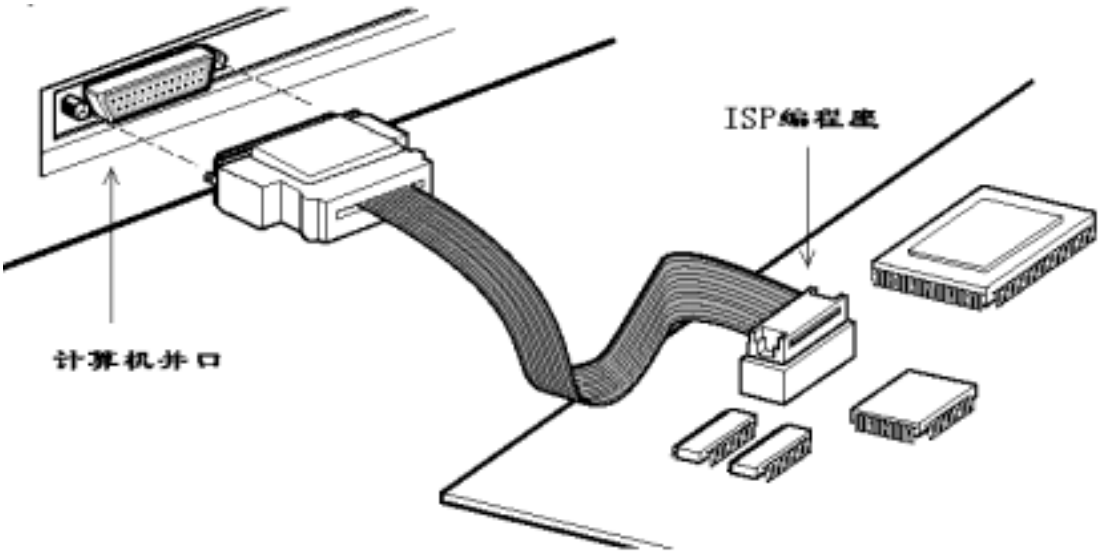
51 单片机 C 语言入门教程

学习单片机前的准备

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. 传统的单片机学习方案是"编程器+仿真器+试验板", 采用此方案的弊端是一次性投入太高，在学习或开发项目时，需要反复不停地拔插电缆、芯片、电源等。这样将大大降低学习和开发效率，如稍有不慎就有可能造成器件和设备的损坏，给使用者带来不必要的麻烦或损失。

现在这些问题都得到了比较好的解决。如在众多单片机中 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.典型的芯片是 AT89S51，这种芯片的特点是可以实现在系统编程（In-System-Programming)，即 ISP，这样，学习单片机就不需要购买高价格的编程器了，只要一根价格低廉的下载线即可，方法是在电路板上留下一个 10 芯的下载线插座，把相关的 io 口与之相连即可。如下图所示：



本下载线可以自制，网路上有很多原理图，是用 244 芯片的。这里介绍的是用 CPLD 芯片制造的多功能的下载线，本产品不但可以下载 ATMEL 的单片机，还可以下载 PIC 的部分单片机以及 CPLD/FPGA，ARM9 等等的后继学习芯片，性价比一流，并申请了“ 并口盒内的多功能免跳线 ISP 下载线 ”专利技术。相关资料，请浏览网页

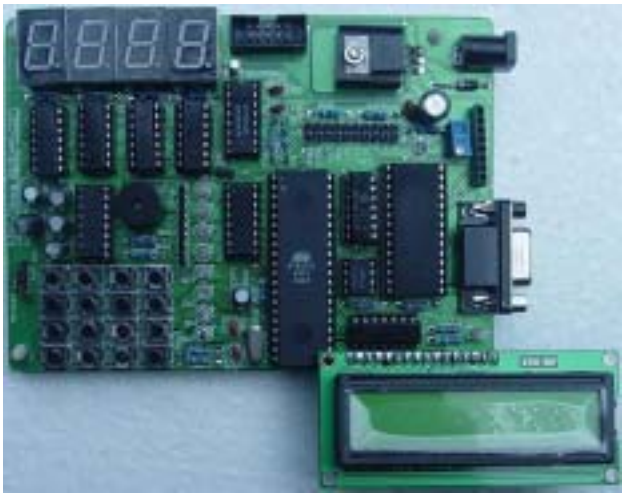
http://www.ispdown.com/showoneproduct.asp?productid=15

