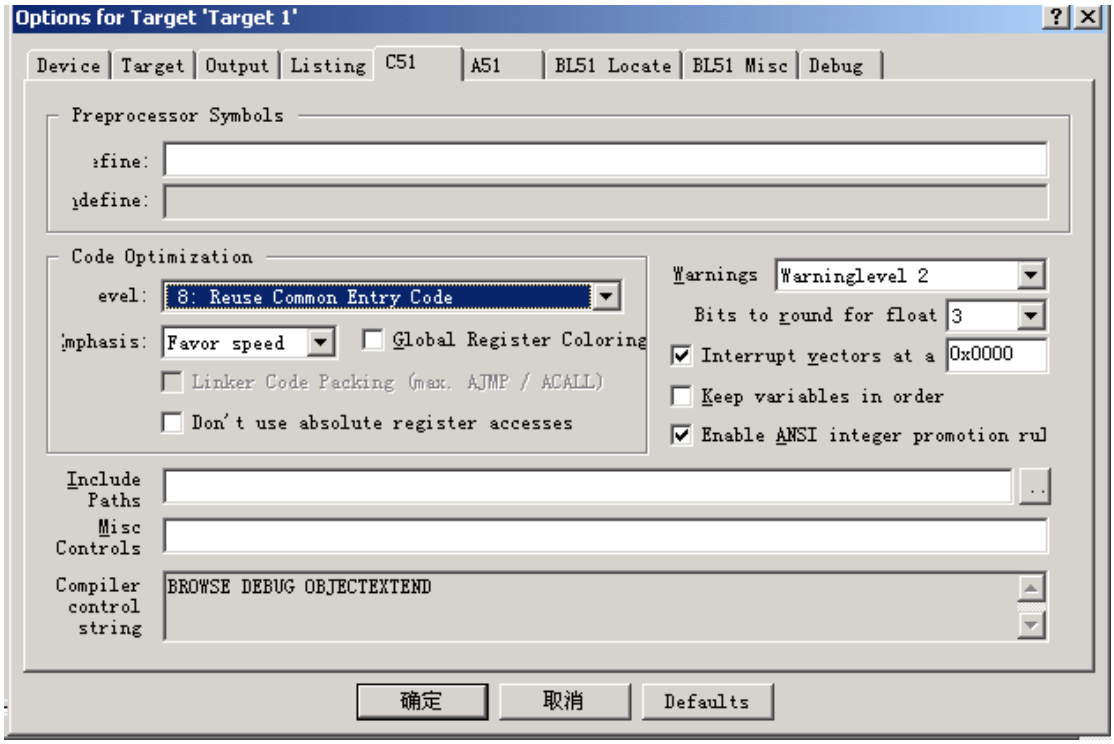
该页用于对 Keil 的 C51 编译器的编译过程进行控制，其中比较常用的是“Code Optimization”组，如图 1.7 所示，该组中 Level 是优化等级，C51 在对源程序进行编译时，可以对代码多至 9 级优化，默认使用第 8 级，一般不必修改，如果在编译中出现一些问题，可以降低优化级别试一试。Emphasis 是选择编译优先方式，第一项是代码量优化（最终生



成的代码量小）；第二项是速度优先（最终生成的代码速度快）；第三项是缺省。默认采用速度优先，可根据需要更改。

Debug 页

该页用于设置调试器，Keil 提供了仿真器和一些硬件调试方法，如果没有相应的硬件调试器，应选择 Use Simulator，其余设置一般不必更改，有关该页的详细情况将在程序调试部



分再详细介绍。

至此，设置完成，下面介绍如何编译、连接程序以获得目标代码，以及如何进行程序的调试工作。

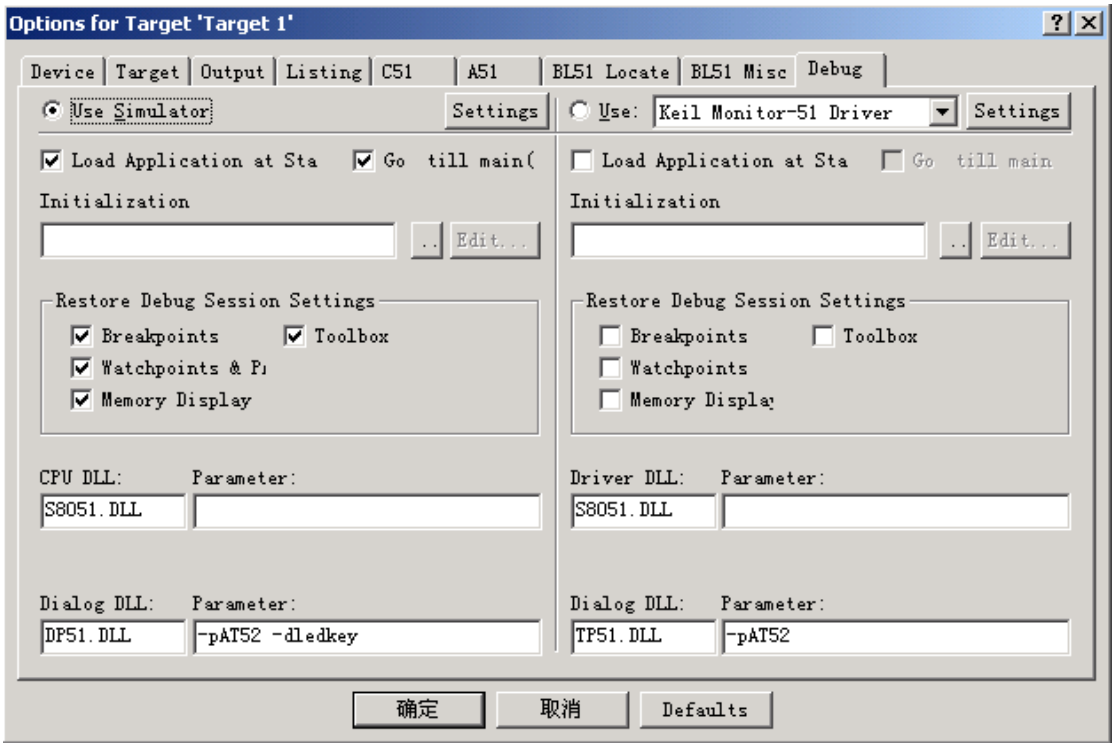
1．4　编译、连接

下面我们通过一个例子来介绍 C 程序编译、连接的过程。这个例子使 P1 口所接 LED 以流水灯状态显示。

将下面的源程序输入，命名为 exam3.c，并建立名为 exam3 的工程文件，将 exam3.c 文件加入该工程中，设置工程，在 Target 页将 Xtal 后的值由 24.0 改为 12.0，以便后面调试时观察延时时间是否正确，本项目中还要用到我们所提供的实验仿真板，为此需在 Debug 页对 Dialog DLL 对话框作一个设置，在进行项目设置时点击 Debug，打开 Debug 页，可以看到 Dialog DLL 对话框后的 Parmeter:输入框中已有默认值-pAT52，在其后键入空格后再输入-dledkey，如图 1-8 所示。

例 1-3　使 P1 口所接 LED 以流水灯状态显示

```
/*************************************************
;      平凡单片机工作室
;      http://www.mcustudio.com
;      Copyright 2003 pingfan's McuStudio
;      All rights Reserved
```

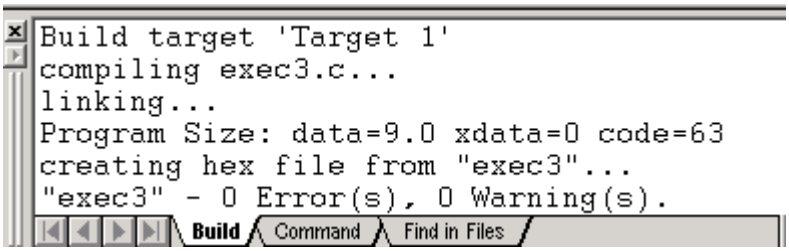



```
;作者：周坚
;lsd.c
;流水灯程序
**************************************************/
#include "reg51.h"
#include "intrins.h"
#define uchar unsigned char
#define uint    unsigned int

/*延时程序
   由 Delay 参数确定延迟时间
*/
void mDelay(unsigned int Delay)
{     unsigned int i;
      for(;Delay>0;Delay--)
      {     for(i=0;i<124;i++)
            {;}
      }
}

void main()
{     unsigned char OutData=0xfe;
      for(;;)
      {
```

```
            P1=OutData;
            OutData=_crol_(OutData,1); //循环左移
            mDelay(1000);      /*延时 1000 毫秒*/
        }
    }
```

设置好工程后，即可进行编译、连接。选择菜单 Project->Build target，对当前工程进行连接，如果当前文件已修改，将先对该文件进行编译，然后再连接以产生目标代码；如果选择 Rebuild All target files 将会对当前工程中的所有文件重新进行编译然后再连接，确保最终生产的目标代码是最新的，而 Translate ….项则仅对当前文件进行编译，不进行连接。以上操作也可以通过工具栏按钮直接进行。图 1-9 是有关编译、设置的工具栏按钮，从左到右分别是：编译、编译连接、全部重建、停止编译和对工程进行设置。

编译过程中的信息将出现在输出窗口中的 Build 页中，如果源程序中有语法错误，会有错误报告出现，双击该行，可以定位到出错的位置，对源程序修改之后再次编译，最终要得到如图 1-10 所示的结果，提示获得了名为 exam3.hex 的文件，该文件即可被编程器读入并写到芯片中，同时还可看到，该程序的代码量（code=63），内部 RAM 的使用量（data=9），外部 RAM 的使用量（xdata=0）等一些信息。除此之外，编译、连接还产生了一些其它相关的文件，可被用于 Keil 的仿真与调试，到了这一步后即进行调试。



1．5　程序的调试

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。

进入调试状态后，Debug 菜单项中原来不能用的命令现在已可以使用了，多出一个用于运行和调试的工具条，如图 1-11 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。

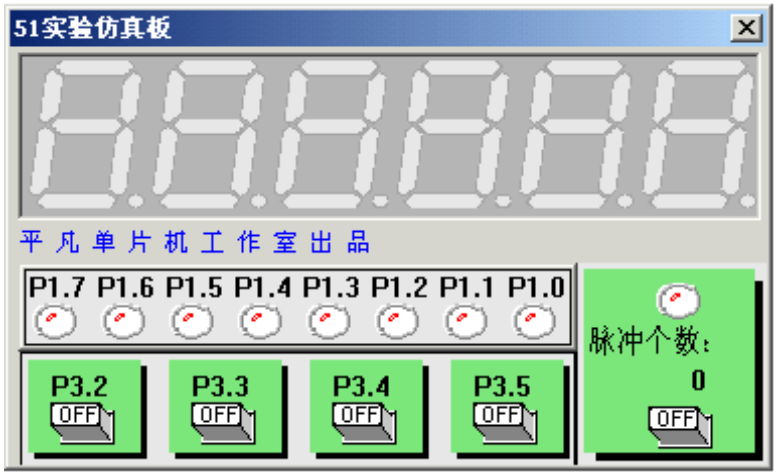
点击菜单 Peripherals，即会多出一项“键盘 LED 仿真板（K）”，选中该项，即会出现如