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. 因此，我们以 C 语言来讲解单片机的开发方法。

本文介绍一种单片机学习开发系统，它将控制软件、单片机实验板、串行下载编程器进行完美的结合。试验过程中无需拔插任何电缆和芯片，也无需切换电源。配合专门开发的 ispdown 下载线和配套程序可轻松地将编绎好的代码下载到实验板上进行验证或演示，整个过程只需利用鼠标操作即可，方便快捷。该开发系统摆脱了传统、繁锁的单片机学习方式，

将当今最流行、最经济有效的学习方案完美地集成在了同一个系统中，是一个快捷、高效、灵活的单片机学习、开发方案。有了这个开发系统就同时拥有了学习实验板、串行编程器，具有非凡的性价比，是单片机爱好者快速掌握 51 系列单片机不可多得的工具。实验板如图：

本实验板已经预留了 isp 下载插座，处于上方中部的 10 芯插座即是。详细资料请浏览网页：



http://www.ispdown.com/showoneproduct.asp?productid=32

在这里，还给各位单片机初学者提供了实验板上的所有源程序（C 语言），力求让初学者快速入门。下载地址为：

http://www.ispdown.com/SoftView.asp?SoftID=196

（含实验板原理图和 PCB 版图）

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)

图 1－1 启动时的屏幕

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.

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 KEILuVision2 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.

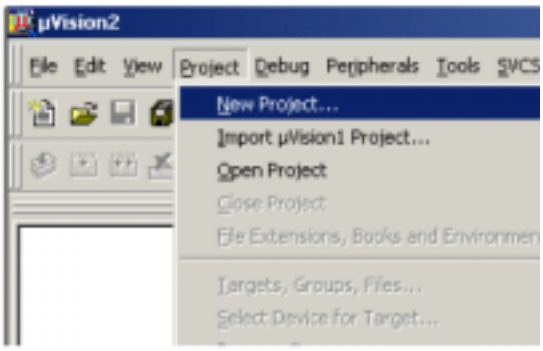


图 1－2 New Project 菜单

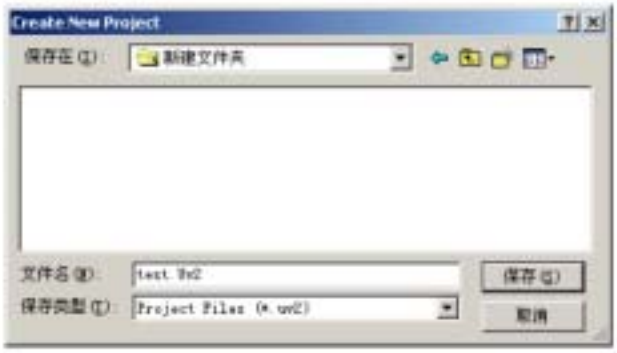


图 1－3 文件窗口

(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.AT 8 9 C 5 1 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<reg51.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
```