图 1-12 所示界面。

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，是指把 C 语言中的一个函数作为一条语句来全速执行。

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。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“mDelay(1000);”行时，再次按下 F11，会发现，箭头指向了延时子程序 mDelay 的第一行。不断按 F11 键，即可逐步执行延时子程序。

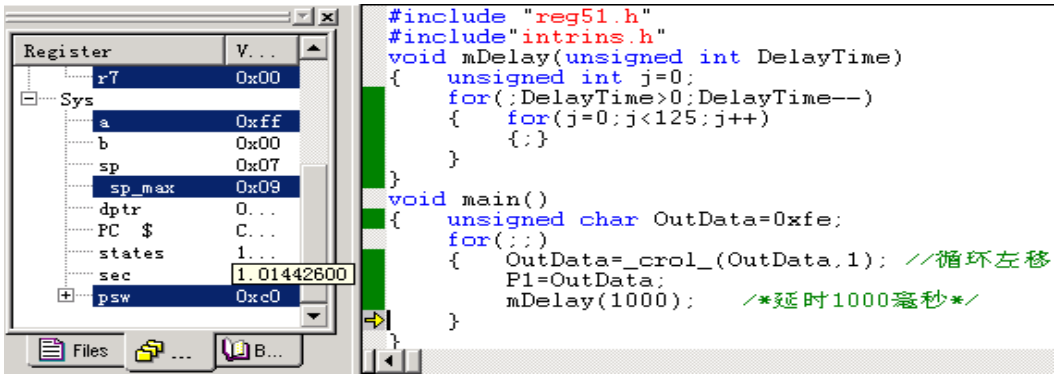
如果 mDelay 程序有错误，可以通过单步执行来查找错误，但是如果 mDelay 程序已正确，每次进行程序调试都要反复执行这些程序行，会使得调试效率很低，为此可以在调试时使用 F10 来替代 F11，在 main 函数中执行到 mDelay(1000)时将该行作为一条语句快速执行完毕。



Keil 软件还提供了一些窗口，用以观察一些系统中重要的寄存器或变量的值，这也是很重要的调试方法。

以下通过一个对延时程序的延迟时间的调整来对这些调试方法作一个简单的介绍。

这个程序中用到了延时程序 mDelay，如果使用汇编语言编程，每段程序的延迟时间可以非常精确地计算出来，而使用 C 语言编程，就没有办法事先计算了。为此，可以使用观察程序执行时间的方法了来解。进入调试状态后，窗口左侧是寄存器和一些重要的系统变量的窗口，其中有一项是 sec，即统计从开始执行到目前为止用去的时间。按 F10，以过程单步的形式执行程序，在执行到 mDelay(1000)这一行之前停下，查看 sec 的值（把鼠标停在 sec 后的数值上即可看到完整的数值），记下该数值，然后按下 F10，执行完 mDelay(1000)后再次观察 sec 值，如图 1-13 所示，这里前后两次观察到的值分别是：0.00040400 和 1.01442600，其差值为 1.014022s，如果将该值改为 124 可获得更接近于 1s 的数值，而当该值取 123 时所获得的延时值将小于 1s，因此，最佳的取值应该是 124。



1．6　C 语言的一些特点

通过上述的几个例子，可以得出一些结论：

1、C 程序是由函数构成的，一个 C 源程序至少包括一个函数，一个 C 源程序有且只有一个名为 main()的函数，也可能包含其它函数，因此，函数是 C 程序的基本单位。主程序通过直接书写语句和调用其它函数来实现有关功能，这些其它函数可以是由 C 语言本身提供给我们的（如例 3 中的_crol_（…）函数），这样的函数称之为库函数，也可以是用户自己编写的（如例 2、3 中用的 mDelay（…）函数），这样的函数称之为用户自定义函数。那么库函数和用户自定义函数有什么区别呢？简单地说，任何使用 Keil C 语言的人，都可以直接调用 C 的库函数而不需要为这个函数写任何代码，只需要包含具有该函数说明的相应的头文件即可；而自定义函数则是完全个性化的，是用户根据自己需要而编写的。Keil C 提供了 100 多个库函数供我们直接使用。

2、一个函数由两部份组成：

（1）函数的首部、即函数的第一行。包括函数名、函数类型、函数属性、函数参数（形参）名、参数类型。

例如：void　　mDelay　　(unsigned int DelayTime)

一个函数名后面必须跟一对圆括号，即便没有任何参数也是如此。

（2）函数体，即函数首部下面的大括号“{}”内的部份。如果一个函数内有多个大括号，则最外层的一对“{}”为函数体的范围。

函数体一般包括：

声明部份：在这部份中定义所用到的变量，例 1.2 中 unsigned char j。

执行部份：由若干个语句组成。

在某此情况下也可以没有声明部份，甚至即没有声明部份，也没有执行部份，如：

void mDelay()

{}

这是一个空函数，什么也不干，但它是合法的。

在编写程序时，可以利用空函数，比如主程序需要调用一个延时函数，可具体延时多少，怎么个延时法，暂时还不清楚，我们可以主程序的框架结构弄清，先编译通过，把架子搭起来再说，至于里面的细节，可以在以后慢慢地填，这时利用空函数，先写这么一个函数，这样在主程序中就可以调用它了。

3、一个 C 语言程序，总是从 main 函数开始执行的，而不管物理位置上这个 main()放在什么地方。例 1.2 中就是放在了最后，事实上这往往是最常用的一种方式。

4、主程序中的 mDelay 如果写成 mdelay 就会编译出错，即 C 语言区分大小写，这一点往往让初学者非常困惑，尤其是学过一门其它语言的人，有人喜欢，有人不喜欢，但不管怎样，你得遵守这一规定。

5、C 语言书写的格式自由，可以在一行写多个语句，也可以把一个语句写在多行。没有行号（但可以有标号），书写的缩进没有要求。但是建议读者自己按一定的规范来写，可以给自己带来方便。

6、每个语句和资料定义的最后必须有一个分号，分号是 C 语句的必要组成部份。

7、可以用/*…..*/的形式为 C 程序的任何一部份作注释，在“/*”开始后，一直到“*/”为止的中间的任何内容都被认为是注释，所以在书写特别是修改源程序时特别要注意，有时无意之中删掉一个“*/”，结果，从这里开始一直要遇到下一个“*/”中的全部内容都被认为是注释了。原本好好的一个程序，编译已过通过了，稍作修改，一下出现了几十甚至上百个错误，初学 C 的人往往对此深感头痛，这时就要检查一下，是不是有这样的情况，如果有的话，赶紧把这个“*/”补上。

特别地，Keil C 也支持 C++风格的注释，就是用“//”引导的后面的语句是注释，例：

```
P1_0=!P1_0;          //取反 P1.0
```

这种风格的注释，只对本行有效，所以不会出现上面的问题，而且书写比较方便，所以在只需要一行注释的时候，我们往往采用这种格式。但要注意，只有 Keil C 支持这种格式，早期的 Franklin C 以及 PC 机上用的 TC 都不支持这种格式的注释，用上这种注释，编译时通不过，会报告编译错误。

第 2 章　分支程序设计

第一部分课程学习了如何建立 Keil C 的编程环境，并了解了一些 C 语言的基础知识，这一部分将通过一个键控流水灯程序的分析来学习分支程序设计。

2．1　程序功能与实现

硬件电路描述如下：89S52 单片机的 P1 口接有 8 个 LED，当某一端口输出为“0”时，相应的 LED 点亮，P3.2、P3.3、P3.4、P3.5 分别接有四个按钮 K1~K4，按下按钮时，相应引脚被接地。现要求编写可键控的流水灯程序，当 K1 按下时，开始流动，K2 按下时停止流动，全部灯灭，K3 使灯由上往下流动，K4 使灯由下往上流动。

下面首先给出程序，然后再进行分析。

例 2-1：键控流水灯的程序

```
#include "reg51.h"
#include "intrins.h"
#define uchar unsigned char
void mDelay(unsigned int DelayTime)
{    unsigned int   j=0;
     for(;DelayTime>0;DelayTime--)
     {    for(j=0;j<125;j++)
          {;}  }}
uchar Key()
{    uchar KeyV;
     uchar tmp;
     P3=P3|0x3c;  //四个按键所接位置
     KeyV=P3;
     if((KeyV|0xc3)==0xff)  //无键按下
          return(0);
     mDelay(10);    //延时，去键抖
     KeyV=P3;
     if((KeyV|0xc3)==0xff)
          return(0);
     else
     {    for(;;){    tmp=P3;
               if((tmp|0xc3)==0xff)
                    break;}
          return(KeyV);}}
void main()
{    unsigned char OutData=0xfe;
     bit UpDown=0;
     bit   Start=0;
     uchar KValue;
```

```
    for(;;)
    {   KValue=Key();
        switch (KValue)
        {   case 0xfb:   //P3.2=0,Start
            {   Start=1;
                break;   }
            case 0xf7:   //P3.3=0,Stop
            {   Start=0;
                break;   }
            case 0xef:   //P3.4=0 Up
            {   UpDown=1;
                break;   }
            case 0xdf: //P3.5=0 Down
            {   UpDown=0;
                break;   }
        }
        if(Start)
        {   if(UpDown)
        OutData=_crol_(OutData,1);
            else
        OutData=_cror_(OutData,1);                  P1=OutData;
        }
        else
            P1=0xff;  //否则灯全灭
        mDelay(1000);
    }
}
```

输入源程序，保存为 exam21.c，建立名为 exam21 的工程文件，选择的 CPU 型号为 AT89S52，在 Debug 页加入-ddpj6，以便使用单片机实验仿真板，其他按默认设置。正确编译、链接后进入调试模式，点击 Peripherals→51 实验仿真板，打开实验仿真板，选择 Run（全速运行），此时实验仿真板没有变化，用鼠标点击上方的 K1 按钮，松开后即可看到 Led“流动”起来，初始状态是由下往上流动，点击 K3 按钮，可改变 LED 的流动方向，改为由上往下流动，点击 K4 按钮，又可将流动方向变换回来。点击 K2 按钮，可使流动停止，所有 LED“熄灭”。

2．1．1　程序分析

本程序中运用到了两种选择结构的程序：if 和 switch，if 语句最常用的形式是：

if(关系表达式)语句 1　else 语句 2

2．1．2　关系运算符和关系表达式

所谓“关系运算”实际上是两个值作一个比较，判断其比较的结果是否符合给定的条件。关系运算的结果只有 2 种可能，即“真”和“假”。例：3>2 的结果为真，而 3<2 的结果为假。

C 语言一共提供了 6 种关系运算符：“＜”（小于）、“＜＝”（小于等于）、“＞”（大于）、“＞＝（大于等于）”、“＝＝”（等于）和“！＝”（不等于）。

用关系运算符将两个表达式连接起来的式子，称为关系表达式。例：

a>b，a+b>b+c，（a=3）>=（b=5）等都是合法的关系表达式。关系表达式的值只有两种可能，即“真”和“假”。在 C 语言中，没有专门的逻辑型变量，如果运算的结果是“真”，用数值“1”表示，而运算的结果是“假”则用数值“0”表示。

如式子：x1=3>2 的结果是 x1 等于 1，原因是 3>2 的结果是“真”，即其结果为 1，该结果被“＝”号赋给了 x1，这里须注意，“＝”不是等于之意（C 语言中等于用“＝＝”表示），而是赋值号，即将该号后面的值赋给该号前面的变量，所以最终结果是 x1 等于 1。

式子：x2=3<=2 的结果是 x2=0，请自行分析。

2．2　逻辑运算符和逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来的式子就是逻辑表达式。C 语言提供了三种逻辑运算符：“&&”（逻辑与）、“||”（逻辑或）和“!”（逻辑非）。

C 语言编译系统在给出逻辑运算的结果时，用“1”表示真，而用“0”表示假，但是在判断一个量是否是“真”时，以 0 代表“假”，而以非 0 代表“真”，这一点务必要注意。以下是一些例子：

（1）　若 a=10，则！a 的值为 0，因为 10 被作为真处理，取反之后为假，系统给出的假的值为 0。

（2）　如果 a=-－2，结果与上完全相同，原因也同上，初学时常会误以为负值为假，所以这里特别提醒注意。

（3）　若 a=10，b=20，则 a&&b 的值为 1，a||b 的结果也为 1，原因为参于逻辑运算时不论 a 与 b 的值究竟是多少，只要是非零，就被当作是“真”，“真”与“真”相与或者相或，结果都为真，系统给出的结果是 1。

2．3 if 语句

if 语句是用来判定所给定的条件是否满足根据判定的结果（真或假）决定执行给出的两种操作之一。

C 语言提供了三种形式的 if 语句

1. if（表达式） 语句

   如果表达式的结果为真，则执行语句，否则不执行

2. if（表达式） 语句 1 else 语句 2

   如果表达式的结果为真，则执行语句 1，否则执行语句 2

3. if（表达式 1） 语句 1

   else if（表达式 2） 语句 2

   else if（表达式 3） 语句 3

   …

   else if（表达式 m） 语句 m

   else 语句 n

   这条语句执行如图 2 所示。

上述程序中的如下语句：

```
if((KeyV|0xc3)==0xff)     //无键按下
        return(0);
```

是第一种 if 语句的应用。该语句中“|”符号是 C 语言中的位运算符，按位相或的意思，相当于汇编语言中“ORL”指令，将读取的 P3 口的值 KeyV 与 0xc3（即 11000011B）按位或，如果结果为 0xff（即 11111111B）说明没有键被按下，因为中间 4 位接有按键，如果有键按下，那么 P3 口值的中间 4 位中必然有一位或更多位是“0”。该语句中的“return（0）”是返回之意，相当于汇编语言中的“ret”指令，通过该语句可以带返回值，即该号中的数值，返回值就是这个函数的值，在这个函数被调用时，用了如下的形式：KValue=Key();因此，返回的结果是该值被赋给 Kvalue 这个变量。因此，如果没有键被按下，则直接返回，并且 Kvalue 的值将变为 0。如果有键被按下，那么 return(0)将不会被执行。

程序其他地方还有这样的用法，请注意观察与分析。

程序中：

```
if(Start)
{… 灯流动显示的代码  }
else
     P1=0xff; //否则灯全灭
```

是 if 语句的第二种用法，其中 Start 是一个位变量，该变量在 main 函数的中被定义，并赋以初值 0，该变量在按键 K1 被按下后置为 1，而 K2 按下后被清为 0，用来控制灯流动是否开始。这里就是判断该变量并决定灯流动是否开始的代码，观察 if 后面括号中的写法，与其他语言中写法很不一样，并没有一个关系表达式，而仅仅只有一个变量名，C 根据这个量是 0 还是 1 来决定程序的走向，如果为 1 则执行灯流动显示的代码，如果为 0，则执行 P1=0xff；语句。可见，在 C 语言中，数据类型的概念比其他很多的编程语言要“弱化”，或者说 C 更着重从本质的角度去考虑问题，if 后面的括号中不仅可以是关系表达式，也可以是算术表达式，还可以就是一个变量，甚至是一个常量，不管怎样，C 总是根据这个表达式的值是零还是非零来决定程序的走向，这个特点是其他中所没有的，请注意理解。

if 语句的第三种用法在本程序中没有出现，下面我们举一例说明。在上述的键盘处理函

数 Key 中，如果没键被按下，返回值是 0，如果有键被按下，经过去键抖的处理，将返回键值，程序中的“return(KeyV);”即返回键值。当 K1 被按下（P3.2 接地）时，返回值是 0xfb（11111011B），而 K2 被按下（P3.3 接地）时，返回值是 0xf7（11110111B），K3 被按下（P3.4 接地）时，返回值是 0xef（11101111B），K4 被按下（P3.5 接地）时，返回值是 0xdf（11011111B），该值将被赋给主程序中调用键盘程序的变量 KValue。程序用了另一种选择结构 switch 进行处理，关于 switch 将在稍后介绍。下面用 if 语句来改写：

```
if（KValue==0xfb）
{Start=1;}
else if（KValue==0xf7）
{Start=0;}
else if（KValue==0xef）
{UpDown=1;}
else if（KValue==0xdf）
{UpDown=0;}
else
{//意外处理}
……
```

程序中第一条语句判断 Kvalue 是否等于 0xfb，如果是就执行 Start=1;执行完毕即退出 if 语句，执行 if 语句下面的程序，如果 Kvalue 不等于 0xfb 就转去下一个 else if 即判断 Kvalue 是否等于 0xf7，如果等于则执行 Start=0;，并退出 if 语句…这样一直到最后一个 else if 后面的条件判断完毕为止，如果所有的条件都不满足，那么就去执行 else 后面的语句（通常这意味着出现了异常，在这里来统一处理这种异常情况）。

2．4 if 语句的嵌套

在 if 语句中又包含一个或多个语句称为 if 语句的嵌套。一般形式如下

```
if()
  if() 语句 1
  else 语句 2
else
  if() 语句 3
  else 语句 4
```

应当注意 if 与 else 的配对关系，else 总是与它上面的最近的 if 配对。如果写成

```
if()
  if（）语句 1
else
  语句 2
```

编程者的本意是外层的 if 与 else 配对，缩进的 if 语句为内嵌的 if 语句，但实际上 else 将与缩进的那个 if 配对，因为两者最近，从而造迈岐义。为避免这种情况，建议编程时使用大括号将内嵌的 if 语句括起来，这样可以避免出现这样的问题。

2．5　swich 语句

当程序中有多个分支时，可以使用 if 嵌套实现，但是当分支较多时，则嵌套的 if 语层数多，程序冗长而且可读性降低。C 语言提供了 switch 语句直接处理多分支选择。Switch 的一般形式如下：

```
switch（表达式）
{case　常量表达式 1：语句 1
 case　常量表达式 2：语句 2
……
 case　常量表达式 n：语句 n
 default：语句 n+1
}
```

说明：switch 后面括号内的“表达式”，ANSI 标准允许它为任何类型；当表达式的值与某一个 case 后面的常量表达式相等时，就执行此 case 后面的语句，若所有的 case 中的常量表达式的值都没有与表达式值匹配的，就执行 default 后面的语句；每一个 case 的常量表达式的值必须不相同；各个 case 和 default 的出现次序不影响执行结果。

另外特别需要说明的是，执行完一个 case 后面的语句后，并不会自动跳出 switch，转而去执行其后面的语句，如上述例子中如果这么写

```
switch (KValue)
{     case 0xfb:   Start=1;
      case 0xf7:   Start=0;
      case 0xef:   UpDown=1;
      case 0xdf:   UpDown=0;
}
if(Start)
{     ……}
```

假如 KValue 的值是 0xfb，则在转到此处执行“Start=1;”后，并不是转去执行 switch 语句下面的 if 语句，而是将从这一行开始，依次执行下面的语句即“Start=0;”、“UpDown=1;”“UpDown=0;”，显然，这样不能满足要求，因此，通常在每一段 case 的结束加入“break;”语句，使流程序退出 switch 结构，即终止 switch 语句的执行。

第 3 章　数据类型

数据是计算机处理的对象，计算机要处理的一切内容最终将要以数据的形式出现，因此，程序设计中的数据有着很多种不同的含义，不同的含义的数据往往以不同的形式表现出来，这些数据在计算机内部进行处理、存储时往往有着很大的区别。下面我们来了解 C 语言数据类型的有关知识。

3．1　C 语言的数据类型概述

C 语言中常的数据类型有：整型、字符型、实型等。

C 语言中数据有常量与变量之分，它们分别属于以上这些类型。由以上这此数据类型还可以构成更复杂的数据结构，在程序中用到的所有的数据都必须为其指定类型。

3．2　常量与变量

在程序运行过程中，其值不能被改变的量称为常量。常量区分为不同的类型，如 12、0 为整型常量，3.14、2.55 为实型常量，‘a’、‘b’是字符型常量。

例 1 符号常量的使用，在 P1 口接有 8 个 LED，执行下面的程序：

```
#define LIGHT0      0xfe
#include “reg51.h”
void main()
{      P1=LIGHT0;
}
```

程序中用#define LIGHT0 0xfe 来定义符号 LIGHT0 等于 0xfe，以后程序中所有出现 LIGHT0 的地方均会用 0xfe 来替代，因此，这个程序执行结果就是 P1=0xfe，即接在 P1.0 引脚上的 LED 点亮。

这种用标识符代表的常量，称为符号常量。使用符号常量的好处是：

1．含义清楚。在单片机程序中，常有一些量是具有特定含义的，如某单片机系统扩展了一些外部芯片，每一块芯片的地址即可用符号常量定义，如:

```
#define PORTA 0x7fff
#define PORTB 0x7ffe
```

程序中可以用 PORTA、PORTB 来对端口进行操作，而不必写 0x7ff、0x7fe。显然，这两个符号比两个数字更能令人明白其含义。在给符号常量起名字时，尽量要做到“见名知意”以充分发挥这一特点。

2、在需要改变一个常量时能做到“一改全改”。如果由于某种原因，端口的地址发生了变化（如修改了硬件），由 0x7fff 改成了 0x3fff，那么只要将所定义的语句改动一下：

```
#define PORTA 0x3fff
```

即可，不仅方便，而且能避免出错。设想一下，如果不用符号常量，要在成百上千行程序中把所有表示端口地址的 0x7fff 找出来并改掉可不是件容易的事。

对于符号常量，初学者往往会和变量的概念混淆起来，它们之间有什么区别呢？

符号常量不等同于变量，它的值在整个作用域范围内不能改变，也不能被再次赋值。比

如下面的语句是错误的：

LIGHT=0x01;

值可以改变的量称为变量。一个变量应该有一个名字，在内存中占据一定的存储单元，在该存储单元中存放变量的值。请注意变量名与变量值的区别，下面从汇编语言的角度对此作一个解释。使用汇编语言编程时，必须自行确定 RAM 单元的用途，如某仪表有 4 位 LED 数码管，编程时将 3CH~3FH 作为显示缓冲区，当要显示一个字串“1234”时，汇编语言可以这样写：

```
MOV     3CH，#1
MOV     3DH，#2
MOV     3EH，#3
MOV     3FH，#4
```

经过显示程序处理后，在数码管上显示 1234。这里的 3CH 就是一个存储单元，而送到该单元中去的“1”是这个单元中的数值，显示程序中需要的是待显示的值“1”，但不借助于 3CH 又没有办法来用这个 1，这就是数与该数据在地址单元的关系。同样，在高级语言中，变量名仅是一个符号，需要的是变量的值，但是不借助于该符号又无法用该值。实际上如果在程序中写上“x1=5;”这样的语句，经过 C 编译程序的处理之后，也会变成“MOV 3CH，#5”之类的语句，只是究竟是使用 3CH 作为存放 x1 内容的单元还是其它如 3DH，4FH 等作为存放 x1 内容的单元，是由 C 编译器确定的。