```
    }
}
```

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.

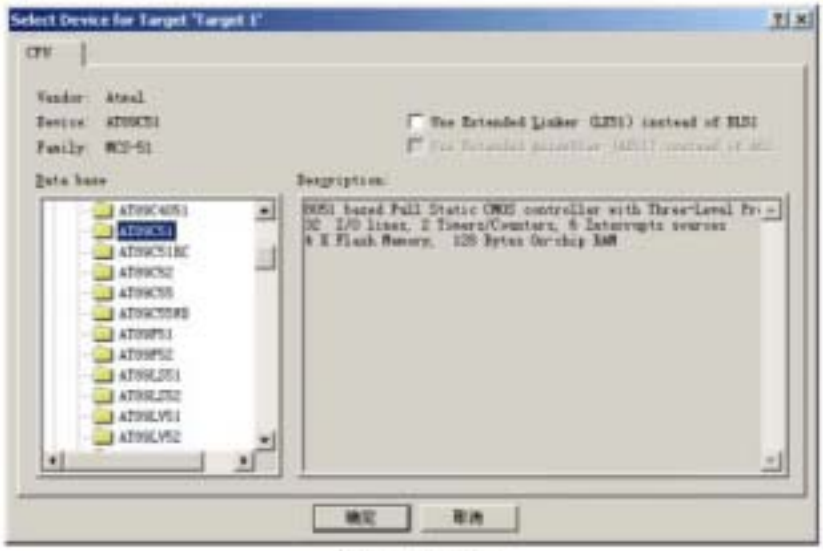


图 1－4 选取芯片

(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 1－5 新建程序文件

(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.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

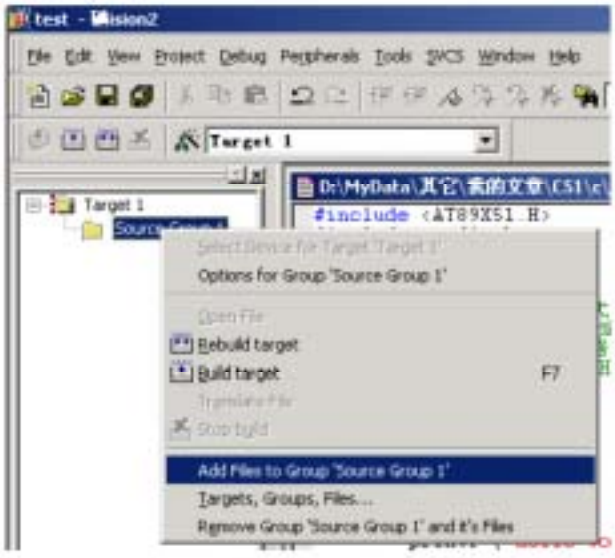


图 1－6 把文件加入到项目文件组中

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.

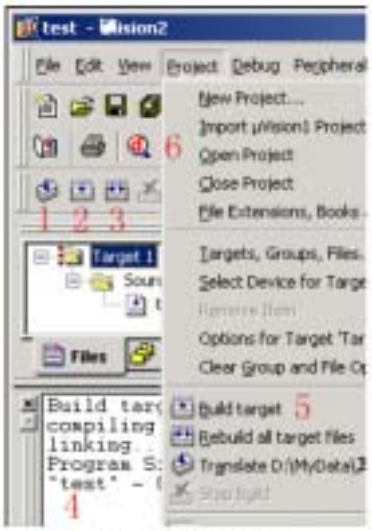


图 1－7 编译程序

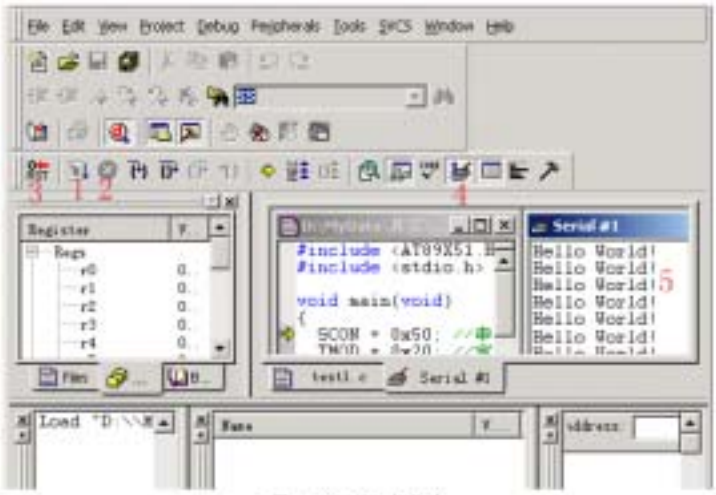


图 1－8 调试运行程序

(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.