用来标识变量名、符号常量名、函数名、数组名、类型名等的有效字符序列称为标识符。简单地说，标识符就是一个名字。

C 语言规定标识符只能由字母、数字和下划线三种字符组成，且第一个字符必须为字母或下划线，要注意的是 C 语言中大写字母与小写字母被认为是两个不同的字符，即 Sum 与 sum 是两个不同的标识符。

标准的 C 语言并没有规定标识符的长度，但是各个 C 编译系统有自己的规定，在 Keil C 编译器中可以使用长达数十个字符的标识符。

在 C 语言中，要求对所有用到的变量作强制定义，也就是“先定义，后使用”。

初学者往往难于理解常量和变量在程序中各有什么用途，这里再举个例子加以说明。

前面的课程中我们多次用到延时程序，其中调用延时程序是这么写的：

```
mDelay(1000);
```

这其中括号中参数 1000 决定了灯流动的速度，在这些程序中我们并未对灯流动的速度有要求，因此，直接将 1000 写入程序中即可，这就是常量。显然，这个数据是不能在现场修改的，如果使用中有人提出希望改变流水灯的速度，那么只能重新编程、写片才能更改。

如果要求在现场有修改流水灯速度的要求，括号中就不能写入一个常数，为此可以定义一个变量（如 Speed），写程序时这么写：mDelay(Speed);然后再编写一段程序，使得 Speed 的值可以被通过按键被修改，那么流水灯流动的速度就可以在现场修改了，显然这时就需要用到变量了。

3．3　字符型数据与整型数据

了解了变量与常量的关，再来看一看不同数据类型究竟有什么区别。前面程序中的延时程序是这么写的：

```
void mDelay(unsigned int DelayTime)
{     unsigned int    j=0;
      for(;DelayTime>0;DelayTime--)
```

```
    {    for(j=0;j<125;j++)
         {;}
    }
}
```

在 main 函数中用 mDelay(1000)的形式调用该函数时，延时时间约为 1s。如果将该函数中的 unsigned int j 改为 unsigned char j，其他任何地方都不作更改，重新编译、连接后，可以发现延迟时间变为约 0.38s。int 和 char 是 C 语言中的两种不同的数据类型，可见程序中仅改变数据类型就会得到不同的结果。那么 int 和 char 型的数据究竟有什么区别呢？

3．3．1　整型数据

1．整型数据在内存中的存放形式

如果定义了一个 int 型变量 i:

int i=10; /*定义 i 为整型变量，并将 10 赋给该变量*/

在 Keil C 中规定使用二个字节表示 int 型数据，因此，变量 i 在内存中的实际占用情况如下：

0000,0000,0000,1010

也就是整型数据总是用 2 个字节存放，不足部分用 0 补齐。

事实上，数据是以补码的形式存在的。一个正数的补码和其原码的形式是相同的。如果数值是负的，补码的形式就不一样了。求负数的补码的方法是：将该数的绝对值的二进制形式取反加 1.例如，-10，第一步取-10 的绝对值 10，其二进制编码是 1010，由于是整型数占 2 个字节（16 位），所以其二进制形式实为 0000，0000，0000，1010，取反，即变为 1111，1111，1111，0101，然后再加 1 变成了 1111，1111，1111，0110，这个就是数-10 在内存中的存放形式。这里其实只要搞清一点，就是必须补足 16 位，其它的都不难理解。

2．整型变量的分类

整型变量的基本类型是 int，可以加上有关数值范围的修饰符。这些修饰符分两类，一类是 short 和 long，另一类是 unsigned，这两类可以同时使用。下面就来看有关这些修饰符的内容。

在 int 前加上 short 或 long 是表示数的大小的，对于 keil C 来说，加 short 和不加 short 是一模一样的（在有一些 C 语言编译系统中是不一样的），所以，short 就不加讨论了。如果在 int 前加上 long 的修饰符，那么这个数就被称之为长整数，在 keil C 中，长整数要用 4 个字节来存放（基本的 int 型是 2 个字节）。显然，长整数所能表达的范围比整数要大，一个长整数表达的范围可以有：

$-2^{31}<x<2^{31}-1$

大概是在正负 21 亿多。而不加 long 修饰的 int 型数据的范围是-32768~32767，可见，二者相差很远。

第二类修饰符是 unsigned 即无符号的意思，如果加上了这样的一个修饰符，就说明其后的数是一个无符号的数，无符号、有符号的差别还是数的范围不一样。对于 unsigned int 而言，仍是用 2 个字节（16 位）表示一个数，但其数的范围是 0~65535，对于 unsigned long int 而言，仍是用 4 个字节（32 位）表示一个数，但其数的范围是 $0\sim2^{32}-1$。

3．3．2　字符型数据

1．字符型数据在内存中的存放形式

数据在内存中是以二进制形式存放的，如果定义了一个 char 型变量 c：

char c=10; /*定义 c 为字符型变量，并将 10 赋给该变量*/

十进制数 10 的二进制形式为 1010，在 Keil　C 中规定使用一个字节表示 char 型数据，因此，变量 c 在内存中的实际占用情如下：

0000,1010

弄明白了整型数据和字符型数据在内存中的存放，两者在前述程序中引起的差别就不难主理解了，当使用 int 型变量时，程序需要对 16 位二进制码运算，而 80C51 是 8 位机，一次只能处理 8 位二进制码，所以就要分次处理，因此延迟时间就变长了。

2．字符型变量的分类

字符型变量只有一个修饰符　unsigned 即无符号的。对于一个字符型变量来说，其表达的范围是-128~+127，而加上了 unsigned 后，其表达的范围变为 0~255。

加了 unsigned 和没有加究竟有何区别呢？其实对于二进制形式而言，char 型变量表达的范围都是 0000,0000~1111,1111，而 int 型变量表达的范围都是 0000,0000,0000,0000~1111,1111,1111,1111，只是我们对这些二进制数的理解不一样而已。

使用 keil C 时，不论是 char 型还是 int 型，我们都非常喜欢用 unsigned 型的数据，这是因为在处理有符号的数时，程序要对符号进行判断和处理，运算的速度会减慢。

对单片机而言，速度比不上 PC 机，又工作于实时状态，任何提高效率的手段都要考虑。

3．字符的处理

在一般的 C 语言中，字符型变量常用处理字符，如：

char c=’a’;

之类等，即是定义一个字符型的变量 c，然后将字符 a 赋给该变量。进行这一操作时，实际是将字符 a 的 ASCII 码值赋给变量 c，因此，做完这一操作之后，c 的值是 97。

既然字符最终也是以数值来存储的，那么和以下的语句：

int i=97;

究竟有多大的区别呢？实际上它们是非常类似的，区别仅仅在于 i 是 16 位的，而 c 是 8 位的，当 i 的值不超过 255 时，两者完全可以互换。C 语言对定符型数据作这样的处理使用得程序设计时增大了自由度。典型地，在 C 语言中要将一个大写字母转化为一个小写字母，只要简单地将该变量加上 32 即可（查 ASCII 码表可以看到任意一个大写字母比小写字母小 32）。由于这一点，我们在单片机中往往是把字符型变量当成一个“8 位的整型变量”来用。

4．数的溢出

一个字符型数的最大值是 127，一个整型数的最大值是 32767，如果再加 1，会出现什么情况呢？下面我们用一个例子来说明。

例：演示字符型数据和整型数据溢出的例子

```
#includee “reg51.h”
void main()
{    unsigned char a,b;
     int c,d;
     a=255;
     c=32767;
```

```
    b=a+1;
    d=a+1;
}
```

输入该文件，命名为 exam23.c，建立工程，加入该文件，在 C 优化页将优化级别设为 0，避免 C 编译器认为这种程序无意义而自动优化使我们不能得到想要的结果。编译、连接后，运行，查看变量，如图 3-1 所示。

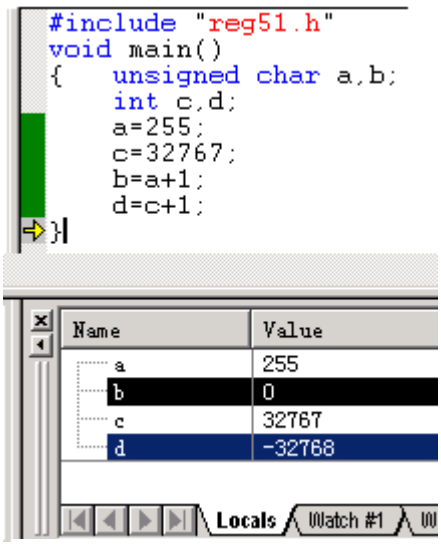


图 3-1　数的溢出

可见，b 和 d 在加 1 之后分别变成了 0 和-32768，这是为什么呢？这与我们的数学计算显然不同。其实只要我们从数字在内存中的二进制存放形式分析，就不难理解。

首先看变量 a，该变量的值是 255，类型是无符号字符型，因此，该变量在内存中以 8 位（一个字节）来存放，将 255 转化为二进制即 1111,1111，如果将该值加 1，结果是 1,0000,0000，由于该变量只能存放 8 位，所以最高位的 1 丢失，于是该数字就变也了 0000,0000，自然就是十进制的 0 了。其实这不难理解，录音机上有磁带计数器，共有 3 位，当转到 999 后，再转一圈，本应是 1000，但实际看到的是 000，除非你借助于其他方法，否则你是无法判断其是转了 1000 转还是根本没有动。

在理解了无符号的字符型数据的溢出后，整型变量的溢出也不难理解。32767 在内存中存放的形式是 0111,1111,1111,1111，当其加 1 后就变成了 1000,0000,0000,0000，而这个二进制数正是－32768 在内存中的存放形式，所以 c 加 1 后就变成了-32768。

可见，在出现这样的问题时 C 编译系统不会给出提示（其他语言中 BASIC 等会报告出错），这有利于编出灵活的程序来，但也会引起一些副作用，这就要求 C 程序员对硬件知识有较多的了解，对于数在内存中的存放等基本知识必须清楚。

第 4 章　循环程序设计

前面的课程中学习了分支程序的设计，下面学习程序设计中另一种常用的程序结构－－循环结构。

4．1　循环程序简介

在一个实用的程序中，循环结构是必不可少的。循环是反复执行某一部分程序行的操作。有两类循环结构：

（1）当型循环，即当给定的条件成立时，执行循环体部分，执行完毕回来再次判断条件，如果条件成立继续循环，否则退出循环。

（2）直到型循环，即先执行循环体，然后判断给定的条件，只要条件成立就继续循环，直到判断出给定的条件不成立时退出循环。

下面我们就通过一些例子来看 C 语言提供的循环语句，及如何利用这些循环语句写循环程序。

例 4-1　使 P1 口所接 LED 以流水灯状态显示

```
#include "reg51.h"
#include"intrins.h" //该文件包含有_crol_(…)函数的说明
void mDelay(unsigned int DelayTime)
{    unsigned int   j=0;
     for(;DelayTime>0;DelayTime--)
     {    for(j=0;j<125;j++)
          {;}
     }
}
void main()
{    unsigned char OutData=0xfe;
  while(1)
  { P1=OutData;
    OutData=_crol_(OutData,1); //循环左移
    mDelay(1000);      /*延时 1000 毫秒*/
    }
}
```

程序分析：输入源程序，并命名为 exam31.c，建立并设置工程，这个例子使用实验仿真板演示的过程请自行完成。如果在演示时，发现灯“流动”的速度太快，几乎不能看清，那么可以将 mDelay(1000)中的 1000 改大一些，如 2000、3000 或更大。软件仿真无法实现硬件实验一样的速度，这是软件仿真的固有弱点，下面介绍如何用具有仿真功能的实验板来实现这个例子。

将随机带的一根串口电缆一端连接到 PC 机的某一个串口，另一端连到本实验板上，设置工程，选中 Debug 页，点击右侧的“Use　Keil Monitor-51 Drive”，然后选中“Load Application at Start”和“Go Till main”，如图 4-1 所示。选择完成后，点击“Setting”按钮，选择你所用的 PC 上的串口（COM1 或 COM2），波特率（通常可以使用 38400），其他设置一般不需

要更改，如图 4-2 所示。点击“OK”回到 Debug 页面后即可完成设置。

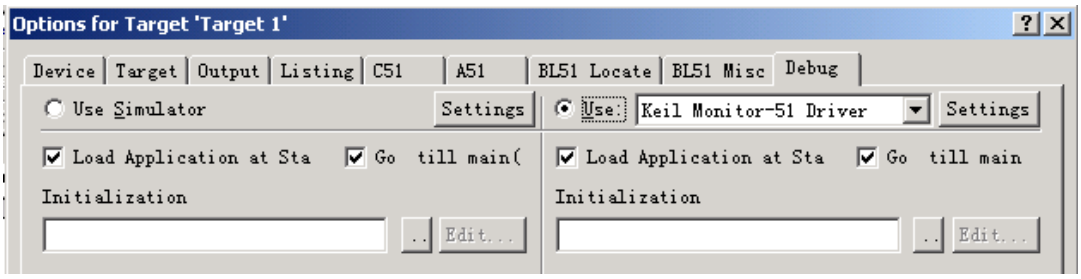


图 4-1　设置 Debug 页

编译、连接正确后，点击菜单 Debug→Start /Stop Debug Session，可以看到在窗口右下角的命令窗口提示正确连接到了 Monitor-51，如图 4-3 所示。此时，即可使用 Keil 提供的单步、过程单步、执行到当前行、设置断点等调试方法进行程序的调试。如果全速运行程序，

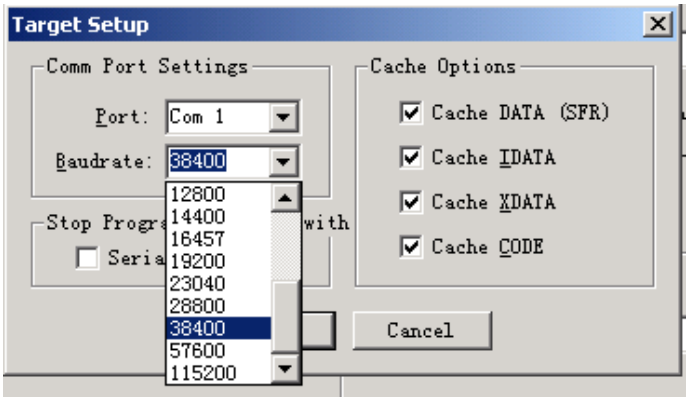


图 4-2　选择串口、波特率及其他选项

可看到流水灯的实验效果。

程序分析：这段程序中在两处用到了循环语句，首先是主程序中使用了:

while（1）

｛……｝

这样的循环语句写法，在{}中的所有程序将会不断地循环执行，直到断电为止；其次是

图 4-3　正确进入程序调试

延时程序，使用了 for 循环语句的形式。下面我们就对循环语句作一个介绍。

4．2　while 语句

While 语句用到实现“当型”循环结构，其一般形式如下：

while(表达式)　语句

当表达式为非 0 值（真）时，执行 while 语句中的内嵌语句。其特点是：先判断表达式，

后执行语句。

在上述例子中，表达式使用了一个常数“1”，这是一个非零值，即“真”，条件总是满足，语句总是会被执行，构成了无限循环。下面再举一例说明：

例 4-2：当 K1 键被按下时，流水灯工作，否则灯全部熄灭。

```
#include "reg51.h"
#include"intrins.h" //该文件包含有_crol_(…)函数的说明
void mDelay(unsigned int DelayTime)
{    unsigned int   j=0;
     for(;DelayTime>0;DelayTime--)
     {    for(j=0;j<125;j++)
          {;}
     }
}
void main()
{    unsigned char OutData=0xfe;
     while(1)
     {    P3|=0x3c;
          while((P3|0xfb)!=0xff)
          {    P1=OutData;
               OutData=_crol_(OutData,1); //循环左移
               mDelay(1000);      /*延时 1000 毫秒*/}
               P1=0xff;
          }
     }
```

程序分析：这个程序中的第二个 while 语句中的表达式用来判断 K1 键是否被按下，如被按下，则执行循环体内的程序，否则执行 P1=0xff;程序行。虽然整个程序是在一个无限循环过程中，但是由于外界条件的变化使得程序执行的过程发生了变化。

4．3　do-while 语句

do-while 语句用来实现“直到型”循环，特点是先执行循环体，然后判断循环条件是否成立。其一般形式如下：

```
do
  循环体语句
while(表达式)
```

对同一个问题，既可以用 while 语句处理，也可以用 do-while 语句处理。但是这两个语句是有区别的，下面我们用 do-while 语句改写例 2。

例 4-3：用 do-while 语句实现如下功能：K1 按下，流水灯工作，K2 松开，灯全熄灭。

```
#include "reg51.h"
#include"intrins.h" //该文件包含有_crol_(…)函数的说明
void mDelay(unsigned int DelayTime)
{    unsigned int   j=0;
     for(;DelayTime>0;DelayTime--)
     {    for(j=0;j<125;j++)
```

```
            {;}
        }
}
void main()
{   unsigned char OutData=0xfe;
    while(1)
    {   P3|=0x3c;
        do
        {   P1=OutData;
            OutData=_crol_(OutData,1); //循环左移
            mDelay(1000);      /*延时 1000 毫秒*/
        } while((P3|0xfb)!=0xff)
        P1=0xff;
    }
}
```

程序分析：这个程序除主程序中将 while 用 do-while 替代外，没有其他的变化，初步设想，如果 while（）括号中的表达式为“真”即 K1 键被按下，应该执行程序体，否则不执行，效果与例 4-2 相同。但是事实上，实际做这个练习就会发现，不论 K1 是否被按下，流水灯都在工作。为何会有这么样的结果呢？

单步运行程序可以发现，如果 K1 键被按下，的确是在执行循环体内的程序，与设想相同。而当 K1 没有被按下时，按设想，循环体内的程序不应该被执行，但事实上，do 后面的语句至少要被执行一次才去判断条件是否成立，所以程序依然会去执行 do 后的循环体部分，只是在判断条件不成立（K1 没有被按下）后，转去执行 P1=0xff;然后又继续循环，而下一次循环中又会先执行一次循环体部分，因此，K1 是否被按下的区别仅在于“P1=0xff;”这一程序行是否会被执行到。

4．4　for 语句

C 语言中的 for 语句使用最为灵活，不仅可以用于循环次数已经确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况。

for 语句的一般形式为：

for（表达式 1；表达式 2；表达式 3）　语句

它的执行过程是：

（1）　先求解表达式 1

（2）　求解表达式 2，其值为真，则执行 for 语句中指定的内嵌语句（循环体），然后执行第（3）步，如果为假，则结束循环。

（3）　求解表达式 3

（4）　转回上面的第（2）步继续执行。

for 语句典型的应用是这样一种形式：

for(循环变量初值;循环条件;循环变量增值)　语句

例如上述例子中的延时程序有这样的程序行：“for(j=0;j<125;j++){;}　”，执行这行程序时，首先执行 j=0，然后判断 j 是否小于 125，如果小于 125 则去执行循环体（这里循环体没有做任何工作），然后执行 j++，执行完后再去判断 j 是否小于 125……如此不断循环，直

到条件不满足（j>=125）为止。

如果用 while 语句来改写，应该这么写

```
j=0;
while(j<125)
{   j++;   }
```

可见，用 for 语句更简单、方便。

如果变量初值在 for 语句前面赋值，则 for 语句中的表达式 1 应省略，但其后的分号不能省略。上述程序中有：“for(;DelayTime>0;DelayTime--){…}”的写法，省略掉了表达式 1，因为这里的变量 DelayTime 是由参数传入的一个值，不能在这个式子里赋初值。

表达式 2 也可以省略，但是同样不能省略其后的分号，如果省略该式，将不判断循环条件，循环无终止地进行下去，也就是认为表达式始终为真。

表达式 3 也可以省略，但此时编程者应该另外设法保证循环能正常结束。

表达式 1、2 和 3 都可以省略，即形成如 for(;;)的形式，它的作用相当于是 while(1)，即构一个无限循环的过程。

循环可以嵌套，如上述延时程序中就是两个 for 语句嵌套使用构成二重循环，C 语言中的三种循环语句可以相互嵌套。

4．5　break 语句

在一个循环程序中，可以通过循环语句中的表达式来控制循环程序是否结束，除此之外，还可以通过 break 语句强行退出循环结构。

例 4：开机后，全部 LED 不亮，按下 K1 则从 LED1 开始依次点亮，至 LED8 后停止并全部熄灭，等待再次按下 K1 键，重复上述过程。如果中间 K2 键被按下，LED 立即全部熄灭，返回起始状态。

```
#include "reg51.h"
#include"intrins.h" //该文件包含有_crol_(…)函数的说明
void mDelay(unsigned int DelayTime)
{    unsigned int   j=0;
     for(;DelayTime>0;DelayTime--)
     {    for(j=0;j<125;j++)
          {;}
     }
}
void main()
{    unsigned char OutData=0xfe;
     unsigned char i;
     while(1)
     {    P3|=0x3c;
          if((P3|0xfb)!=0xff) //K1 键被按下
          {    OutData=0xfe;
               for(i=0;i<8;i++)
               {    mDelay(1000);      /*延时 1000 毫秒*/
                    tmp=0xfe;
```

```
                    if((P3|0xf7)!=0xff) //K2 键被按下
                        break;
                    OutData=_crol_(OutData,i);
                    P1&=OutData;
                }
            }
            P1=0xff;
        }
    }
```

请读者输入程序、建立工程，使用实验仿真板或者实验板来验证这一功能，注意，K2 按下的时间必须足够长，因为这里每 1s 才会检测一次 K2 是否被按下。

程序分析：开机后，当检测到 K1 键被按下，执行一个:

for(i=0;i<8;i++)

{…}

的循环，即循环 8 次后即停止，而在这段循环体中，又用到了如下的程序行：“if((P3|0xf7)!=0xff) break;”即判断 K2 是否按下，如果 K2 被按下，则立即结束本次循环。