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。这里介绍的是 ATMEL 公司的 AT89S51 芯片，价格便宜，功能强大，简单易学，最重要的是它有了 isp 技术。下面将介绍如何应用 isp 技术下载程序而不需要编程器的方法。

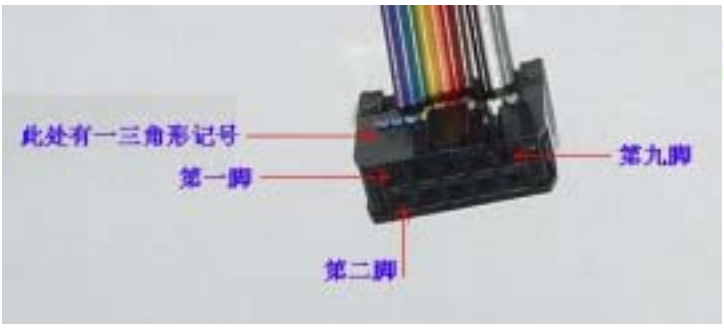
在 AT89S51 中，P1 引脚有几个复用的引脚。

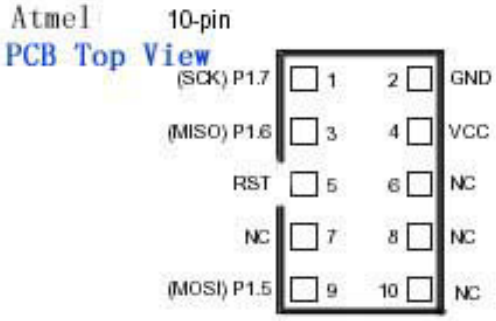
P15：MOSI

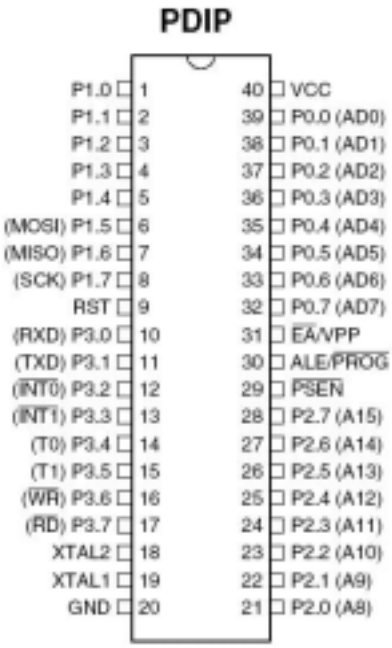
P16：MISO

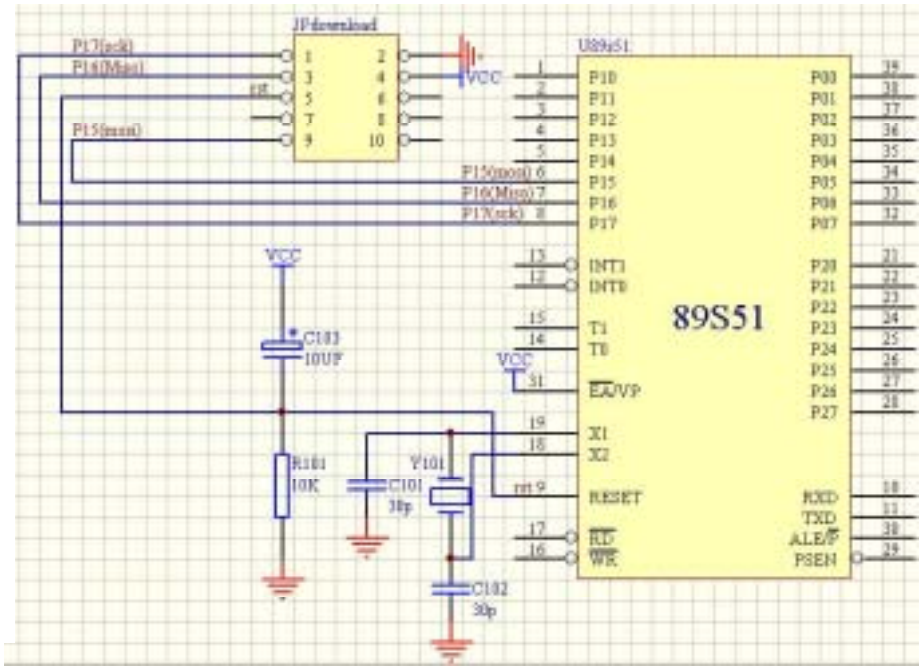
P17：SCK

连同 VCC，GND，RESET 六根线，连接到 10 芯下载线，这样当目标板完成后，就可以运用下载线来下载程序了。10 芯下载线的定义如下：









最后，能够下载程序的 S51 的最小系统如下图：

芯片引脚如下图所示：

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

XTAL1 19

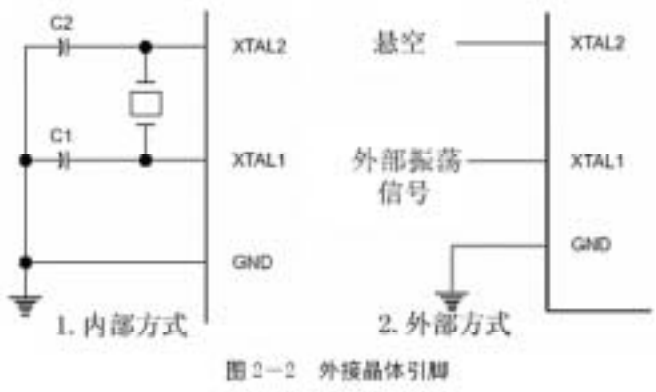


图 2－2 外接晶体引脚

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 AT 8 9 C 5 1 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.