4．6 continue 语句

该语句的用途是结束本次循环，即跳过循环体中下面的语句，接着进行下一次是否执行循环的判定。

Continue 语句和 break 语名的区别是：continue 语句只结束本次循环，而不是终止整个循环的执行；而 break 语句则是结束整个循环过程，不会再去判断循环条件是否满足。

例 5：将上述例 4 中的 break 语句改为 continue 语句，会有什么结果？

程序分析：开机后，检测到 K1 键被按下，各灯开始依次点亮，如果 K2 键没有被按下，将循环 8 次，直到所有灯点亮，又加到初始状态，即所有灯灭，等待 K1 按键。如果 K2 键被按下，不是立即退出循环，而只是结束本次循环，即不执行 continue 语句下面的“OutData=_crol_(OutData,i); P1&=OutData;”语句，但要继续转去判断循环条件是否满足，因此，不论 K2 键是否被按下，循环总是要经过 8 次才会终止，差别在于是否执行了上述两行程序。如果上述程序行有一次未被执行，意味着有一个 LED 未被点亮，因此，如果按下 K2 过一段时间（1、2s）松开，中间将会有一些 LED 不亮，直到最后一个 LED 被点亮，又回到全部熄灭的状态，等待 K1 被按下。

练习：基本要求同例 5，但不是在按下 K2 后有一些灯不亮，而是固定每点亮 2 个 LED 后，第三个 LED 不亮，请编程实现。

第 5 章　单片机内部资源编程

通过前面课程的学习，我们已了解了“通用”的 C 语言特性，本课将介绍针对 80C51 单片机特性的 C 语言编程。

5．1 定时器编程

定时器编程主要是对定时器进行初始化以设置定时器工作模式，确定计数初值等，使用 C 语言编程和使用汇编编程方法非常类似，以下通过一个例子来分析。

例 5-1：用定时器实现 P1 所接 LED 每 60ms 亮或灭一次，设系统晶振为 12M。

参考图 5-1 输入源程序，建立并设置工程，本例使用实验仿真板难以得到理想的结果，应使用 DSB-1A 型实验板进行练习。

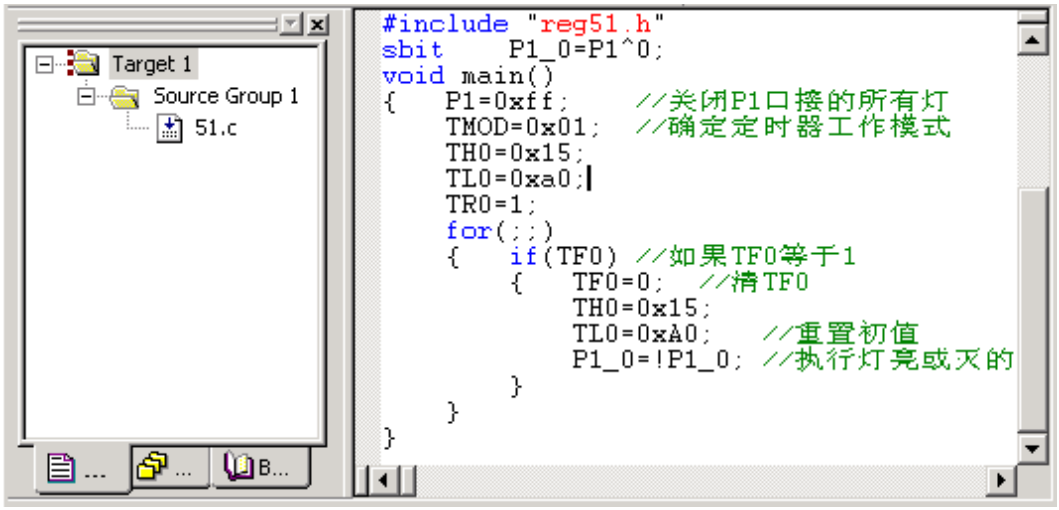


图 5-1　用定时器实现 LED 闪烁

分析：要使用单片机的定时器，首先要设置定时器的工作方式，然后给定时器赋初值，即进行定时器的初始化。这里选择定时器 0，工作于定时方式，工作方式 1，即 16 位定时/计数的工作方式，不使用门控位。由此可以确定定时器的工作方式字 TMOD 应为 00000001B，即 0x01。定时初值应为 65536－60000＝5536，由于不能直接给 T0 赋值，必须将 5536 化为十六进制即为 0x15a0，这样就可以写出初始化程序即：

TMOD＝0x01；

TH0＝0x15；

TL0＝0xa0；

初始化定时器后，要定时器工作，必须将 TR0 置 1，程序中用“TR0＝1；”来实现。

可以使用中断也可以使用查询的方式来使用定时器，本例使用查询方式，中断方式稍后介绍。

当定时时间到后，TF0 被置为 1，因此，只需要查询 TF0 是否等于 1 即可得知定时时间是否到达，程序中用“if(TF0){…}”来判断，如果 TF0＝0，则条件不满足，大括号中的程序行不会被执行到，当定时时间到 TF1＝1 后，条件满足，即执行大括号中的程序行，首先将 TF0 清零，然后重置定时初值，最后是执行规定动作－－取反 P1.0 的状态。

5．2　中断编程

C51 编译器支持在 C 源程序中直接开发中断过程，使用该扩展属性的函数定义语法如下：

返回值　函数名　interrupt n

其中 n 对应中断源的编号，其值从 0 开始，以 80C51 单片机为例，编号从 0~4，分别对应外中断 0、定时器 0 中断、外中断 1、定时器 1 中断和串行口中断。

5．2．1　中断应用实例

下面我们同样通过一个例子来说明中断编程的应用。

例 5-2　用中断法实现定时器控制 P1.0 所接 LED 以 60ms 闪烁。

参考图 2 输入源程序，设置工程，同样，本例用实验仿真板难以看到真实的效果，应使用 DSB-1A 型实验板来完成这一实验。

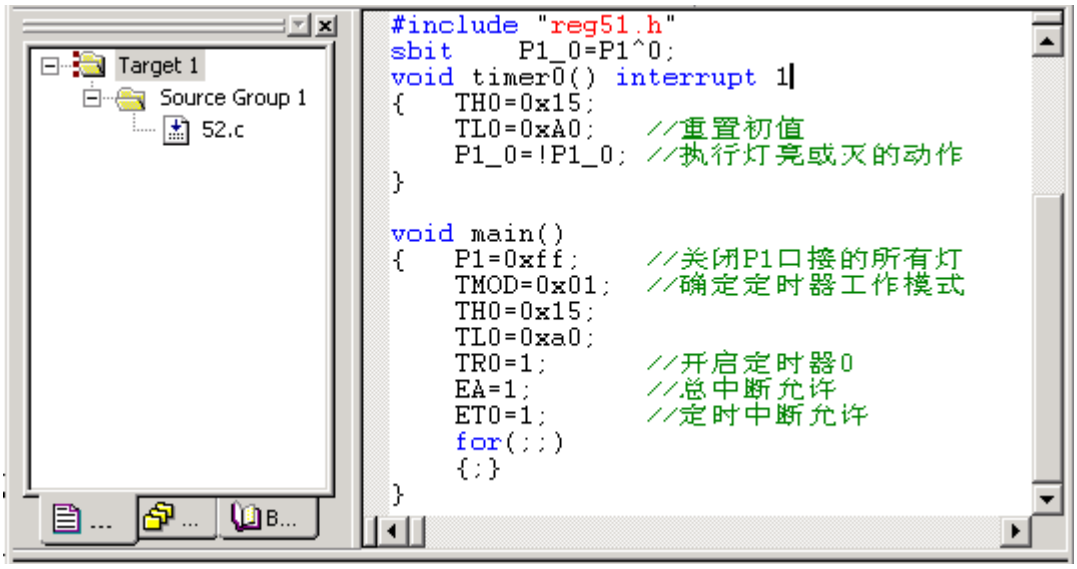


图 2　用中断法使用定时器

分析：本例与例 1 的要求相同，唯一的区别是必须用中断方式来实现。这里仍选用定时器 T0，工作于方式 1，无门控，因此，定时器的初始化操作与上例相同。要开启中断，必须将 EA（总中断允许）和 ET0（定时器 T0 中断允许）置 1，程序中用“EA＝1;”和“ET0＝1;”来实现。在做完这些工作以后，就用 for(;;){;}让主程序进入无限循环中，所有工作均由中断程序实现。

由于定时器 0 的中断编号为 1，所以中断程序中这样写：

void timer0() interrupt 1

{…}

可见，用 C51 语言写中断程序是非常简单的，只要简单地在函数名后加上 interrupt 关键字和中断编号就成了。

5．2．2　寄存器组切换

为进行中断的现场保护，80C51 单片机除采用堆栈技术外，还独特地采用寄存器组的方式，在 80C51 中一共有 4 组名称均为 R0~R7 的工作寄存器，中断产生时，可以通过简单地设置 RS0、RS1 来切换工作寄存器组，这使得保护工作非常简单和快速。使用汇编语言时，内存的使用均由编程者设定，编程时通过设置 RS0、RS1 来选择切换工作寄存器组，但使用 C 语言编程时，内存是由编译器分配的，因此，不能简单地通过设置 RS0、RS1 来切换工作寄存器组，否则会造成内存使用的冲突。

在 C51 中，寄存器组选择取决于特定的编译器指令，即使用 using　n 指定，其中 n 的值是 0~3，对应使用四组工作寄存器。

例如上述例子中可以这么样来写：

```
void timer0() interrupt 1 using 2
{…}
```

即表示在该中断程序中使用第 2 组工作寄存器。

5．3　串行口编程

80C51 系列单片机片上有 UART 用于串行通信，80C51 中有两个 SBUF，一个用作发送缓冲器，一个用作接收缓冲器，在完成串口的初始化后，只要将数据送入发送 SBUF，即可按设定好的波特率将数据发送出去，而在接收到数据后，可以从接收 BUF 中读到接收到的数据。下面我们通过一个例子来了解串行口编程的方法。

例 3　单片机 P1 口接 8 只发光二极管，P3.2~P3.5 接有 K1~K4 共四个按键，使用串行口编程，1）由 PC 机控制单片机的 P1 口，将 PC 机送出的数以二进制形式显示在发光二极管上；2）按下 K1 向主机发送数字 0x55，按下 K2 向主机发送数字 0xAA，使显示转下一行。

```
#define uchar unsigned char
#include "string.h"
#include "reg51.h"
void SendData(uchar Dat)
{    uchar i=0;
     SBUF=Dat;
     while(1){       if(TI)
          {    TI=0;
               break;}}
     }
void mDelay(unsigned int DelayTime)
{    unsigned char j=0;
     for(;DelayTime>0;DelayTime--)
     {    for(j=0;j<125;j++)
          {;}
     }
}
uchar Key()
{    uchar KValue;
```

```
    P3|=0x3e;       //中间 4 位置高电平
    if((KValue=P3|0xe3)!=0xff)
    {   mDelay(10);
        if((KValue=P3|0xe3)!=0xff)
        {   for(;;)
            if((P3|0xe3)==0xff)
                return(KValue);
        }
    }
    return(0);
}

void main()
{   uchar KeyValue;
    P1=0xff; //关闭 P1 口接的所有灯
    TMOD=0x20; //确定定时器工作模式
    TH1=0xFD;
    TL0=0xFD;   //定时初值
    PCON&=0x80;     //SMOD=1
    TR1=1;          //开启定时器 1
    SCON=0x40;  //串口工作方式 1
    REN=1;          //允许接收
    for(;;)
    {   if(KeyValue=Key())
        {   if((KeyValue|0xfb)!=0xff)     //K1 按下
                SendData(0x55);
            if((KeyValue|0xf7)!=0xff)
                SendData(0xaa);
        }
        if(RI)
        {   P1=SBUF;
            RI=0;
        }
    }
}
```