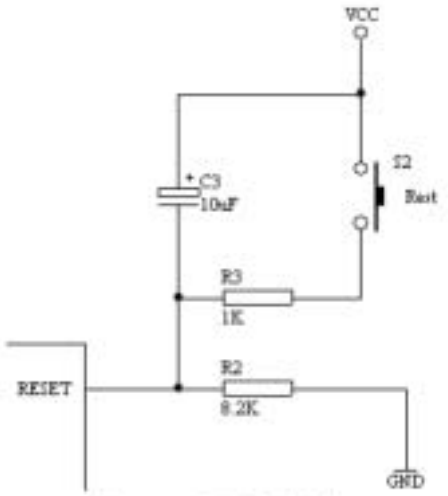


图 2－3 常用复位电路

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

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（$\overline{INT0}$）
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

的电流.

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

(5) 其它的控制或复用引脚

1、 ALE/PROG 30 访问外部存储器时,ALE(地址锁存允许)的输出用于锁存地址的低位字节.即使不访问外部存储器,ALE 端仍以不变的频率输出脉冲信号(此频率是振荡器频率的 1/6).在访问外部数据存储器时,出现一个 ALE 脉冲.对 Flash 存储器编程时,这个引脚用于输入编程脉冲 PROG

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

3、 EA/Vpp 31 外部访问允许端.当该引脚访问外部程序存储器时,应输入低电平.要使 AT89C51 只访问外部程序存储器(地址为 0000H-FFFFH),这时该引脚必须保持低电平.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.

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.

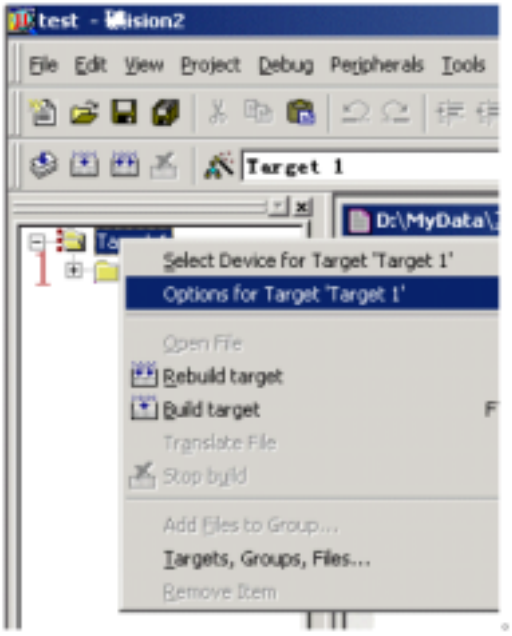


图3－1项目功能菜单

(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.)

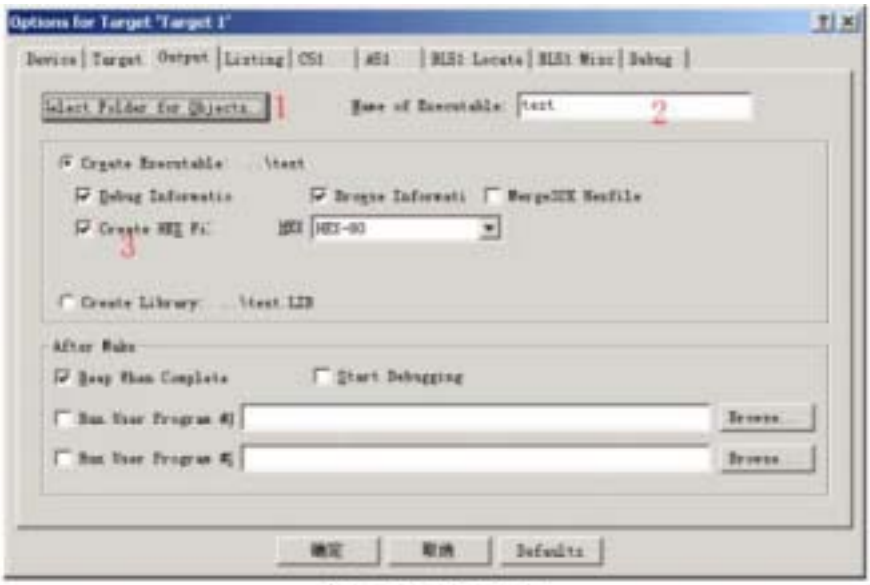


图3—2 项目选项窗口

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.

下面再以一实例程序验证系统是否在运行,这个系统也易于自制用于实验.图 3-4 便是 AT89S51 的 LED 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.

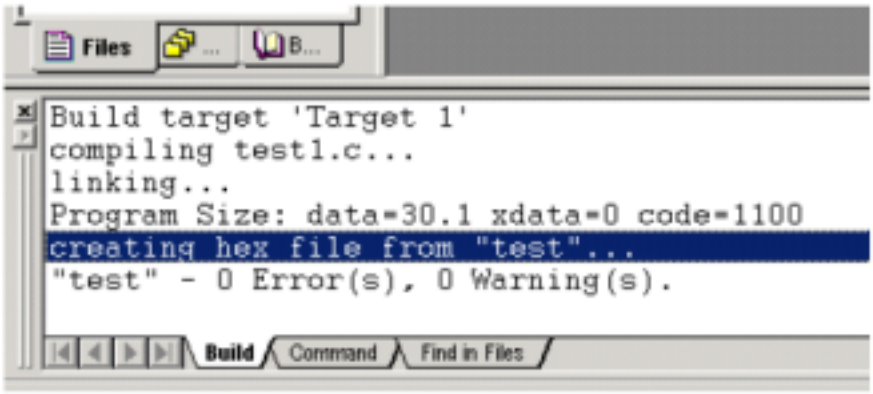


图 3—3 编译信息窗口

下面,我们建一个新的项目名为 LED 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.ispdown.com).程序如下:

```
#include<reg51.h>
//---------------------------------
void delay400ms(void)
  {
    unsigned char TempCycA = 5;
    unsigned int TempCycB;
    while(TempCycA--)
    {
      TempCycB=7269;
      while(TempCycB--);
```

```
    };
  }
//----------------------
main()
{
  while(1)
  {
    P1=0xff;
    delay400ms();
    delay400ms();
    P1=0x00;
    delay400ms();
    delay400ms();
  }
}
```

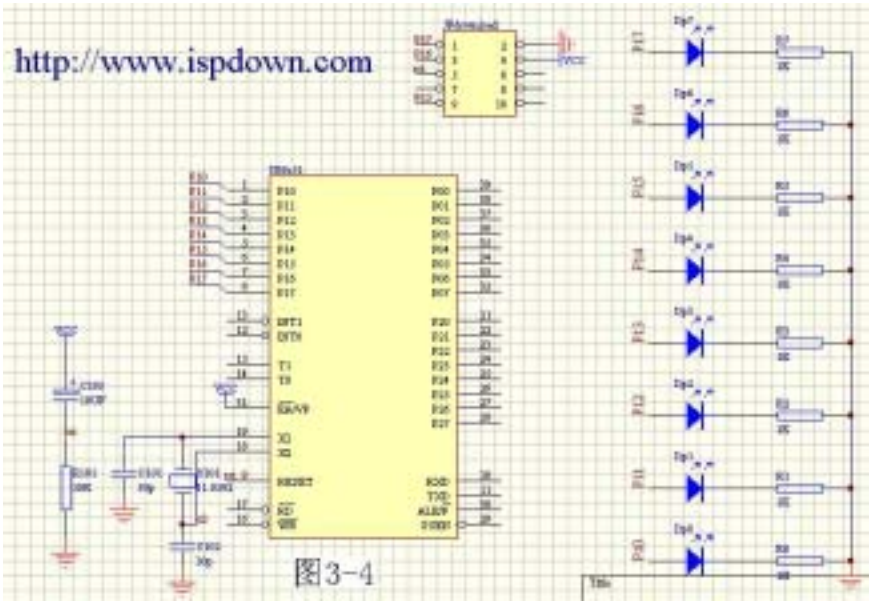


图3-4

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 LED 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,LED 灭,再延时,LED 亮,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 3-4，电路图不变.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.

实验程序如下:

```
#include<reg51.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,熄灭 LED
      P1^0 = 1; //设 P1.0 口为高电平,点亮 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，熄灭 LED
    P1^1 = 1; //设 P1.1 口为高电平,点亮 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 3-4 所示，新建一个 LED 的项目,主程序如下:

```
#include<reg51.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 AT89S51 的 P1 引脚为低电平才会点亮,所以我们要向 P1 口的各引脚写数据 O 对应连接的 LED 才会被点亮,P1 口的八个引脚刚好对应 P1 口特殊寄存器的八个二进位,如向 P1 口定数据 0xFE,转成二进制就是 11111110,最低位 D0 为 0 这里 P1.0 引脚输出低电平,LED1 被点亮.如此类推,大家不难算出自己想要做的效果了.大家编译烧写看看,效果就出来,显示的速度您可以根据需要调整延时 a 的值,不要超过变量类型的值域就很行了.哦,您还没有实验板?那如何可以知道程序运行的结果呢?呵,不用急,这就来说说用 KEIL uVision2 的软件仿真来调试 IO 口输出

输入程序.

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.

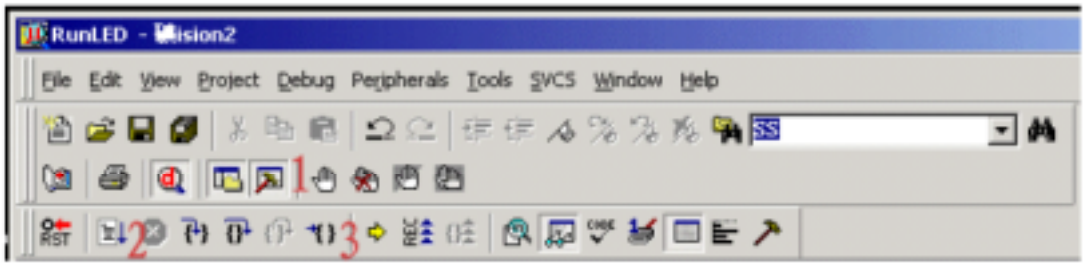


图 5－2　调试用快捷菜单栏

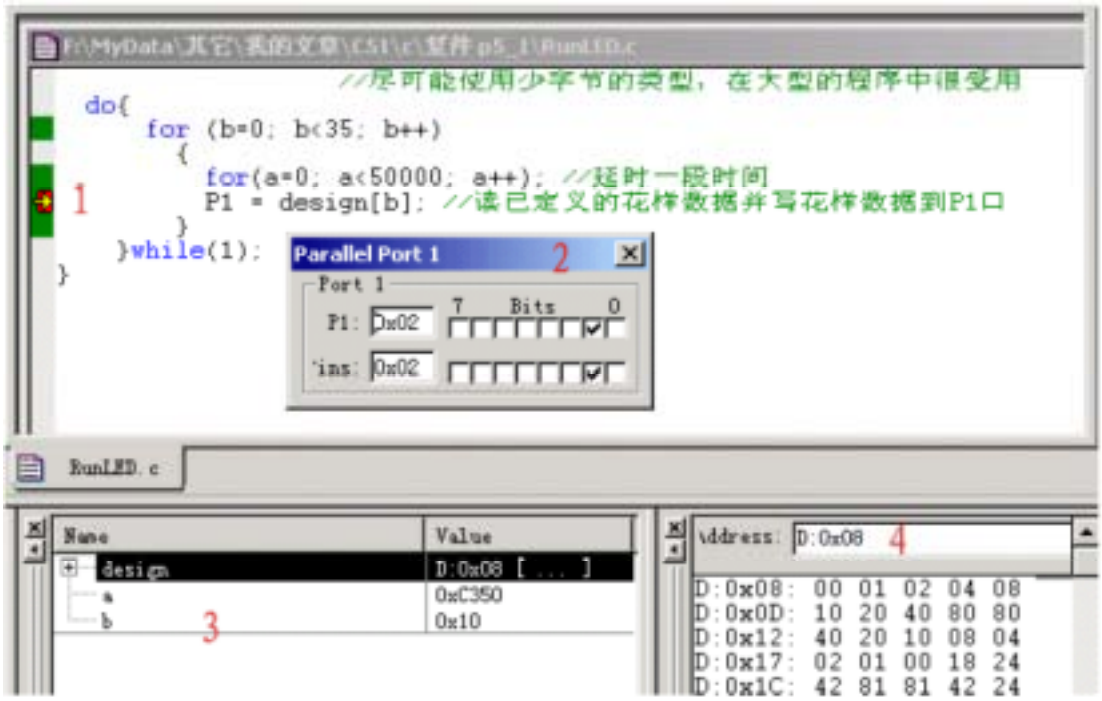


图 5－3　各调试窗口

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).这些存储种类的具体含义和用法,将在第七课《变量的存储》中进一步进行学习.

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

地址表请看附录二 AT89C51 特殊功能寄存器列表

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

表 6－1 存储器类型

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

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

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

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

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

sfr 和 sfr16 可以直接对 51 单片机的特殊寄存器进行定义,定义方法如下:

sfr 特殊功能寄存器名= 特殊功能寄存器地址常数;

sfr16 特殊功能寄存器名= 特殊功能寄存器地址常数;

我们可以这样定义 AT89C51 的 P1 口

sfr P1 = 0x90; //定义 P1 I/O 口,其地址 90H

sfr 关键定后面是一个要定义的名字,可任意选取,但要符合标识符的命名规则,名字最好有一定的含义如 P1 口可以用 P1 为名,这样程序会变的好读好多.等号后面必须是常数,不允许

有带运算符的表达式,而且该常数必须在特殊功能寄存器的地址范围之内(80H-FFH),具体可查看附录中的相关表.sfr 是定义 8 位的特殊功能寄存器而 sfr16 则是用来定义 16 位特殊功能寄存器,如 8052 的 T2 定时器,可以定义为:

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

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

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

(1) sbit 位变量名=位地址

sbit P1_1 = Ox91;

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

(2) sbit 位变量名=特殊功能寄存器名^位位置

sft P1 = 0x90;

sbit P1_1 = P1 ^ 1; //先定义一个特殊功能寄存器名再指定位变量名所在的位置,当可寻址位位于特殊功能寄存器中时可采用这种方法

(3) sbit 位变量名=字节地址^位位置

sbit P1_1 = 0x90 ^ 1;

这种方法其实和 2 是一样的,只是把特殊功能寄存器的位址直接用常数表示. 在 C51 存储器类型中提供有一个 bdata 的存储器类型,这个是指可位寻址的数据存储器,位于单片机的可位寻址区中,可以将要求可位录址的数据定义为 bdata,如:

unsigned char bdata ib; //在可位录址区定义 ucsigned char 类型的变量 ib

int bdata ab[2]; //在可位寻址区定义数组 ab[2],这些也称为可寻址位对象

sbit ib7=ib^7 //用关键字 sbit 定义位变量来独立访问可寻址位对象的其中一位

sbit ab12=ab[1]^12;

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

下面我们用上一课的电路来实践一下这一课的知识.同样是做一下简单的跑马灯实验,项目名为 LED.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);
}
```