实现过程：输入程序，命名为 exam53.c，建立名为 exam53 的工程，将文件加入，设置工程，使用实验仿真板进行调试。正确编译连接后进入调试，打开实验仿真板，然后再点击 view→serial #1 打开串行窗口，在窗口空白处点右键，在弹出式菜单中选择“Hex Mode”如图 3 所示。

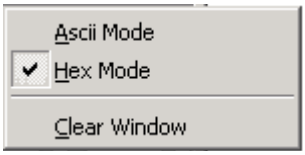


图 5-3　选择显示模式

单击实验仿真板的 K1 键和 K2 键，即可看到在串行窗口中分别出现 55 和 AA；单击串行窗口的空白处，使其变为活动窗口，即可接收键盘输入，按下键盘上不同的字符键，可见实验仿真板上的 LED 产生相应的变化。图 5-4 是按下 K1 一次、K2 连续两次、再按一次 K1 后看到的串行窗口现象，而实验仿真板则是在键盘上按下字符 1 之后看到的现象，灯亮为“0”，灯灭为“1”，因此灯的组合

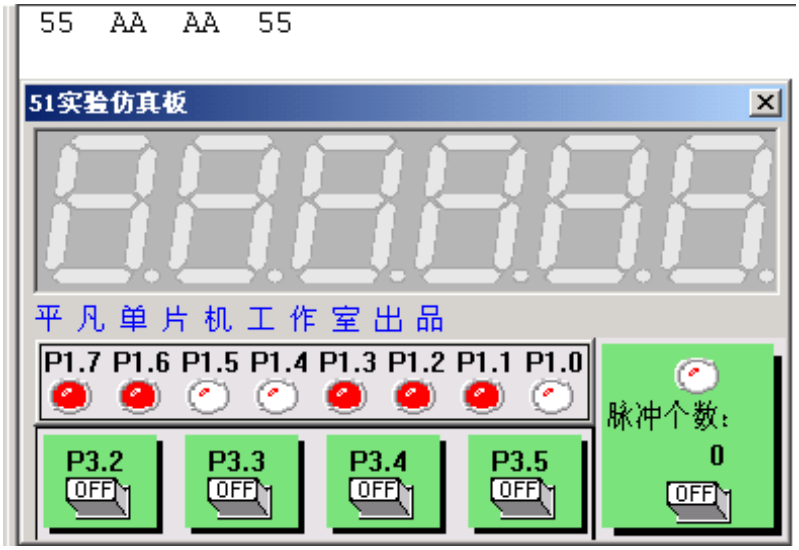


图 4　使用实验仿真板演示串行口操作

为 00110001，即 0x31，这正是字符 1 的 ASCII 码值。

程序分析：本程序使用 T1 作为波特率发生器，工作于方式 2（8 位自动重装入方式），波特率为 19200，串行口工作于方式 1，根据以上条件不难算出 T1 的定时初值为 0xfd，TMOD 应初始化为 0x20，SMOD 应初始化为 0x30，而 PCON 中的 SMOD 位必须置 1，主程序 main 的开头对这些初值进行了设置。设置好初值后，使用“TR1＝1;”开启定时器 1，使用“REN＝1;”允许接收数据，然后即进入无限循环中开始正常工作。在这个无限循环中首先调用键盘程序，检测是否有键按下，如果有键按下，那么检测是否 K1 被按下，如果 K1 被按下，则调用发送数据程序，将数据 0x55 送出，如果 K2 被按下，则将数据 0xAA 送出。然后检测 RI 是否等于 1，如果 RI 等于 1，说明接收到字符，清 RI，准备下一次接收，并将接收到的数据送往 P1 口显示。这样，一次循环结束，继续开始下一次循环。

图 5-5 设置串口参数

发送函数 SendData 中有只有一个参数 Dat，即待发送的字符，函数将待发送的字符送入 SBUF 后，使用一个无限循环等待发送的结束，在循环中通过检测 TI 来判断数据是否发送完毕，发送完毕使用 break 语句退出循环。

如果使用 DSB－1A 型实验板做实验，需要用到一个 PC 端的串口调试程序，并正确设置该调试程序的有关参数，这里以“串口调试助手”软件为例，其参数设置如图 5-5 所示。

由于该板占用了串口，因此做串口通讯类实验只能用下载全速运行的方法，具体步骤如下：

1．设置工程，在 Debug 页将波特率设置为 19200 上；

2．进入调试后全速运行程序，然后按 Debug->Stop Runing 停止运行，实际上这不会中断硬件电路的工作；

3．打开 PC 端串口调试软件，正确设置串口参数，即可正常工作。

第六章　C 语言编程综合练习

前面课程中我们学习了单片机 C 语言的基本知识，了解了单片机内部资源的 C 语言编程方法，这一节通过若干例子进一步学习 C 语言程序的有关知识点。

1．计数器

要求：编写一个计数器程序，将 T0 作为计数器来使用，对外部信号计数，将所计数字显示在数码管上。

该部分的硬件电路如图 6-1 所示，U1 的 P0 口和 P2 口的部份引脚构成了 6 位 LED 数码管驱动电路，数码管采用共阳型，使用 PNP 型三极管作为片选端的驱动，所有三极管的发射极连在一起，接到正电源端，它们的基极则分别连到 P2.0…P2.5，当 P2.0…P2.5 中某引脚输是低电平时，三极管导通，给相应的数码管供电，该位数码管点亮哪些笔段，则取决于笔段引脚是高或低电平。图中看出，所有 6 位数码管的笔段连在一起，通过限流电阻后接到 P0 口，因此，哪些笔段亮就取决于 P0 口的 8 根线的状态。

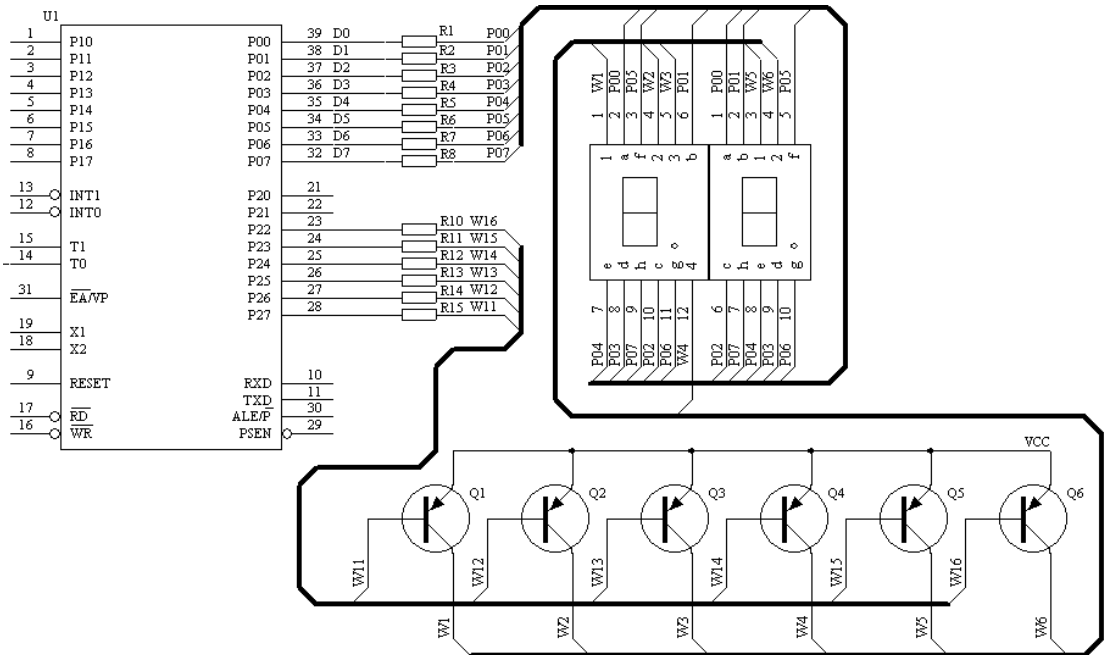


图 6-1　计数器实验硬件电路图

编写程序时，首先根据硬件连线写出 LED 数码管的字形码、位驱动码，然后编写程序如下：

```
#include "reg51.h"
#define uchar unsigned char
#define uint    unsigned int
uchar code BitTab[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB};   //位驱动码
uchar                                                                code
DispTab[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,0x88,0x83,0xC6,0xA1,0x8
6,0x8E,0xFF};   //字形码
uchar DispBuf[6];  //显示缓冲区
```

```
void Timer1() interrupt 3
{     uchar tmp;
      uchar Count;  //计数器，显示程序通过它得知现正显示哪个数码管
      TH1=(65536-3000)/256;
      TL1=(65536-3000)%256;     //重置初值
      tmp=BitTab[Count];      //取位值
      P2=P2|0xfc;   //P2 与 11111100B 相或
      P2=P2&tmp;  //P2 与取出的位值相与
      tmp=DispBuf[Count];//取出待显示的数      tmp=DispTab[tmp];       //取字形码
      P0=tmp;
      Count++;
      if(Count==6)
            Count=0;
}
void main()
{     uint  tmp;
      P1=0xff;
      P0=0xff;
      TMOD=0x15; //定时器 0 工作于计数方式 1，定时器 1 工作于定时方式 1
      TH1=(65536-3000)/256;
      TL1=(65536-3000)%256;     //定时时间为 3000 个周期
TR0=1;          //计数器 0 开始运行
TR1=1;
EA=1;
ET1=1;
for(;;)
{     tmp=TL0|(TH0<<8);   //取 T0 中的数值
      DispBuf[5]=tmp%10;
      DispBuf[4]=tmp%10;
      tmp/=10;
      tmp/=10;
      DispBuf[3]=tmp%10;
      tmp/=10;
      DispBuf[2]=tmp%10;
      DispBuf[1]=tmp/10;
      DispBuf[0]=0;
}
}
```

这个程序中用到了一个新的知识点，即数组，首先作一个介绍。

数组是 C51 的一种构造数据类型，数组必须由具有相同数据类型的元素构成，这些数据的类型就是数组的基本类型，如：数组中的所有元素都是整型，则该数组称为整型数组，如所有元素都是字符型，则该数组称为字符型数组。

数组必须要先定义，后使用，这里仅介绍一维数组的定义，其方式为：

类型说明符　数组名[整型表达式]

定义好数组后，可以通过：数组名[整型表达式]来使用数组元素。

在定义数组时，可以对数组进行初始化，即给其赋予初值，这可用以下的一些方法实现：

1．在定义数组时对数组的全部元素赋予初值：

例：int a[5]={1,2,3,4,5};

2．只对数组的部分元素初始化；

例：int　a[5]={1,2};

上面定义的 a 数组共有 5 个元素，但只对前两个赋初值，因此 a[0]和 a[1]的值是 1、2，而后面 3 个元素的值全是 0。

3．在定义数组时对数组元素的全部元素不赋初值，则数组元素值均被初始化为 0

4．可以在定义时不指明数组元素的个数，而根据赋值部分由编译器自动确定

例：uchar　BitTab[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB};

则相当于定义了一个 BitTab[6]这样一个数组。

5．可以为数组指定存储空间，这个例子中，未指定空间时，将数组定义在内部 RAM 中，可以用 code 关键字将数组元素定义在 ROM 空间中。

uchar code BitTab[]={0x7F,0xBF,0xDF,0xEF,0xF7,0xFB};

用这两种定义分别编译，可以看出使用了 code 关键字后系统占用的 RAM 数减少了，这种方式用于编程中不需要改变内容的场合，如显示数码管的字形码等是很合适的。

6．C 语言并不对越界使用数组进行检测，例如上例中数组的长度是 6，其元素应该是从 BitTab[0]~BitTab[5]，但是如果你在程序中写上 BitTab[6]，编译器并不会认为这有语法错误，也不会给出警告（其他语言如 BASCI 等则有严格的规定，这种情况将视为语法错误），因此，编程者必须自己小心确认这是否是你需要的结果。

程序分析：程序中将定时器 T1 用作数码管显示，通过 interrupt 3 关键字定义函数 Timer1() 为定时器 1 中断服务程序，在这个中断服务程序中，使用

TH1=(65536-3000)/256;

TL1=(65536-3000)%256;

来重置定时器初值，这其中 3000 即为定时周期，这样的写法可以直观地看到定时周期数，是常用的一种写法。其余程序段分别完成取位码以选择数码管、从显示缓冲区获得待显示数值、根据该数值取段码以点亮相应笔段等任务。其中使用了一个计数器，该计数器的值从 0~5 对应第 1 到第 6 位的数码管。

主程序的第一部分是做一些初始化的操作，设置定时器工作模式、开启定时器 T1、开启计数器 T0、开启 T1 中断及总中断，随后进入主循环，主循环首先用 unsigned int 型变量 tmp 取出 T0 中的数值，这里使用了“tmp=TL0|(TH0<<8);”这样的形式，这相当于 tmp=TH0*256+TL0，但比之于后一种形式，该方式可以得到更高的效，其后就是将 tmp 值不断地除 10 取整，这样将 int 型数据的各位分离并送入相应的显示缓冲区。

二、液晶显示

字符型液晶显示器用于显示数字、字母、图形符号。这类显示器均把 LCD 控制器、点

阵驱动器、字符存贮器等做在一块板上，再与液晶屏一起组成一个显示模块，因此，这类显示器安装与使用都较简单。

图 6-2 是 DSB-1A 实验板上字符型液晶的接口电路。要求编写程序从该显示器的第二行第 1 列开始显示“Hello World!”。

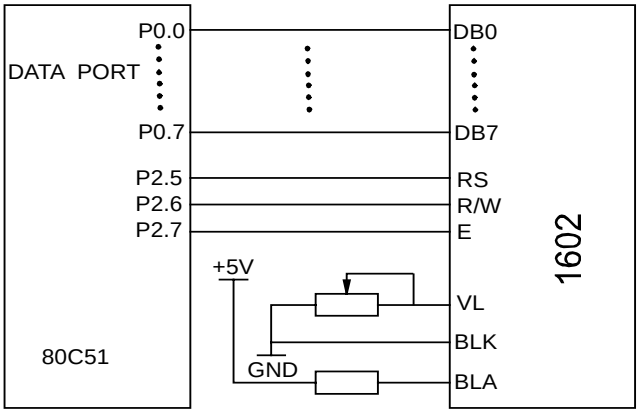


图 6-2　字符型液晶与单片机的接线图

由于市面上常见的字符型液晶驱动器均由 HD44780 或其兼容芯片构成，因此，这类液晶屏的驱动程序具有一定的通用性，这里给出用 C 语言写的驱动程序。在设置字符的起始行、列后，直接调用驱动程序提供的 WriteString 函数即可将字符串显示在指定的位置，使用非常简单。在熟悉了程序后，略作改动，可用于 2002、2004 等型号的液晶屏。

程序如下：

```
/*****************************************************
*       平凡单片机工作室
*       http://www.mcustudio.com
*       Copyright 2003 pingfan's McuStudio
*       All rights Reserved
*作者：周坚
*yj.c
*连线图:
*   DB0---DPROT.0   DB4---DPROT.4      RS-------------P2.5
*   DB1---DPROT.1   DB5---DPROT.5      RW-------------P2.6
*   DB2---DPROT.2   DB6---DPROT.6      E--------------P2.7
*   DB3---DPROT.3   DB7---DPROT.7      VLCD 接 10K 可调电阻到 GND*
*80C51 的晶振频率为 12MHz
*液晶显示程序
*****************************************************/
#include "reg51.h"
#include<absacc.h>
#include <intrins.h>
#define DPORT   P0
#define uchar unsigned char
sbit  RS  =    P2^5;
sbit  RW  =    P2^6;
sbit  E   =    P2^7;
```

```
uchar Xpos;          //列方向地址指针
uchar Ypos;          //行方向地址指针

#define NoDisp      0
#define NoCur1
#define CurNoFlash      2
#define CurFlash    3
/*延时程序
   由 Delay 参数确定延迟时间
*/
void LcdWcn(uchar);
void LcdWc(uchar);
void WriteChar(uchar);
void LcdPos();
void LcdWd(uchar);
void LcdWdn(uchar);

void mDelay(unsigned int Delay)
{     unsigned int i;
      for(;Delay>0;Delay--)
      {     for(i=0;i<124;i++)
            {;}
      }
}

/*光标设置命令
Cur  为设定光标参数
*/
void SetCur(uchar Cur)
{     switch(Cur)
      {     case 0x0:
            {     LcdWc(0x08);//关显示
                  break;
            }
            case 0x1:
            {     LcdWc(0x0c);//开显示但无光标
                  break;
            }
            case 0x2:
            {     LcdWc(0x0e);//开显示有光标但不闪烁
                  break;
            }
            case 0x3:
```

```
            {   LcdWc(0x0f); //开显示有光标且闪烁
                break;
            }
            default: break;
        }
}
/*清屏命令
*/
void ClrLcd()
{   LcdWc(0x01);
}
/*在指定的行与列显示
*/
void WriteChar(uchar c)
{   LcdPos();
    LcdWd(c);
}
/*正常读写操作之前检测 LCD 控制器
*/
void WaitIdle()
{   uchar tmp;
    DPORT=0xff;
    RS=0;
    RW=1;
    E=1;
    _nop_();
    for(;;)
    {   tmp=DPORT;
        tmp&=0x80;
        if(tmp==0)
            break;
    }
    E=0;
}

/*不检测忙的写字符子程序
*/
void LcdWdn(uchar c)
{
    RS=1;
    RW=0;
    DPORT=c;    //写入待写字符
    E=1;
    _nop_();
```

```
    E=0;
}
/*带忙检测的写字符子程序
*/
void LcdWd(uchar c)
{   WaitIdle();
    LcdWdn(c);
}

/*检测忙信号的送控制字子程序*/
void LcdWcn(uchar c)
{   RS=0;
    RW=0;
    DPORT=c;
    E=1;
    _nop_();
    E=0;
}
/*检测忙信号的送控制字子程序*/
void LcdWc(uchar c)
{   WaitIdle();
    LcdWcn(c);
}
void LcdPos()
{   uchar tmp;
    Xpos&=0x0f;  //16xx 型液晶的范围是 0~15
    Ypos&=0x01; //Y 的范围是 0~1
    tmp=Xpos;
    if(Ypos==1)
    {   tmp+=0x40;
    }
    tmp|=0x80;
    LcdWc(tmp);
}

/*LCD 的复位程序
*/
void RstLcd()
{   mDelay(15);   //延时 15ms
    LcdWcn(0x38);
    mDelay(5);
    LcdWcn(0x38);
    mDelay(5);
    LcdWcn(0x38);
```

```
    LcdWc(0x38);
    LcdWc(0x08);
    LcdWc(0x01);
    LcdWc(0x06);
    LcdWc(0x0c);
}

void WriteString(char s[])
{   uchar pS=0;
    for(;;)
    {   WriteChar(s[pS]);
        pS++;
        if(s[pS]==0)
            break;
        if(++Xpos>=15)    //每行最多显示 16 个字符
            break;
    }
}

void main()
{   uchar s1[]="Hellow World!";
    RstLcd();//复位 LCD
    ClrLcd();
    SetCur(CurFlash);  //光标显示且闪烁
    Xpos=2;
    Ypos=1;
    WriteString(s1);
    for(;;)
    {;}
}
```

程序分析：本程序中大量使用了函数，在此对函数的功能作一个简介。

C 语言程序是由一个个函数构成的，从函数定义的形式上划分，函数有三种形式：无参数函数、有参数函和空函数。

无参数函数的定义形式为：

返回值类型识别符　函数名()｛函数体语句｝

如本例中的 void WaitIdle(){ …… }就是一个无参数函数

有参数函数的定义形式为：

返回值类型识别符　函数名(形式参数列表)｛函数体语句｝

如本例中的 void LcdWdn(uchar c){　…… }就是一个有参数的函数

函数可以返回一个值，也可以什么值也返回，如果函数要返回一个值，在定义这个函数时要定义好这个值的数据类型，这里所说的数据类型就是指前面课程中介绍到的 int、char、float 等类型，如果在定义函数时没有定义返回值的类型，系统默认为返回一个 int 型的值。如果明确地知道一个函数将没有返回值，可以将其定义为 void 型，这样，如果在调用函数

时错误地使用了“变量名＝函数名”的方式来调用函数，编译器就能发现这一错误并指出。本例中就大量地应用到了 void 型函数。

C 语言采用函数之间的参数传递方式，这使得一个函数能对不同的变量进行功能相同的处理，使函数具有了通用性。定义函数时，写在函数名括号中的称之为形式参数，而在实际调用函数时写在函数括号中的称之为实际参数。本例中：

```
void SetCur(uchar Cur)
{    …}
```

函数中 Cur 就是一个形式参数，而在主函数中调用时写的：

```
SetCur(CurFlash);
```

其中 CurFlash 就是一个用符号常量表示的实际参数，在执行该函数时该值被传递到函数内部并执行。

每一个函数所调用的函数必须已被定义，否则就会出现语法错误，因此程序中一般要求在程序的开头对程序中用到的函数进行统一的说明，然后再分别定义有关函数，本例中有：

```
void WriteChar(uchar);
…
void LcdWdn(uchar);
```

就是首先在程序的前方写一个有关函数的说明，而真正的函数定义则在程序放在后部。但细心的读者可能发现有一些函数并未写其说明，而是直接在程序中定义了，如 mDelay 函数，这是为何呢？这是因为这些函数出现在程序的前面，在还没有任何函数调用它们之前它们就被定义了，因此就不需要再单独写一个函数说明。读者可将 mDelay 函数的定义移到程序的后面位置，再次编译就会出错。当然，好的编程习惯是不论函数在何处被定义，总是在写前面写一个函数说明。

有关单片机的 C 语言编程到此就告一个段落，虽然 C 语言很多特性尚未介绍，但通过上面有关内容的学习，我们已经可以使用 C 语言进行一些实际的工程开发工作，大家可以在工作中继续学习有关 C 语言的知识。